The Science of Foundation Models for Science

By

Nicholas Roberts

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2026

Date of final oral examination: May 5th, 2026

The dissertation is approved by the following members of the Final Oral Committee:

Frederic Sala, Assistant Professor, Computer Sciences
Aws Albarghouthi, Associate Professor, Computer Sciences
Junjie Hu, Assistant Professor, Computer Sciences
Dimitris Papailiopoulos, Associate Professor, Electrical and Computer Engineering
Xiaojin (Jerry) Zhu, Professor, Computer Sciences

To my parents and Risa.

Acknowledgments

My Ph.D. journey would not have been possible without the endless support from my family, friends, labmates, and mentors. There are many people who deserve thanks in this acknowledgements section, but in keeping with the brevity of the rest of this dissertation, I have made an attempt at a summary below.

First, I want to express my deepest gratitude to my dissertation advisor, Professor Frederic Sala, for taking a chance on admitting me to his lab, and for supporting me through the countless times I came to him with silly and anxious questions over the course of half a decade. Thank you Fred. I also want to thank the many other faculty who helped me get to this point, including my committee: Aws Albarghouthi, Junjie Hu, Dimitris Papailiopoulos, and Jerry Zhu, and all of the other faculty at the University of Wisconsin-Madison. This list extends far beyond Madison: I want to thank Christopher Ré for graciously hosting me in his lab when I visited Stanford, Ameet Talwalkar and Zachary Lipton for shaping my early research vision during my M.S. at Carnegie Mellon University, as well as Gary Cottrell and Sanjoy Dasgupta for patiently teaching me the ways of research as an overly enthusiastic undergraduate at the University of California, San Diego. I also want to thank Mikhail Khodak, who mentored me during my M.S., and continued to support me as a mentor and friend when he joined the University of Wisconsin-Madison as faculty.

I owe just as much gratitude to my labmates in the Sprocket Lab: Changho Shin, Dyah Adila, Harit Viswakarma, Brian Huang, Jitian Zhao, Gabe Orlanski, Albert Ge, Zhiqi Gao, Avi Trost, Sam Guo, June Cho, John Cooper, Kendall Park, Jiayu Wang, and Sonia Cromp, as well as our amazing undergrads, including: Kaylee Li, Chris Cai, and Albert Wu. We had some unforgettable times in Madison and at conferences–specifically the NeurIPS editions that were held in New Orleans. I also owe many thanks to my friends from the Astronomy department and the folks I met through them–Roark Habegger, Patrick Gorman, Isaac Laseter, and Lizhou Sha–for all of the fun and good beer throughout my Ph.D., and

to many others in the CS department: Vinay Banakar, Sushma Ambekar, Sujay Yadalam, Bobbi Yogatama, Konstantinos Kanellis, Jihye Choi, and Siddharth Suresh, for all of the great times and brunches we've had. Thank you all for making my Ph.D. journey so much more fun–thanks to all of you, I've finished with (most of) my sanity intact. Your research insights, collaborations, and friendship grew me into the researcher that I am today.

I have also made many friends and learned from several mentors during my time interning at Meta, Together AI, and Microsoft Research. I would like to thank Dieuwke Hupkes for being such an amazing internship host during my time at Meta—a lovely time in which I had a unique opportunity to explore the UK and Europe for six months—and for introducing me to research on scaling laws, which ended up becoming a central research pillar of this dissertation. I made many friends during that time, who also continue to be amazing collaborators: Deepak Nathani, Rishi Hazra, Lovish Madaan, and many more. I will never forget our road trip to the Isle of Skye. I would like to thank Tri Dao, Avner May, and Ben Athiwaratkun for being incredible collaborators and mentors during my internship at Together AI. I would also like to add that I first met Tri during a collaboration during my M.S. (I obviously learned a lot from him then as well), and that was partly how I ended up connecting with Fred (they knew each other from Stanford; it is a small world, it turns out). I met many friends at Together as well, who deserve thanks here–Ted Zadouri, Berlin Chen, Krithik Ramesh, Rahul Chalamala, and Jason Wiemels. I learned a lot from all of you, and you made that summer in San Francisco all the more memorable. My first internship during my Ph.D. was at Microsoft Research in Redmond, Washington (I made a road trip out of it from Wisconsin). I was fortunate to be mentored by Mojan Javaheripi, and collaborated closely with Yi Zhang and Ronen Eldan. I was also extremely fortunate to make some amazing friends during this internship, including Asher Trockman (who I had met previously in Rwanda), Mayee Chen (who I knew previously through Fred), Sadhika Malladi, Jaume de Dios Pont, and Bingbin Liu. This was an unforgettable summer in Seattle, and I learned so much from all of you.

Before starting my Ph.D., I was an M.S. student in the Machine Learning Department at Carnegie Mellon University. I would like to thank my entire cohort for making that experience such a memorable one, especially during the challenging times that were the COVID-19 pandemic. I would like to give specific thanks to my friends who I would study with until the wee hours of the morning in the 8th floor kitchenette: Euxhen Hasanaj, Vishwak Srinivasan, and to Shantanu Gupta, who is dearly missed. I would also like to

thank Jeffrey Li and Elan Rosenfeld for all of the great times while living together during the pandemic, and Tanya Marwah for the career advice and chats later on in the final year of my Ph.D. I was fortunate to collaborate and learn from Jeff throughout my Ph.D. as well, and share many fun times at conferences and trips. I would also like to thank Vinay Prabhu for taking me on as an intern at UnifyID between my undergraduate years and my M.S., which was a formative experience in my research career and an incredibly memorable internship.

Finally, I want to thank my family, Bruce and Laurie Roberts, and my love, Risa Dickerman. I literally would not be here without my parents' support at every stage that brought me to where I am today–I couldn't list all of the ways in which you've supported me, but if I could, I'm sure no book bindery would have enough paper to print this dissertation. Risa, you have supported me through the most logistically, psychologically, and intellectually challenging parts of my Ph.D., and have done so with unwavering grace. Thank you all for believing in me and for investing so much in me. This dissertation is for you.

Contents

List of Tables

List of Figures

Abstract

Foundation models are poised to transform scientific discovery, yet their development remains ad-hoc—driven by rules of thumb and closed industrial pipelines rather than systematic understanding. This dissertation pursues a science of foundation models, organized around three research directions: principled scaling recipes, data efficient learning, and adaptation to scientific domains.

In Part I, we develop scaling laws that go beyond standard prescriptions. We introduce Train-to-Test scaling laws that jointly optimize pretraining and inference-time compute, showing that optimal pretraining shifts radically into the overtraining regime when test-time scaling is accounted for. We demonstrate that compute-optimal allocations are skill-dependent—knowledge tasks favor larger models while reasoning tasks favor more data—and we present Manticore, a system for automatically constructing pretrained hybrid architectures from different model families.

In Part II, we address the challenge of learning from limited data. We introduce Skill-it, an online data selection algorithm that reduces pretraining data requirements by up to $3\times$ by exploiting prerequisite dependencies between skills. We develop Loki, a training-free adaptation method that enables pretrained models to predict unseen classes by leveraging geometric structure, improving zero-shot performance by up to 19.5%. We also construct AutoWS-Bench-101, a benchmark revealing that automated weak supervision must incorporate foundation model signal to outperform simple baselines.

In Part III, we build evaluation methods and neural primitives for diverse scientific tasks. We introduce the AUP score for principled cross-task evaluation, construct benchmarks spanning PDEs, audio, biological signals, and more. Finally, we develop XD-operations—a search space capable of discovering novel domain-tailored neural operations that outperform expert-designed architectures on PDE solving, protein folding, and music modeling.

Chapter 1

Introduction

Foundation models promise to transform how we do science: how we discover new drugs, design novel materials, and solve longstanding mathematical problems, among a growing list of applications. While the advancements that have led us to the cusp of this transformation are extraordinary, the development of the most capable models remains confined to industrial laboratories. The research that drives these models forward is often closed and non-systematic—resembling alchemy more than science, with recipes passed around via hearsay or discovered by reverse engineering other models' outputs. This ad-hoc approach is a barrier not only to scientific understanding of these models, but also to their effective application in the scientific domains where they could have the greatest impact.

This dissertation takes steps toward a more principled foundation. We pursue what may be termed a *science of foundation models*—an effort to replace rules of thumb with systematic understanding—organized around three interrelated research directions:

1. Scaling recipes for foundation models (Part I). For a given compute budget, how should we allocate resources across model size, training data, and inference-time computation, and how do these decisions depend on the capabilities we want the model to have? We develop scaling laws that go beyond one-size-fits-all prescriptions by accounting for specific skills, test-time scaling, and architectural design choices.

2. Data efficient learning (Part II). As model capabilities reach the frontier of human ability, the amount of relevant training data decreases. We develop methods to squeeze as much usable information as possible from limited data, through optimized data selection, structured knowledge, and automated supervision.

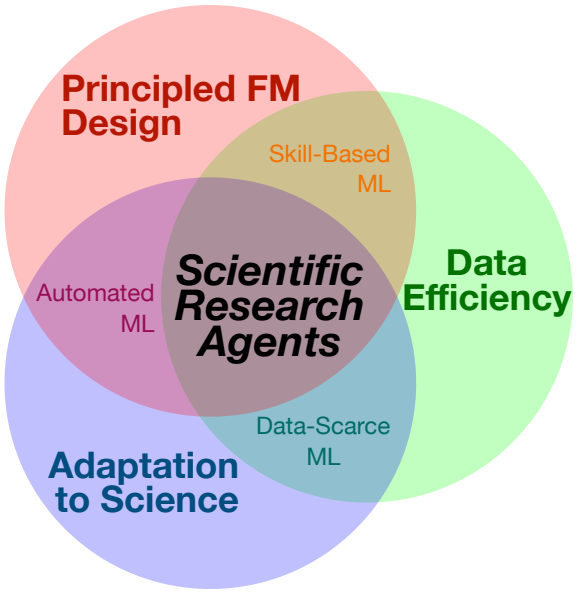


Figure 1.1: Overview of the three research directions in this dissertation and their intersections. Principled foundation model design, data efficiency, and adaptation to scientific domains each contribute independently but also interact: skill-based methods sit at the intersection of principled design and data efficiency, automated ML connects principled design with scientific adaptation, and data-scarce ML bridges data efficiency with scientific domains. Together, they converge on the long-term vision of scientific research agents.

3. Adaptation to scientific domains (Part III). Standard machine learning techniques prioritize well-studied domains like vision and language, leaving scientific applications underserved. We build new evaluation approaches, design methods, and neural primitives that explicitly tackle the challenges of diverse scientific tasks.

These three directions are deeply interconnected, as illustrated in Figure 1.1. Understanding how foundation models scale (Part I) informs how to make the most of limited data (Part II); both, in turn, are prerequisites for adapting these models to domains beyond their original training distribution (Part III). At the intersections, we find recurring themes that cut across the dissertation: *skill-based* methods connect principled FM design with data efficiency (Chapters 3 and 5), *automated ML* techniques bridge principled design with scientific adaptation (Chapters 4, 9, and 11), and *data-scarce ML* links data efficiency with scientific domains where labeled data is rare (Chapters 6 and 7). Together, these directions aim to make foundation model development more transparent, more efficient, and more broadly applicable to humanity's greatest scientific challenges.

In the remainder of this chapter, we provide additional motivation for these research directions (§1.1) and a detailed outline of the contributions in each chapter (§1.2).

1.1 Motivation

1.1.1 The Need for Principled Scaling Recipes

The development of large language models and other foundation models has followed a remarkably consistent pattern: scale up the model, scale up the data, and hope that new capabilities emerge. Scaling laws—predictive relationships that forecast optimal design choices for a given compute budget—have provided rough guidance for this process. The most influential of these, the Chinchilla scaling law [Hoffmann et al., 2022a], prescribes a roughly linear relationship between optimal model size and the number of training tokens, superseding earlier work suggesting that models should be scaled faster than data [Kaplan et al., 2020a]. These recipes have guided the development of models from Llama [Touvron et al., 2023a] to Gemma [Team et al., 2024], becoming some of the most consequential findings in modern machine learning.

However, current scaling recipes suffer from two critical limitations. First, deriving them requires large-scale experiments that only a handful of industrial laboratories can afford. A typical Chinchilla-style analysis involves training dozens to hundreds of models at multiple compute scales, costing millions of dollars in compute—making this line of research inaccessible to most of the academic community. Second, existing recipes are *blunt instruments*: they treat all model capabilities uniformly, optimizing a single aggregate metric (usually validation loss on a generic held-out corpus) while ignoring factors that practitioners actually care about. These include the specific skills a model must learn (e.g., factual recall versus multi-step reasoning), the composition of training data across different stages of development, the interplay between pretraining and inference-time computation (e.g., repeated sampling or chain-of-thought), and architectural design choices (e.g., whether to use Transformer attention, state space models, or hybrids thereof).

As a result, practitioners are left with coarse guidance that often leads to suboptimal models. A model optimized for aggregate validation loss may be excellent at one skill but poor at another. A model trained to Chinchilla-optimal specifications may be poorly suited for deployment with test-time scaling, where smaller, more overtrained models

can outperform their larger counterparts. Similarly, a model built on a fixed Transformer architecture may miss efficiency gains available from hybrid designs. The only recourse under current recipes is to scale further to paper over these inefficiencies—an increasingly expensive proposition.

Ideally, we would have scaling recipes that are both *accessible*—requiring only modest compute to derive—and *fine-grained*, accounting for the many variables beyond model and dataset size that impact final model capabilities. Part I of this dissertation takes steps in this direction by developing scaling laws that integrate test-time compute, skill-dependent behavior, and techniques for architectural composition that bypass scaling entirely.

1.1.2 The Data Efficiency Challenge

Much of the progress in foundation model capabilities has exploited data that—luckily—happened to be available for independent reasons. Base language models took advantage of Internet-scale text corpora accumulated over decades of human web activity [Villalobos et al., 2024]. More recent reasoning models benefit from tens of thousands of mathematical competition problems developed over centuries of human mathematical culture [Hendrycks et al., 2021b]. Vision-language models like CLIP [Radford et al., 2021] were similarly trained on hundreds of millions of image-text pairs scraped from the web.

However, as model capabilities approach the frontier of human ability, the amount of relevant available data shrinks in tandem. If we seek a model that can tackle frontier research in algebraic geometry, we can no longer depend on thousands of worked examples; only a tiny handful of cutting-edge results exist in the literature. If we want a model that can design novel materials, the relevant experimental data may consist of only dozens of measurements from expensive laboratory procedures. More broadly, projections suggest that we may exhaust the supply of publicly available, high-quality text data within years [Villalobos et al., 2024], creating a looming bottleneck for the current paradigm of ever-larger pretraining datasets.

For these reasons, the next generation of foundation models must be able to learn from less. This requires rethinking multiple stages of the data pipeline: how we *select* which data to train on, how we *generate* or *annotate* new data when labels are scarce, and how we *adapt* pretrained models to new domains without expensive retraining. Part II of this dissertation addresses these challenges through skills-based data selection algorithms that

exploit the structure of what models need to learn, geometric adaptation techniques that leverage relational knowledge to enable zero-shot prediction, and benchmarks that evaluate how well automated methods can generate training labels from minimal supervision.

1.1.3 Bridging the Gap to Scientific Domains

Machine learning research has historically concentrated on a small number of well-studied domains—primarily natural image classification, object detection, and natural language understanding. The architectures, search spaces, hyperparameters, and evaluation protocols that the community has developed are heavily tuned to these settings. When practitioners attempt to apply the same methods to scientific domains—partial differential equations, protein structure prediction, electrocardiogram analysis, malware detection, audio classification—the results are often disappointing.

The reasons for this gap are both methodological and structural. Methodologically, standard neural architecture search (NAS) techniques rely on hand-curated operation sets (various convolutions, pooling layers, skip connections) designed for image recognition. These operations may be entirely inappropriate for scientific data that lives on different manifolds, has different symmetries, or requires different inductive biases. Structurally, the community lacks standardized benchmarks for evaluating ML methods across diverse scientific tasks. Existing benchmarks typically focus on a single domain or a single task type, making it difficult to assess whether a method generalizes or merely overfits to the idiosyncrasies of one setting. Even basic questions—such as how to fairly compare methods across tasks with different metrics—lack principled answers.

Part III of this dissertation addresses these gaps. We develop the AUP score, a principled metric for aggregating performance across tasks with heterogeneous evaluation criteria. We construct benchmarks (NAS-Bench-360, AutoWS-Bench-101, the AutoML Decathlon) that evaluate methods across truly diverse tasks, revealing that state-of-the-art NAS and AutoML techniques often fail catastrophically outside their home domain. Finally, we introduce XD-operations, a search space of neural operations that generalizes standard convolutions and can discover novel domain-tailored operations for scientific tasks including PDE solving, protein folding, and music modeling.

1.2 Contributions and Outline

This dissertation is organized into three parts, each corresponding to one of the research directions outlined above. We summarize the contributions of each chapter below.

1.2.1 Part I: Scaling Recipes for Foundation Models

The first part of this dissertation develops scaling laws and design methods that go beyond standard prescriptions by accounting for test-time compute, skill-dependent behavior, and architectural composition.

Chapter 2: Test-Time Scaling Makes Overtraining Compute-Optimal. Modern LLMs scale at test time via repeated sampling, where inference cost grows with both model size and the number of samples. This creates a fundamental tradeoff that pretraining scaling laws like Chinchilla do not address. We present Train-to-Test (T^2) scaling laws that jointly optimize model size N, training tokens D, and the number of inference samples k under fixed end-to-end compute budgets. We develop two complementary approaches—one modeling the task loss, the other modeling pass@k accuracy directly—and show that both agree: when accounting for inference cost, *optimal pretraining decisions shift radically into the overtraining regime*, recommending models that are substantially smaller and more overtrained than Chinchilla prescribes. Across eight downstream tasks, we validate these predictions by training models in the forecasted optimal region, confirming their superior performance. We further show that these findings persist after post-training via fine-tuning and supervised fine-tuning.

Chapter 3: Compute-Optimal Scaling of Skills. While scaling laws provide useful rules of thumb, their predictions are typically derived from a single aggregate metric—validation loss on a held-out corpus—which can bias results depending on the content of that corpus. In this chapter, we ask whether compute-optimal allocations differ across specific model capabilities. Through experiments spanning nine compute scales and two skill categories measured across nineteen benchmarks, we demonstrate that *knowledge-based tasks are capacity-hungry*—favoring larger models with fewer training tokens—while *code-based reasoning tasks are data-hungry*—favoring smaller models with more data. We show that these differences are fundamental, not merely artifacts of data composition: changing the

proportion of skill-relevant data in the pretraining mix shifts the compute-optimal point for that skill, but the differences between skills persist even at comparable data proportions. These findings have practical implications for how practitioners should select validation sets and allocate compute when training models intended for specific downstream capabilities.

Chapter 4: Pretrained Hybrids with MAD Skills. Recent breakthroughs in sub-quadratic architectures like Mamba have shown that hybrid models mixing different architectural families can achieve better efficiency-performance tradeoffs than pure Transformers. However, designing such hybrids requires manually exploring a combinatorially large search space, and new hybrids must typically be trained from scratch. We present Manticore, a system that automates hybrid design while reusing pretrained model weights. Manticore first learns lightweight projectors that translate features from different pretrained model blocks (e.g., Transformer and Mamba) into a shared representation space, then applies differentiable neural architecture search to discover task-specific mixtures of these blocks. The resulting pretrained hybrids are competitive with carefully hand-designed architectures on both synthetic and real-world tasks, while requiring only a fraction of the compute that would be needed for training from scratch.

1.2.2 Part II: Data Efficient Learning

The second part develops techniques for learning effectively from limited data, through data selection, geometric adaptation, and automated supervision.

Chapter 5: Skill-it! A Data-Driven Skills Framework. A key ingredient in enabling foundation models to perform diverse tasks is the data on which they are trained. Despite its importance, data selection for state-of-the-art models mostly relies on heuristics for filtering and mixing datasets. We introduce Skill-it, a framework that formalizes the notion of *skills*—units of model capability such as the ability to recall factual knowledge or perform multi-step reasoning—and an online data selection algorithm that exploits prerequisite dependencies between them. Skill-it dynamically reweights training data based on the model's current proficiency at each skill, prioritizing prerequisites before advancing to more complex capabilities. In the continual pretraining setting, Skill-it achieves up to $3\times$ reduction in data requirements while improving performance. When applied to the

RedPajama dataset for a 3B parameter model, Skill-it achieves higher accuracy with 1B additional training tokens than uniform sampling achieves with 3B tokens.

Chapter 6: Geometry-Aware Adaptation for Pretrained Models. While there is a vast amount of structured knowledge available about general concepts—from knowledge graphs to ontologies to geometric properties of embeddings—the default approach to adapting pretrained models to new concepts is expensive fine-tuning. We present Loki, a training-free method that adapts pretrained classifiers to predict classes they have never observed by exploiting metric structure in the label space. Loki replaces standard argmax prediction with the Fréchet mean, enabling models to navigate the space of observed and unobserved classes via a simple linear transformation of model outputs. We provide theoretical analysis characterizing when Loki can predict *any* unobserved class, develop sample complexity bounds, and introduce a theoretically-grounded active learning procedure for optimally introducing new concepts. Empirically, Loki improves CLIP by up to 19.5% on zero-shot CIFAR-100 tasks and scales to label spaces with hundreds of thousands of classes.

Chapter 7: AutoWS-Bench-101. Weak supervision is a powerful method for building labeled datasets by aggregating multiple noisy label estimates, but its application scope is limited by the difficulty of constructing labeling functions for domains with complex features. We introduce AutoWS-Bench-101, a benchmark for evaluating *automated* weak supervision techniques across ten diverse datasets spanning images, text, time series, and tabular data. The central question of the benchmark is whether, given 100 initial labels, a practitioner should use an AutoWS method to generate additional labels or rely on a simpler baseline such as zero-shot predictions from a foundation model. We find that in many settings, AutoWS methods must incorporate signal from foundation models if they are to outperform simple few-shot baselines, pointing to the need for tighter integration between automated supervision and pretrained models.

1.2.3 Part III: Adaptation to Scientific Domains

The third part builds evaluation frameworks, scoring methods, and neural primitives for applying machine learning to the long tail of scientific tasks beyond standard vision and language benchmarks.

Chapter 8: The AUP Score. A recurring challenge in evaluating methods across diverse scientific tasks is the lack of a principled way to aggregate performance when each task has its own domain-specific metric. Naive approaches such as averaging scores or rankings can weight metrics unfairly or disproportionately penalize tied methods. We present the Area Under the Performance Profile (AUP) score, which computes the area under a performance profile curve to produce a single aggregate score that accounts for relative performance differences across tasks. Originally introduced in the AutoML Decathlon competition, the AUP score has since been adopted in multiple subsequent competitions and benchmarks, including the AutoML Cup and MLCommons AlgoPerf. We demonstrate the AUP score in the context of MLGym, a benchmark for evaluating LLM agents on AI research tasks.

Chapter 9: AutoML Decathlon. To evaluate whether modern AutoML methods can handle truly diverse tasks, we organized the AutoML Decathlon as a NeurIPS 2022 competition. Participants were presented with ten development tasks—spanning PDEs, spherically projected images, financial time series, audio, biological signals, and tabular data—and evaluated on ten separate hidden test tasks. The competition attracted participants from across the AutoML community and revealed that the most successful approaches combined modern ML techniques (large-scale transfer learning, differentiable NAS, advanced hyperparameter optimization) rather than relying on any single paradigm. We provide a detailed analysis of the top solutions, comparisons to additional baselines, and insights into which task characteristics most influenced method selection.

Chapter 10: NAS-Bench-360. Neural architecture search has achieved impressive results on standard benchmarks like CIFAR and ImageNet, but its effectiveness on tasks outside these well-studied domains is poorly understood. We present NAS-Bench-360, a benchmark of ten tasks drawn from diverse application domains including protein folding, PDE solving, audio classification, and satellite imagery. Each task is carefully chosen to interoperate with modern CNN search methods while being far-afield from their original development domain. We show that several state-of-the-art NAS methods perform *inconsistently* across these tasks, with many producing catastrophically poor results on tasks outside their comfort zone. For two tasks, we release precomputed performance of 15,625 architectures to enable tabular NAS research in new domains.

Chapter 11: Rethinking Neural Operations for Diverse Tasks. Standard NAS is limited to searching over small, hand-curated sets of operations—different convolution kernel sizes, pooling, skip connections—designed primarily for image classification. These operations may be entirely inappropriate for scientific domains with different symmetries and inductive biases. We present XD-operations, an expressive search space that generalizes convolutions by replacing their Fourier diagonalization with learnable efficient linear transforms. XD-operations can discover novel domain-tailored neural operations without manual design. Starting from vanilla CNN backbones, we show that XD-operations outperform the state-of-the-art Fourier Neural Operator on PDE solving, exceed human-designed architectures on protein folding tasks, and match or surpass specialized models for music modeling—establishing that automated operation discovery can compete with expert-designed domain-specific architectures across diverse scientific applications.

1.2.4 Outline

The remainder of this dissertation is organized as follows. Chapters 2–4 comprise Part I on scaling recipes for foundation models. Chapters 5–7 comprise Part II on data efficient learning. Chapters 8–11 comprise Part III on adaptation to scientific domains. Chapter 12 concludes the dissertation. Appendices for each chapter follow, containing supplementary experiments, proofs, and additional details.

Part I

Scaling Recipes for Foundation Models

Chapter 2

Test-Time Scaling Makes Overtraining Compute-Optimal

Pretraining scaling laws tell us how to optimally train language models, but not how to deploy them [Kaplan et al., 2020a, Hoffmann et al., 2022a]. Test-time scaling laws tell us how to optimally allocate compute at deployment, but not how to train models [Snell et al., 2024, Brown et al., 2025]. The two have developed largely in *isolation*, yet are fundamentally coupled. Model size and training duration determine both the quality and cost of inference samples. Models designed to reason through frontier research problems will be sampled from hundreds or thousands of times [Jaech et al., 2024, Guo et al., 2025]; these should be trained differently from chat models that instantly answer everyday questions.

Should parameter and token counts change if you know how your model will be used at test time? In practice, Chinchilla [Hoffmann et al., 2022a] scaling laws guide the allocation of pretraining compute for flagship models. However, modern model releases are families spanning a range of sizes [Touvron et al., 2023a, Groeneveld et al., 2024, Qwen et al., 2024], with the lower end intentionally *overtrained* well beyond Chinchilla-optimal ratios to reduce per-query inference cost. This makes them natural candidates for test-time scaling, yet nothing connects pretraining decisions to this inference strategy. No existing scaling law captures the core tradeoff: smaller models are cheaper per sample but weaker per sample, and the benefit of repeated sampling is a highly nonlinear function of per-sample quality.

Unifying pretraining and inference scaling is challenging because the two regimes operate under fundamentally different evaluation criteria. Pretraining is evaluated using

the loss, a smooth, continuous quantity. Test-time scaling, by contrast, is evaluated through downstream task metrics such as pass@k—the probability of producing at least one correct answer in k independent attempts. Should a unified scaling law across pretraining and test-time scaling model the loss or model the pass@k accuracy?

Prior work has addressed pieces of this problem but not the whole. Sardana et al. [2023] extends Chinchilla to account for inference cost, but considers only the aggregate volume of *single-pass* serving instead of the multiplicative cost and performance gains from repeated sampling. Recent studies empirically show that allocating more inference compute to smaller models via repeated sampling can match or exceed the performance of larger ones [Brown et al., 2025, Snell et al., 2024], but they treat pretrained models as given and do not address how they should have been trained. Schaeffer et al. [2026] develop scaling laws that predict pass@k from pretraining compute, but treat this as forecasting rather than an optimization problem—they predict what performance *will be* for a given model, not what model *should be* trained for a given budget. No existing work jointly optimizes model size, training duration, and the number of inference samples under a single compute budget.

In this work, we close the loop between pre-training and test-time scaling. We propose Train-to-Test (T^2) scaling laws that predict performance as a function of model size N, training tokens D, and number of samples k, and optimize over all three under a total compute budget that includes both training ($6ND$) and inference ($2Nk$) cost. Following Chinchilla, we evaluate multiple modeling approaches: whether to model the loss or pass@k as functions of N, D, and k. Although the two approaches are quite different, we find that they agree closely: both suggest substantial overtraining and test-time scaling across our evaluations. We build on an existing set of Chinchilla scaling checkpoints from Porian et al. [2024], extending it into the overtrained regime and assembling a testbed of over 100 models across 12 compute levels spanning three orders of magnitude.

Using T^2 scaling laws, we find that *optimal pretraining decisions shift radically into the overtraining regime* when considering test-time compute. When we correct for the cost of repeated sampling, the optimal model is substantially smaller and more overtrained than what Chinchilla prescribes. Our evaluation spans eight tasks covering knowledge, reasoning, and language understanding, on which we investigate three research questions:

RQ1 Should pretraining change if you know your test-time scaling budget? Yes—T^2 scaling consistently recommends small overtrained models. (§2.3.1)

RQ2 Does T^2 extrapolate to overtrained checkpoints? Yes—we overtrain models from

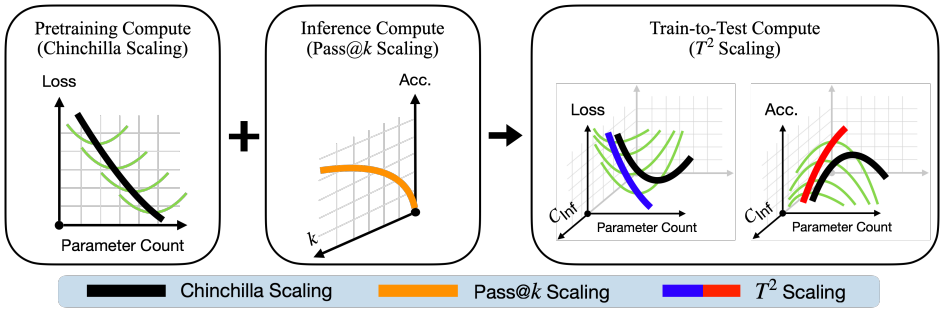


Figure 2.1: Our T^2 scaling laws combine Chinchilla scaling for pretraining with pass@k modeling for test-time scaling via repeated sampling to obtain optimal pretraining allocations subject to a test-time scaling budget. T^2 recommends overtraining compared to Chinchilla.

scratch and show that they consistently outperform Chinchilla checkpoints. (§2.3.2)

RQ3 Does T^2 scaling survive post-training? Yes—we find that compute-optimal trade-offs derived from base models persist after supervised fine-tuning. (§2.3.3)

- End-to-end scaling: We formalize train-to-test scaling as a joint optimization over model size N, dataset size D, and inference compute k under train and test budgets.
- Loss and accuracy scaling: We introduce two complementary approaches: (i) loss- and (ii) accuracy-based formulations that explicitly incorporate inference cost.
- Validation on overtrained checkpoints: We train models in the predicted overtrained regime and show improved performance under a range of fixed inference budgets.
- Interactions with post-training: The predictions from our scaling approach persist after post-training, even though overtrained models are harder to fine-tune.

2.1 Background

Our work connects two important areas: (i) pretraining scaling laws and (ii) test-time sampling strategies after deployment. We begin with their setups, and then dive into our new modeling techniques. A summary of additional related work can be found in Appendix 2.A.

Chinchilla scaling laws for pretraining. The Chinchilla scaling law [Hoffmann et al., 2022a] models the pretraining loss as a function of finite model capacity N and dataset size D (number of training tokens): $L(N, D) = E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}}$, where E represents an irreducible loss floor fit for the given data distribution and evaluation setup while the remaining terms capture reducible contributions from N and D. The parameters A, B, α, β, and E are all non-

negative and are fit empirically from a grid of training runs. Here, the loss is assumed to be the negative log-likelihood (NLL) over the data distribution: $\mathbb{E}_{(x,y)\sim\mathcal{D}}[-\log(p(y|x))]$ with $p(y|x)$ being the probability assigned by the model. Given a pretraining budget $C_{\text{train}} \approx 6ND$, the *compute-optima* minimize L subject to this constraint, yielding $N^*(C_{\text{train}}) \propto C_{\text{train}}^a$ and $D^*(C_{\text{train}}) \propto C_{\text{train}}^b$ with $a \approx b \approx 0.5$. That is, the optimal model size and training tokens should scale at similar rates as a function of the pretraining compute budget.

Pass@k estimation for test-time scaling. The standard metric for evaluating repeated sampling is pass@k: draw k independent samples from a model and succeed if *any* sample is correct. For a single problem i with per-sample success probability p_i, the probability of at least one answer in k attempts being correct is $\text{pass@}k_i = 1-(1-p_i)^k$. Aggregating over a benchmark $\mathcal{D}$ of M problems gives the expected pass@k:

$$\text{pass@}k_{\mathcal{D}} = \mathbb{E}_{i\sim\mathcal{D}}\left[\text{pass@}k_i\right] = \frac{1}{M}\sum_{i=1}^{M}\left[1-(1-p_i)^k\right].$$

2.2 Estimating Optimal Pretraining Allocations for Test-Time Scaling

We present two modeling approaches for T^2 scaling that answer our central research question: *should choices made during pretraining change if you know your test-time scaling budget?* In our first approach, we model the impact of repeated sampling on the loss by fitting a parametric function of the negative log pass@k. In our second approach, we model the pass@k accuracy directly by composing Chinchilla scaling with a pass@k estimator. In §2.3, we show that our findings are robust across both approaches. Finally, once we establish these two approaches, we answer our main research question by standardizing the test-time scaling budget: using more repeated samples for smaller models and fewer for larger models. Standardizing the inference budget of test-time scaling across checkpoints allows us to see how optimal pretraining decisions shift in light of test-time scaling considerations. If the optimal pretraining decisions (model size and the number of training tokens) shift compared to those recommended by standard Chinchilla scaling, then the answer to RQ1 is yes: pretraining decisions should change if you know your test-time scaling budget.

We first describe the optimization objectives of our T^2 approaches. Given a compute budget for training (C_{train}) and inference (C_{inf}), the optimization problem in terms of the

NLL is:

$$\min_{N,D,k} \; L(N,D,k) \qquad \text{s.t.} \quad 6ND \leqslant C_{\text{train}} \;\text{ and }\; 2Nk \leqslant C_{\text{inf}}, \tag{2.1}$$

or similarly, in terms of the pass@k accuracy:

$$\max_{N,D,k} \; \text{Acc}(N,D,k) \qquad \text{s.t.} \quad 6ND \leqslant C_{\text{train}} \;\text{ and }\; 2Nk \leqslant C_{\text{inf}}. \tag{2.2}$$

$L(N,D,k)$ and $\text{Acc}(N,D,k)$ represent the aggregated NLL and accuracy respectively, as functions of model capacity N, dataset size D, and number of sampling attempts k.

2.2.1 Approach 1: T^2 as a Parametric Model of the Task Loss

Our first approach models the loss as a function of the parameter count N, training tokens D, and the number of repeated samples k used at test-time in order to optimize Equation 2.1. First, in order to make repeated sampling compatible with the negative log likelihood (NLL), we rewrite the single-sample probability in terms of the probability that the target outcome is obtained at least once under k repeated samples, following prior work on pass@k [Chen et al., 2021b, Brown et al., 2025, Ehrlich et al., 2025, Schaeffer et al., 2025]. That is, working with the definition of pass@k_i allows us to define the corresponding NLL-style objective under repeated sampling as

$$\mathbb{E}_{i\sim\mathcal{D}_{\text{task}}}[-\log \text{pass@k}_i] = \mathbb{E}_{i\sim\mathcal{D}_{\text{task}}}\left[-\log\left(1-(1-p_i)^k\right)\right],$$

where $\mathcal{D}_{\text{task}}$ is a distribution over samples i representing a downstream task.

With this in place, we can model the negative log pass@k as an extension of the Chinchilla scaling law, $\widehat{L}(N,D)$ by adding a power-law term in k:

$$\widehat{L}(N,D,k) = \widehat{L}(N,D) + \frac{G}{k^\gamma} = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta} + \frac{G}{k^\gamma}.$$

We choose this model because prior work has found that the negative log pass@k contribution from k yields power law scaling[1] under an assumption that the task diffi-

[1] By Jensen's inequality, our NLL-style objective acts as an upper-bounding surrogate on the negative log expected pass@k, which scales as a power law (we minimize the expected negative log pass@k). Therefore, minimizing our surrogate minimizes the quantity of interest.

culty distribution can be modeled by a Beta distribution, which has been found to hold in practice [Brown et al., 2025, Schaeffer et al., 2025]. This has convenient properties when combined with the other power law terms in N and D in the Chinchilla scaling law:

First, when $k = 1$, we recover standard Chinchilla scaling:

$$\widehat{L}(N, D, 1) = E' + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}} = \widehat{L}(N, D),$$

where $E' = E + G$ absorbs the additional constant. Second, a property of Chinchilla scaling is that as $N, D \to \infty$, the model approaches the 'irreducible loss' term E. Given its power law form, this is still true when k approaches infinity alongside N and D.

2.2.2 Approach 2: T^2 as a Parametric Model of the Task Accuracy

While the previous model is simple, it trades off interpretability—practitioners often value pass@k forecasts due to their interpretation as the likelihood of solving a problem given a certain compute investment. Our second approach addresses this by modeling the pass@k directly as an accuracy-like metric as a function of N, D, and k, which optimizes Equation 2.2.

A naive approach to modeling pass@k might be to begin with $\widehat{L}(N, D)$, and simply map the NLL to accuracy p for the same task, then compute pass@$k = 1 - (1 - p)^k$. Prior work has shown that the relationship between the mean NLL and the mean accuracy can be well approximated using a fitted sigmoid [Grattafiori et al., 2024]. In other words, we can model the mean single-pass task accuracy, $\mathbb{E}_{\mathcal{D}_{\text{task}}}[\text{Acc}(N, D)]$, as $\sigma_\theta(\widehat{L}(N, D))$ with a parameterized sigmoid σ_θ fit to pairs of NLL and accuracy values on the task distribution across the model population. So this naive model of the pass@k might take the following form:

$$\widehat{\text{Acc}}_{\text{naive}}(N, D, k) = 1 - (1 - \sigma_\theta(L(N, D)))^k.$$

However, our goal is instead to obtain an estimator of the *mean pass@k accuracy* that depends on the scaling parameters, namely $\mathbb{E}_{\mathcal{D}_{\text{task}}}[\text{Acc}(N, D, k)]$, rather than the single-pass accuracy. This naive model overestimates due to the concavity of the pass@k:

$$1 - \left(1 - \mathbb{E}_{\mathcal{D}_{\text{task}}}[\text{Acc}(N, D)]\right)^k \geqslant \mathbb{E}_{\mathcal{D}_{\text{task}}}\left[1 - (1 - \text{Acc}(N, D))^k\right] = \mathbb{E}_{\mathcal{D}_{\text{task}}}[\text{Acc}(N, D, k)].$$

A simple way to avoid overestimating the pass@k would be to directly use the per-

question probabilities from model likelihoods, which would allow us to compute the mean pass@k exactly. However, our goal is a scaling law, a parametric model that can forecast pass@k at unevaluated (N, D, k) configurations. This requires us to model the *distribution* of per-question probabilities and how this distribution varies with model size and training tokens.

Intuitively, we want to account for the natural spread of difficulty between tasks in our data distribution. We do this by modeling the per-question single-pass accuracies as a Beta distribution, following prior work [Kazdan et al., 2025]. We model $\mathrm{Acc}(N, D) \sim \mathrm{Beta}(a_{N,D}, b_{N,D})$, and parameters $a_{N,D}$ with $b_{N,D}$ related to N and D via the NLL, which we model as a Beta regression problem. Using the mean (μ) and sample size (ν) parameterization of the Beta distribution, we model $\mu \in (0,1)$ and $\nu \in (0,\infty)$ using standard link functions from Beta regression: a logit link for the mean (which we rescale with an additional parameter), and a log link for the sample size. We relate this to the loss by using the Chinchilla loss estimate as our linear predictor. This yields the following parameterization of $a_{N,D}$ and $b_{N,D}$:

$$
\begin{aligned}
\mu_{N,D} &= \sigma_\theta(\widehat{L}(N,D)) = \frac{\theta_2}{1+\exp\big(\theta_1 \cdot (\widehat{L}(N,D) - \theta_0)\big)}, \\
\nu_{N,D} &= \exp(\theta_3 + \theta_4 \cdot \widehat{L}(N,D)), \\
a_{N,D} &= \mu_{N,D}\nu_{N,D}, \\
b_{N,D} &= (1-\mu_{N,D})\nu_{N,D}.
\end{aligned}
$$

Finally, using this model of the single-pass accuracy, we obtain the following pass@k model via properties of the Beta distribution:[2]

$$
\begin{aligned}
\widehat{\mathrm{Acc}}(N,D,k) &= \mathbb{E}_{\mathrm{Acc}(N,D)\sim\mathrm{Beta}(a_{N,D},b_{N,D})}\left[1-(1-\mathrm{Acc}(N,D))^k\right] \\
&= 1-\mathbb{E}_{\mathrm{Acc}(N,D)\sim\mathrm{Beta}(a_{N,D},b_{N,D})}\left[(1-\mathrm{Acc}(N,D))^k\right] \\
&= 1-\frac{\mathrm{B}(a_{N,D},\, b_{N,D}+k)}{\mathrm{B}(a_{N,D},\, b_{N,D})} \\
&= 1-\frac{\mathrm{B}(\mu_{N,D}\nu_{N,D},\, (1-\mu_{N,D})\nu_{N,D}+k)}{\mathrm{B}(\mu_{N,D}\nu_{N,D},\, (1-\mu_{N,D})\nu_{N,D})}.
\end{aligned}
$$

[2] $\mathrm{B}(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$ is the Beta function, where Γ is the Gamma function.

2.2.3 Inference Cost Correction

We equalize our T^2 scaling laws over an inference budget, C_{inf}, measured as the inference FLOPs per-token served. Just as the pretraining cost, $C_{train} = 6ND$, scales multiplicatively as a function of N and the number of training tokens D, the inference budget C_{inf} scales multiplicatively in k and approximately $2N$ FLOPs for a forward pass:

$$C_{inf} = 2Nk.$$

Then for a fixed budget C_{inf}, this gives us

$$k = \frac{C_{inf}}{2N},$$

where smaller models are allocated more repeated samples compared to larger models, subject to the same inference budget. We plug this into both of our T^2 scaling approaches.[3] For Approach 1, this gives us the inference-corrected loss model:

$$\widehat{L}\left(N, D, \frac{C_{inf}}{2N}\right) = \widehat{L}(N, D) + \frac{G}{k^{\gamma}} = E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}} + \frac{G}{\left(\frac{C_{inf}}{2N}\right)^{\gamma}},$$

and for Approach 2, the inference-corrected pass@k accuracy model:

$$\widehat{\mathrm{Acc}}\left(N, D, \frac{C_{inf}}{2N}\right) = 1 - \frac{B(\mu_{N,D}\nu_{N,D}, (1-\mu_{N,D})\nu_{N,D} + \frac{C_{inf}}{2N})}{B(\mu_{N,D}\nu_{N,D}, (1-\mu_{N,D})\nu_{N,D})}.$$

Now for both models, we can choose an inference budget C_{inf}, and observe the pretraining decisions that optimize both the pretraining and inference budgets C_{train} and C_{inf}. We represent Approach 1 in blue and Approach 2 in red for consistency with our Figures.

2.3 Experiments

In this section, we provide experimental results addressing the three research questions about our T^2 scaling approaches.[4] First, in §2.3.1, we show that if you know your test-time

[3] Optimization details for fitting Approach 1 and Approach 2 can be found in Appendix 2.F.

[4] Checkpoints, data, and experiment code will be released upon acceptance.

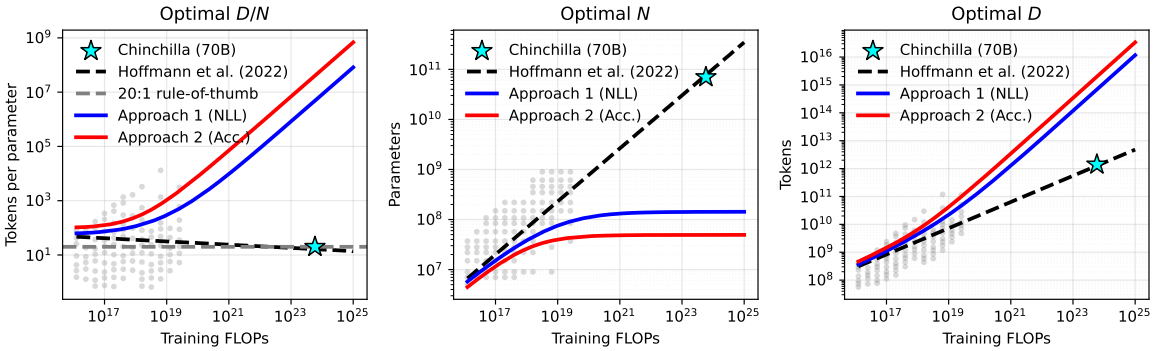


Figure 2.2: Optimal pretraining forecasts predicted by both T^2 approaches, compared to Hoffmann et al. [2022a]. (Left) Optimal tokens per parameter (including the 20 tokens per parameter rule of thumb used by practitioners), (Middle) Optimal model sizes. (Right) Optimal training set sizes. Both T^2 approaches forecast extreme overtraining.

scaling budget prior to pretraining, you should overtrain significantly beyond the standard Chinchilla recommendation of 20 tokens per parameter. In §2.3.2, we validate our predictions against overtrained checkpoints that extend standard Chinchilla scaling suites, showing that our scaling approaches extrapolate to the optimal regions that they predict. Finally, in §2.3.3, we show that overtraining predictions from our T^2 approaches persist after post-training. We fit T^2 scaling to checkpoints from Porian et al. [2024], which we extend with additional overtrained checkpoints, all trained on RefinedWeb [Penedo et al., 2023].

Tasks. We evaluate T^2 across eight real and synthetic tasks that we select to be simple enough for small base models, as all of our checkpoints have fewer than 1B parameters. The real tasks that we evaluate include the OpenAI variant of LAMBADA [Paperno et al., 2016, Radford et al., 2019], ARC-Easy [Clark et al., 2018], SciQ [Johannes Welbl, 2017], and OpenBookQA [Mihaylov et al., 2018]. We also evaluate on four synthetic tasks: simple knowledge recall, multi-step arithmetic reasoning, commonsense causal reasoning, and spatial reasoning, each consisting of 1,000 fill-in-the-blank or short completion questions that were generated using GPT-5 and Claude Opus 4.6. We provide additional task details in Appendix 2.E. Unless otherwise noted, we present macro averaged results over all tasks.

2.3.1 RQ1: Should Pretraining Change if You Know Your Test-Time Scaling Budget?

We evaluate RQ1 by comparing the predictions from T^2 to Chinchilla scaling and find that if you know your test-time scaling budget, you should significantly overtrain.

Setup. We fit both T^2 approaches to a suite of 106 checkpoints ranging in size from 5M to 901M parameters trained on roughly 50M to 120B tokens. Next, we set the per-token inference budget $C_{inf} = 140$B FLOPs, or approximately the cost of a single forward pass using the 70B Chinchilla model [Hoffmann et al., 2022a]. Finally, to compare T^2 forecasts to Chinchilla, we extrapolate the predictions from our T^2 approaches and standard Chinchilla scaling beyond our scaling suite to 10^{25} FLOPs. Using the same fits, we visualize pretraining isoFLOP profiles for both approaches. We compare the standard single-pass setting (k=1) to the inference-corrected setting with $C_{inf} = 2 \times 10^9$ FLOPs and $k = \frac{C_{inf}}{2N}$. Each of the 12 isoFLOP curves traces out a fixed pretraining budget C_{train} by varying N and D subject to $C_{train} = 6ND$. We plot the Chinchilla optimal frontier in black and that of T^2 in red. Results are macro averaged across all eight tasks. Individual scaling fits for each task across different budgets can be found in Appendix 2.B.

Results. Our results are shown in Figure 2.2 and Figure 2.3. Figure 2.2 shows that we can answer RQ1 in the affirmative: both T^2 approaches forecast models that are dramatically smaller and more overtrained than what Chinchilla prescribes. We additionally confirm that the Chinchilla scaling fit is consistent with Hoffmann et al. [2022a] by overlaying the 70B Chinchilla hero run model described in their paper, alongside the 20 tokens per parameter rule of thumb. Despite modeling fundamentally different quantities (NLL vs accuracy), both T^2 recommend extreme overtraining, with Approach 2 recommending more aggressive overtraining than Approach 1. Figure 2.3 shows isoFLOP curves under our T^2 approaches, how the overtraining trend develops within our scaling population. At every compute scale, the optimal frontier of both T^2 approaches shifts considerably toward smaller overtrained models with more repeated samples compared to the Chinchilla optimum. When inference-corrected, we see that the Chinchilla optimal frontier exhibits non-monotonic improvement in C_{train}. This is consistent with the findings of Snell et al. [2024], showing that smaller models with more test-time compute can outperform larger models. On the other hand, T^2 shows both stronger *and consistently monotonic* improvement, as we jointly model pretraining and test-time scaling. These results confirm that if you know your test-

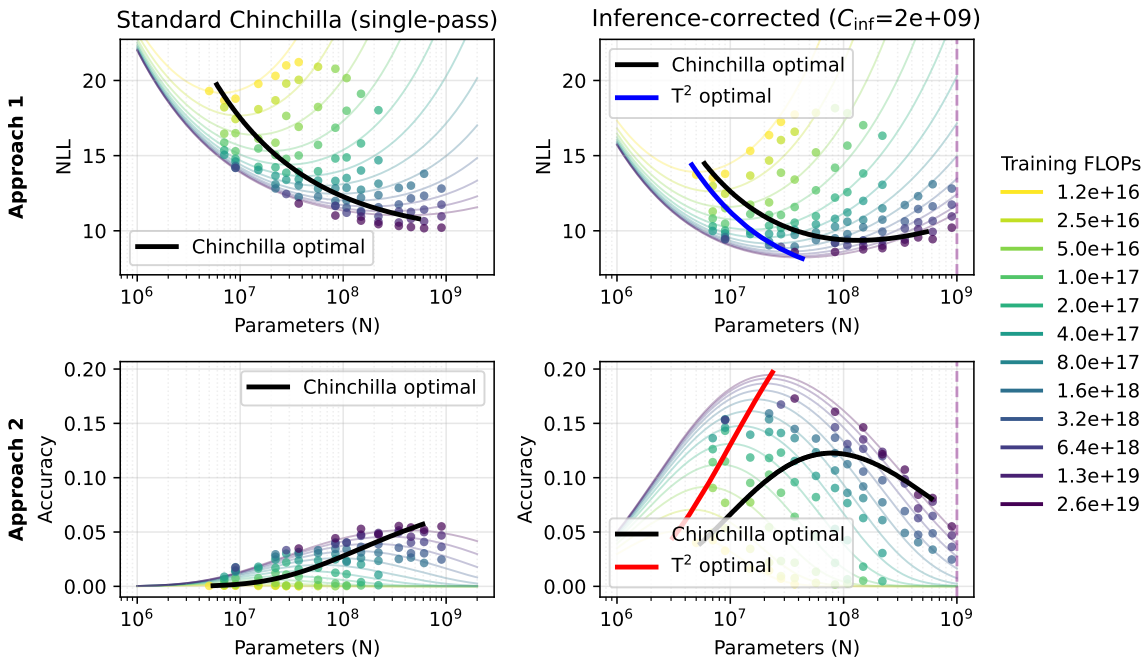


Figure 2.3: T^2 scaling across all of our evaluation tasks. Both approaches improve monotonically over Chinchilla scaling, while Chinchilla exhibits non-monotonic scaling in C_{train}.

time scaling budget, you should substantially overtrain compared to Chinchilla optimal pretraining.

2.3.2 RQ2: Does T^2 Scaling Extrapolate to Overtrained Checkpoints?

Next, we evaluate RQ2 by fitting both T^2 approaches to standard Chinchilla scaling checkpoints and measuring the performance of extrapolation to overtrained checkpoints.

Setup. We fit both of our T^2 approaches to a suite of 85 Chinchilla scaling checkpoints from Porian et al. [2024] (which stop short of the optimal overtraining regime that T^2 predicts) and measure the relative absolute error of extrapolating the predictions to 21 overtrained checkpoints that we train using an identical pretraining setup. We include training details and the exact checkpoint grid in Appendix 2.C. We also compare the empirical best overtrained checkpoint (among our 21) in the inference-corrected regime and compare it to the empirical Chinchilla optimal checkpoint at a pretraining budget of $C_{train} = 2.56 \times 10^{19}$

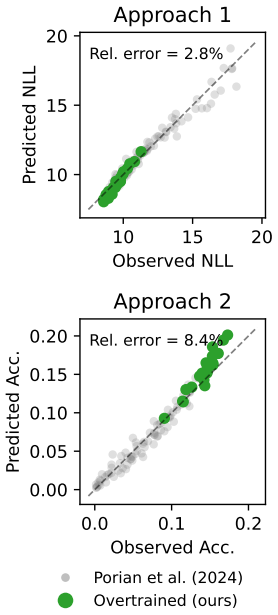


Figure 2.4: Extrapolating Porian et al. [2024] checkpoints to the overtraining regime.

		Best overtrained	Chinchilla opt.
Real	LAMBADA OpenAI	**49.90% (37M)**	27.30% (455M)
	OpenBookQA	**1.40% (37M)**	0.30% (901M)
	SciQ	**1.20% (37M)**	0.22% (611M)
	ARC-Easy	**0.14% (149M)**	0.07% (611M)
Synthetic	Simple Knowledge	**14.60% (84M)**	5.80% (901M)
	Simple Reasoning	**57.90% (37M)**	18.40% (901M)
	Commonsense Causal	**8.10% (37M)**	1.40% (901M)
	Spatial Reasoning	**6.00% (37M)**	1.10% (901M)

Table 2.1: Comparison of overtrained base models vs Chinchilla optimal pass@k, subject to $C_{train} = 2.56 \times 10^{19}$ and $C_{inf} = 2 \times 10^9$ FLOPs. Optimal model sizes are shown in parentheses.

		OpenBookQA	SciQ	ARC-Easy
FT	Best overtrained	**2.80% (37M)**	**56.10% (149M)**	**5.60% (149M)**
	Chinchilla opt.	0.45% (901M)	29.00% (901M)	1.50% (901M)
SFT	Best overtrained	**2.60% (37M)**	**66.80% (84M)**	**8.20% (37M)**
	Chinchilla opt.	0.38% (901M)	57.60% (347M)	3.40% (455M)

Table 2.2: Post-training comparison of overtraining vs Chinchilla optimal pass@k, subject to $C_{train} = 2.56 \times 10^{19}$ and $C_{inf} = 2 \times 10^9$ FLOPs. Optimal model sizes are shown in parentheses.

across all eight tasks. We set $C_{inf} = 2 \times 10^9$ for all of the above.

Results. Our extrapolation results are shown in Figure 2.4 and empirical checkpoint pass@k results are shown in Table 2.1. Figure 2.4 shows that our T^2 approaches both extrapolate to the 16 new overtrained checkpoints. While both approaches somewhat overestimate performance, Approach 1 extrapolates better than Approach 2, with a relative error of 2.8% compared to 8.4%. Table 2.1 shows that our best small overtrained checkpoints always outperform the Chinchilla optimal checkpoints when inference corrected, across all eight tasks. This confirms that T^2 extrapolates to real overtrained checkpoints, and that this phenomenon is not just an artifact of our T^2 approaches.

2.3.3 RQ3: Does T^2 Scaling Survive Post-Training?

Finally, we evaluate RQ3 by showing that our findings persist after post-training.

Setup. We explore two canonical post-training techniques: standard fine-tuning (FT) and supervised fine-tuning (SFT), where we only fine-tune on the targets. We post-train on the three real tasks that have a standard training set: ARC-Easy, SciQ, and OpenBookQA, and report improved performance on the test sets for each of these. Additional post-training details can be found in Appendix 2.D. We allocate the same number of training steps to each checkpoint, rather than scaling training based on FLOPs, since we ultimately train to convergence. After post-training, we fit both T^2 approaches to the FT and SFT checkpoints and evaluate their optimal tokens per parameter frontier compared to base models under T^2 scaling and the Chinchilla frontier. Finally, like in RQ2, we compare the best overtrained FT and SFT checkpoints to the Chinchilla optimal checkpoints for each task.

Results. Our results are shown in Figure 2.5 and Table 2.2. We see in Figure 2.5 that the optimal frontier continues to shift toward smaller overtrained models with more test-time samples across all three tasks and methods. Again, we find that these results are consistent between Approach 1 and Approach 2. On the other hand, we find that the optimal overtraining recommendation is somewhat subdued compared to T^2 on the base models alone, but not enough to shift it back to the original Chinchilla recommendation. The finding that it is subdued is consistent with prior work showing that overtrained models are harder to fine-tune [Springer et al., 2025]. Finally, we see in Table 2.2 that our best overtrained checkpoints still outperform the Chinchilla optimal checkpoints after post-training, and that performance improves across the board compared to the same analysis on base models in Table 2.1. This confirms that our findings with T^2 scaling persist after post-training.

2.4 Conclusion

In this work, we have presented T^2 scaling laws that jointly optimize model size, training tokens, and the number of repeated samples at test-time under fixed pretraining and inference budgets. We find that when test-time compute via repeated sampling is accounted for during pretraining decisions, the optimal model is substantially smaller and more overtrained than what standard Chinchilla scaling prescribes. This finding is consistent across two complementary modeling approaches: Approach 1, which models the NLL, and Approach 2, which models the pass@k accuracy directly. We validated this across eight real and synthetic downstream tasks, validated that T^2 scaling extrapolates to the overtraining regime where its optima are predicted, and that our findings persist after post-training. Based

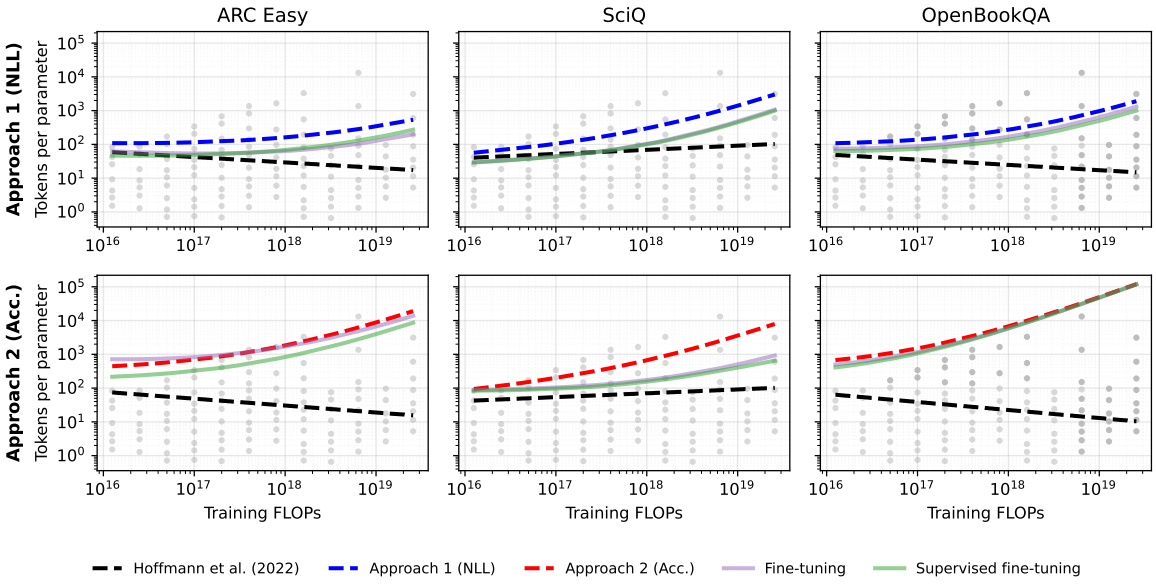


Figure 2.5: T^2 overtraining findings survive post-training. The optimal frontier is slightly subdued compared to base models, which is consistent with Springer et al. [2025].

on our findings, we offer a recommendation to practitioners: if you know your test-time scaling budget with repeated sampling, you should train a smaller model for longer, and T^2 scaling offers a blueprint for doing so. In future work, we plan to validate our prescribed overtraining recipes at larger scales, account for transformer-specific inference cost models, and explicitly model the role of post-training in T^2 scaling.

2.A Related Work

Our work sits at the intersection of three research threads: (i) pretraining scaling laws, (ii) test-time scaling, and (iii) overtrained models.

2.A.1 Pretraining Scaling Laws

Kaplan et al. [2020a] established that model loss follows predictable power laws as a function of model size and training data. Hoffmann et al. [2022a] (Chinchilla) refined this into *compute-optimal training recipes*, prescribing how model size and token count should scale together under a fixed compute budget. Recent extensions have broadened the scope of scaling law modeling: studying data quality and quantity [Goyal et al., 2024], incorporating

downstream task accuracy [Isik et al., 2024, Bhagia et al., 2024b], decomposing scaling behaviors across knowledge and reasoning skills [Roberts et al., 2025], and extending to multimodal settings [Shukor et al., 2025]. These frameworks, however, treat inference as an afterthought—optimizing for a model that is trained once and queried once. Sardana et al. [2023] take a step toward *deployment-aware* scaling by folding inference serving volume into the compute-optimal recipe, yet their analysis is limited to single-pass queries. We modernize this line of work, where the optimal training decisions must account for both the cost and the compounding performance gains of drawing multiple inference samples.

2.A.2 Test-Time Scaling

Beyond scaling pretraining compute, recent work has increasingly focused on investing computation at *inference time* [Snell et al., 2024, Zhang et al., 2025, Jaech et al., 2024, Orlanski et al., 2025]. This test-time paradigm often focuses on the search for a correct reasoning path rather than the model's inherent knowledge and can broadly be categorized into three regimes: (i) *parallel scaling*, which uses consensus through self-consistency [Brown et al., 2025], or verification over multiple independent responses [Saad-Falcon et al., 2025]; (ii) *sequential scaling*, which refines reasoning through iterative improvements or hierarchical pruning [Wei et al., 2022, Madaan et al., 2023]; and (iii) *internal scaling*, which allows the model to dynamically adjust generation depth based on task difficulty [Jaech et al., 2024]. In this work, we focus on *parallel repeated sampling*—the most common form of test-time scaling—and incorporate pretraining compute budget to jointly optimize allocation decisions.

2.A.3 Overtraining

Hoffmann et al. [2022a] (Chinchilla) prescribes a compute-optimal ratio of roughly 20 training tokens per model parameter, yet modern models releases routinely deviate from this blueprint by *training smaller models on far more tokens than recommended.* This deliberate overtraining is motivated by *inference efficiency*: a smaller model costs less per query at deployment. Recent model families illustrate this trend—Llama-2-7B [Touvron et al., 2023a] was trained on 2T tokens ($\sim$290$\times$ the recommended ratio); Google's Gemma-7B [Team et al., 2024] was trained on 6T tokens ($\sim$857$\times$), and its successor Gemma 2-9B [Team et al., 2024] on 8T tokens ($\sim$889$\times$)—with OLMo [Groeneveld et al., 2024] following a similar philosophy. Our

work complements these findings by examining overtraining through a different lens: rather than studying its effect on post-training [Springer et al., 2025], we show that overtraining is actively *beneficial* when models are deployed with a repeated-sampling inference budget, and we provide a principled framework for determining how much to overtrain given a joint train-and-test compute allocation.

2.B Per-Task Analysis

We present isoFLOP profiles for each of the individual tasks in our evaluation suite in Figure 2.6 for Approach 1 and Figure 2.7 for Approach 2 . We find that overtraining predictions are relatively stable across inference budgets for both approaches.

2.C Pretraining Details

In this section, we provide details of our pretraining setup and scaling grid.

2.C.1 Checkpoint Scaling Grid

Figure 2.8 shows our checkpoint grid, comprising pretrained checkpoints from Porian et al. [2024] alongside additional overtrained checkpoints we pretrained in this work. Model sizes range from 5M to 901M parameters, and training FLOPs span 1.25×10^{16} to 2.56×10^{19}. Each cell reports the number of tokens per parameter, which characterizes the degree of overtraining. Typically, a suite of Chinchilla scaling checkpoints contains checkpoints at either side of the typical 20 tokens per parameter recommendation derived from Hoffmann et al. [2022a]. However, since T^2 suggests overtraining beyond the available set of checkpoints, we train additional checkpoints at higher tokens per parameter ratios. The overtrained checkpoints (shown in orange) are used to validate our forecasts in §2.3.2.

2.C.2 Hyperparameters

We train our overtrained checkpoints, shown in Figure 2.8, from scratch using the OpenLM framework with the same fixed hyperparameters used for the Chinchilla-optimal checkpoints from Porian et al. [2024]. Specifically, we use their `hparams=base`, `warmup=short`,

`decay=chinchilla` configuration. We use the AdamW optimizer with a learning rate of 3×10^{-3}, $\beta_1 = 0.9$, $\beta_2 = 0.95$, and a decoupled weight decay of 1×10^{-4}. Training uses a global batch size of 256 sequences of length 2048 tokens, cosine learning rate decay to zero matched to the token budget of each run, and a warmup period equal in tokens to the model's parameter count. We apply gradient clipping with a max norm of 1.0, QK-normalization, z-loss with coefficient 10^{-4}, and train in bfloat16 mixed precision. All hyperparameters are held fixed across model sizes, consistent with the base (untuned) configuration of Porian et al. [2024]. We train on the RefinedWeb dataset with a vocabulary size of 50,432.

2.D Post-Training Details

We describe our post-training setup and configurations below. We employ two variants of post-training: **(i) standard fine-tuning** and **(ii) supervised fine-tuning (SFT)**. Standard fine-tuning follows the conventional next-token prediction objective, computing loss over both the instruction (question) and completion (answer). SFT, in contrast, computes loss over the completion only, excluding instruction tokens from parameter updates.

We fine-tune on three tasks—ARC Easy [Clark et al., 2018], SciQ [Johannes Welbl, 2017], and OpenBookQA [Mihaylov et al., 2018]—covering the full population of pretrained checkpoints, including the overtrained ones. Each model is trained for 6 epochs until convergence using a batch size of 8 and a constant learning rate of 2×10^{-5}, after that we evaluate on the respective test set. All fine-tuning experiments are conducted on 4 NVIDIA A10 GPUs. Box 2.D presents the training data format for each task, where the highlighted tokens indicate the completion portion used in the SFT loss computation. Their evaluation follows the same format: we measure negative log-likelihood over the correct answer placed in the highlighted placeholder.

2.E Evaluation Tasks

Next, we describe the eight downstream tasks used to evaluate T^2 scaling, covering both real-world benchmarks and synthetic tasks. For all tasks, we measure the NLL of each model over the correct answer.

We evaluate on four real-world benchmarks.

Box 1: Training Data Formats

Each format separates the prompt (plain) from the completion (highlighted), which is the only portion used in the SFT loss.

ARC Easy:

```
Question: {question}\nAnswer: {answer}
```

OpenBookQA:

```
{question} {answer}
```

SciQ:

```
{support}\nQuestion: {question}\nAnswer: {answer}
```

1. LAMBADA [Paperno et al., 2016] (OpenAI variant): tests long-range language understanding, where the model must predict the final word of a passage given a broad context.
2. ARC Easy [Clark et al., 2018]: consists of elementary-level science questions in a four-way multiple choice format, drawn from standardized tests.
3. SciQ [Johannes Welbl, 2017]: contains science exam questions paired with supporting passages, presented in a multiple-choice format.
4. OpenBookQA [Mihaylov et al., 2018]: requires multi-step reasoning by combining an open book of core science facts with broader common knowledge, presented as four-way multiple choice questions.

In addition to these four benchmarks, we incorporate four synthetic tasks spanning different domains. These tasks are designed to evaluate models on (i) simple knowledge recall, (ii) multi-step arithmetic reasoning, (iii) commonsense causal reasoning, and (iv) spatial reasoning. Each task consists of 1,000 fill-in-the-blank or short-completion questions, generated using GPT-5 and Claude Opus 4.6. Below, we present representative examples from each task along with their evaluation format. As in Box 2.D, the token spans used to compute the NLL are highlighted in each example below.

Box 2: Commonsense Causal Reasoning

Example 1:

```
Grandparents tell stories to grandchildren. Teachers
explain concepts to students. Coaches demonstrate
techniques to players
```

Example 2:

```
A mother comforts a crying baby. A teacher encourages
a struggling student. A coach motivates a
discouraged player
```

Box 3: Simple Knowledge Recall

Example 1:

```
The capital of Egypt is Cairo
```

Example 2:

```
The fifth taste is umami
```

2.F Fitting T^2 Scaling

In this section, we describe how each of our T^2 approaches are fit to empirical checkpoints.

Fitting Approach 1. We fit the seven parameters $(\log A, \log B, \log E, \alpha, \beta, \log G, \gamma)$ of the additive model by minimizing the sum of squared errors (SSE) between predicted and empirical NLL values across all checkpoints and sampled values of k. We use the L-BFGS-B algorithm with 500 random restarts (each with up to 5,000 iterations and a tolerance of 10^{-15}) and we select the run with the lowest objective value.

Fitting Approach 2. We fit the model in two stages. First, we fit the standard Chinchilla scaling model $\widehat{L}(N, D) = E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}}$ to the empirical NLL values of all checkpoints. We profile over a grid of 40 candidate E values spaced between $0.01 \cdot \min(\text{NLL})$ and $0.95 \cdot \min(\text{NLL})$; for each, we optimize the remaining four parameters $(\log A, \log B, \alpha, \beta)$ via L-

Box 4: Multi-Step Arithmetic Reasoning

Example 1:

```
I have 5 toys. I give away 2 toys. Step 1: I started with 5
toys. Step 2: I gave away 2 toys. Step 3: 5 minus 2 equals 3
```

Example 2:

```
Pattern: 10, 20, 30, ... This adds 10 each time. After 30
comes 40
```

Box 5: Spatial Reasoning

Example 1:

```
The baby is in the crib. The crib is in the nursery. The
nursery is in the house. So the baby is in the house
```

Example 2:

```
The glasses are in the case. The case is in the handbag.
So the glasses are in the handbag
```

BFGS-B with 50+ random restarts, using inverse-variance weighting across isoFLOP groups. Second, we fit the Beta regression parameters. The per-question success probability is modeled as $p \sim \text{Beta}(a_{N,D}, b_{N,D})$ where $\mu = a_{N,D}/(a_{N,D} + b_{N,D})$ is a scaled logit link and the concentration $\nu = a_{N,D} + b_{N,D}$ is parameterized as a log link function. Together, the five parameters $(\theta_0, \theta_1, \theta_2, \theta_3, \theta_4)$ are fit by minimizing SSE between predicted and empirical pass@k accuracy values over a grid of initializations seeded from a sigmoid baseline, again using L-BFGS-B.

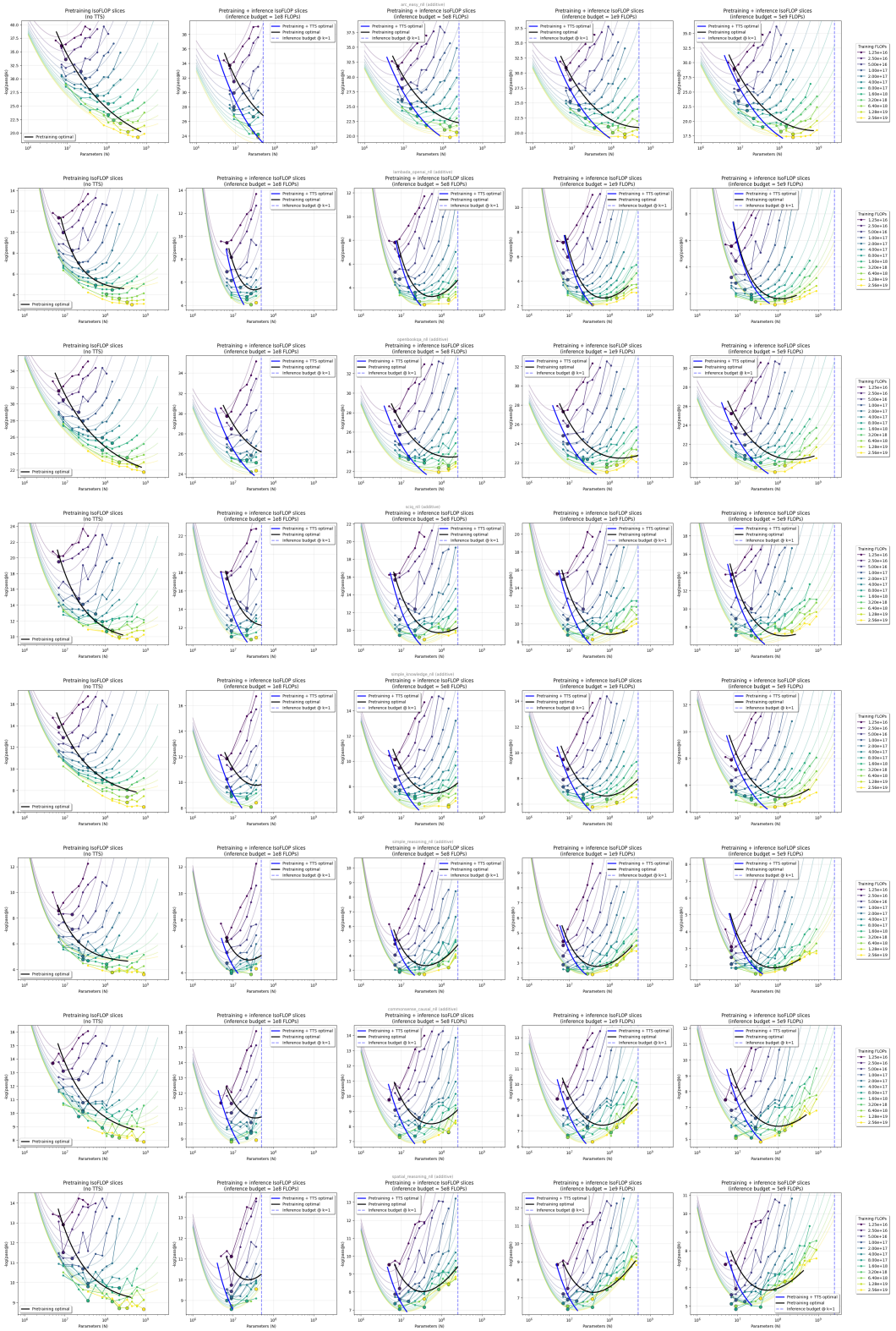


Figure 2.6: Approach 1 IsoFLOP profiles across different scaling budgets for all eight tasks.

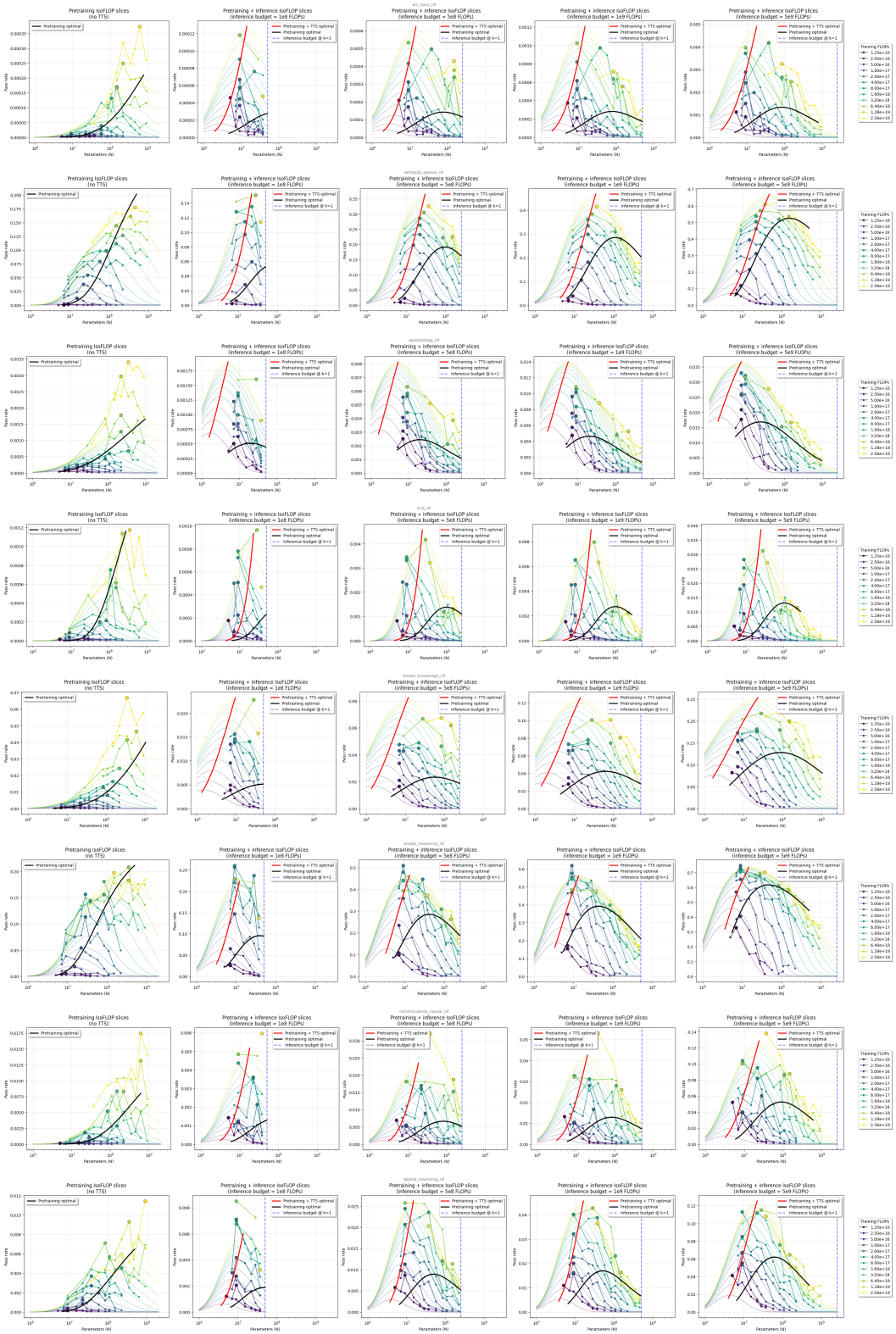


Figure 2.7: Approach 2 IsoFLOP profiles across different scaling budgets for all eight tasks.

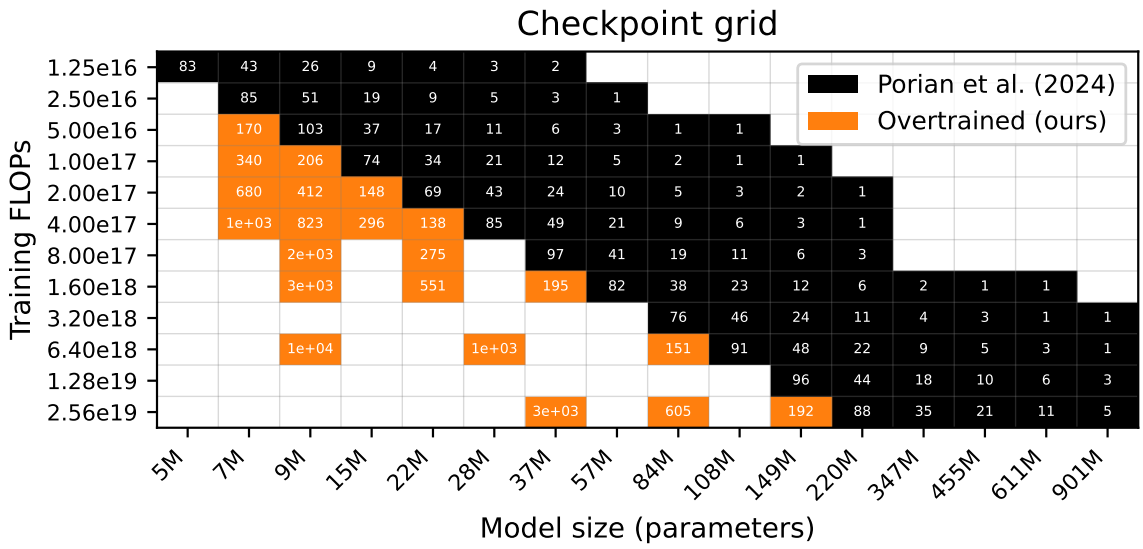


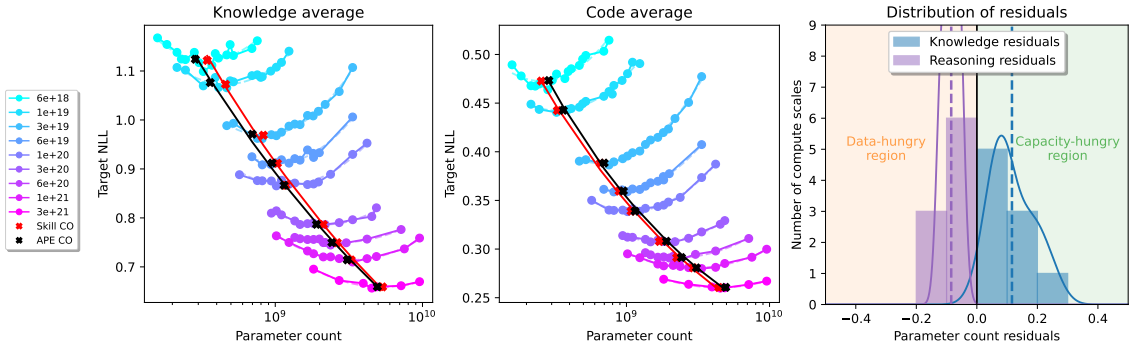
Figure 2.8: Overall checkpoint scaling grid. Each cell reports the number of tokens per parameter. Orange cells are overtrained checkpoints we created.

Chapter 3

Compute Optimal Scaling of Skills: Knowledge vs Reasoning

Used both to forecast performance for early pretraining decisions as well as decide on the optimal trade-off between parameters and pretraining dataset size given a particular compute budget, *scaling laws* [Kaplan et al., 2020a, i.a.] have played an important role in the development of large language models (LLMs). Famously, with a series of experiments with models with different data/parameter trade-offs, Hoffmann et al. [2022c] showed that previous LLMs were all erring on the side of too many parameters, causing a shift in the amount of training tokens to train LLMs. More recently, Grattafiori et al. [2024] used scaling laws not only to determine the optimal parameter count given their available compute budget, but also to forecast the impact of data selection decisions on evaluation scores.

In these works, the *compute optima* (COs), describing the optimal parameter count and number of training tokens, are selected based on *aggregate performance estimators* (APEs), in the form of negative log-likelihood (NLL) on a validation set not part of the pretraining corpus. Little is known, however, about whether the COs of individual skills such as mathematical reasoning, question answering (QA), or coding, align with these APE COs. While some studies use scaling laws to predict how downstream task performance improves with scale [e.g. Ye et al., 2025, Held et al., 2025], none of these studies cover whether COs themselves may be skill dependent. Is it possible that some skills are more *data-hungry*, whereas others benefit more from *extra parameters*? If so, how should that impact model training and training data selection?



(a) Knowledge QA. **(b)** Code. **(c)** Data- vs capacity-hunger.

Figure 3.1: Isoflop curves and COs for code and knowledge QA, along with the APE COs. In (a), we see that on average across held-out datasets, knowledge QA tends to be capacity-hungry compared to the APE CO. On the other hand, in (b), we see that code tends to be data-hungry relative to the APE CO. We show the distribution of these relationships in (c), where we plot distributions of the log-scale differences in parameter count between skill-dependent COs and APE COs and find their means lie on opposite sides of the APE CO. The black curves in (a) and (b) represent the predicted APE COs from Grattafiori et al. [2024], mapped to their respective IsoFLOP groups.

In this chapter, with an extensive set of experiments across 9 different compute scales and 2 skills as measured with 19 datasets across two different splits, we study exactly that. Specifically, we focus on the three research questions:

R1. Are COs skill dependent? First, we consider how the IsoFLOP curves and corresponding COs for *code*[1]- and *knowledge*-based skills compare to the APE COs, given a canonical datamix. Across the board, we find pronounced differences between the COs for these different skills: where knowledge QA tasks are capacity-hungry, code tasks instead prefer data.

R2. Is this an artefact of the pretraining datamix, or are code and knowledge skills fundamentally different? Using only the smallest compute scale, we next investigate whether these difference are a consequence of the proportion of skill-relevant data in the pretraining datamix of our experiments or that there is in fact a difference in how data- or capacity-hungry the different skills are. We find that both are true: changing the proportion of skill-relevant data shifts the CO for that skill, but COs for skills differ even with comparable

[1]We use code as a proxy for reasoning skills to avoid ambiguity and the challenge of concretely defining reasoning.

proportions of skill specific data.

R3. How does the existence of skill-dependent COs impact LLM training? Lastly, we focus on the more practical question of how these findings should impact model pretraining. First, again using only the smallest compute scale, we investigate if it is possible to *align* the COs for the two skills under consideration. We find that it is, but that whether this should be considered advantageous for overall model training depends highly on the choice of validation set. We then investigate how much the estimated optimal parameter count depends on the choice of validation set. We find that, on our smallest scale experiments (6×10^{18}), the optimal parameter count for the same skill varies by almost 50% across datamixes, and by more than 30% across validation sets. At larger scale, the difference is smaller, but it still exceeds 10% for the three largest compute scales. These experiments show that choosing a validation set that adequately represents what the final model should capture is critical to finding the right parameter counts.

Outline. With our work, we contribute to an empirical understanding of how different skills behave across model scales, and explore the concrete implications for CO training of LLMs. In the remainder of this chapter, we first provide the necessary background and definitions (section 3.1), and describe the data we use for our experiments (section 3.2). Next, in section 3.3, we describe our experiments and their results. Lastly, we discuss related works about scaling and data selection (section 3.4), and conclude in section 3.5.

3.1 Background

When computing scaling laws, prior work typically focused on modelling the loss on a validation set, which is usually a general text corpus drawn from the same distribution as the pretraining dataset. In this setup, the loss is effectively a weighted combination of all what the model should learn during training. *In contrast, we consider skill-dependent scaling laws.* In this section, we discuss the relevant background and notions of scaling laws, COs, and skills underpinning our analysis. In section 3.4, we describe related efforts in more detail, and in section 3.C, we provide formal definitions of the concepts that this section introduces.

3.1.1 Scaling laws

Neural scaling laws aim to model and predict loss values as a function of the compute scale in FLOPs. FLOPs are often estimated as $\mathrm{B} \approx 6\mathtt{pt}$, which involves contributions from the size of the training set in tokens, $\mathtt{t}$, and the parameter count of the model, $\mathtt{p}$. Typically, scaling laws that model these two quantities in relation to the loss exhibit a tradeoff between parameter count and compute scale. These are depicted using *IsoFLOP curves,* where the x-axis is the parameter count, and the y-axis is the loss on the training or validation set. To obtain those curves, Hoffmann et al. [2022c] pioneered using a 2D power law model which has separate power law components for parameter and token counts. In our case, we found 2D power law models to produce a poor fit to downstream evaluations, which we attribute to noise and small dataset sizes, compared to validation sets. Instead, we opted to follow the approach used by Grattafiori et al. [2024], which involves fitting separate degree-2 polynomials to each compute scale. We extend this by fitting a power law to the optima of the compute scales.

Compute budget and COs. Central to the idea of IsoFLOP curves, which model the loss at many compute scales, is the idea of *IsoFLOP groups* which model tradeoffs between dataset and model size *at fixed compute scales.* In other words, an IsoFLOP group is a set of models where the amount of training data, $\mathtt{t}$, and model size, $\mathtt{p}$, is varied subject to a fixed approximate compute budget $\mathrm{B} \approx 6\mathtt{pt}$. This tradeoff between dataset size and parameter count can be optimised at each compute scale, which means that at a given compute budget, there is a *optimal* parameter count and dataset size, $(\mathtt{p}^*, \mathtt{t}^*)$ s.t. B FLOPs. Typically, IsoFLOP curves (and therefore COs) are computed using the loss on the training set, or for flexibility, a validation set. Since both the training and validation sets include mixtures of data that might influence scaling behaviours differently, we refer to the losses on these sets as *aggregate performance estimators, or APEs.* Under this terminology, standard practice involves selecting COs based on APEs. Next, we define our notion of skills and how our analysis involves COs based on skills instead of APEs.

3.1.2 Skills

Following Chen et al. [2023a], we formalise the notion of a 'skill' by starting from the metric and dataset by which it can be quantified. More precisely, we say that a dataset $\mathcal{D}$ quantifies

Split	Knowledge skills					Code skills				
Hypothesis	Trivia QA (dev)		NQ (dev)	SQuAD (train)	MMLU (train)	RepoBench	BigCodeBench	CAT-LM (test gen.)	MultiPL-E (HumanEval)	MBXP
Held-out	Trivia QA (test)	Trivia QA (wiki test)	NQ (test)	SQuAD (dev)	MMLU (val)	SWE-bench (oracle)	BigCodeBench (test gen.)	CrossCodeEval	HumanEval	MBPP

Table 3.1: Hypothesis and held-out splits for code and knowledge-based skills. In all our experiments, we first develop hypothesis on a *hypothesis split* and then confirm them on a *held-out* split containing different datasets for the same skills. References to each of these datasets are provided in the text.

skill s if the performance of a model on that dataset according to metric $\mathcal{L}$ correlates with a model's ability to perform the intended skill s. Under this definition, multiple datasets can be associated with the same skill – something that we exploit in our experiments to validate our conclusions for one skill across multiple datasets – and in some cases one dataset can quantify multiple skills. It is difficult to exactly label what skills different datasets quantify, and the extent to which they correctly do so can be open for debate, so we largely follow the labelling of the creators of our evaluation datasets, but we do so under scrutiny. For simplicity, our work focuses on two specific skills which we suspect to have different scaling properties: *knowledge*, in the form of knowledge-based QA and *code*. In section 3.2, we discuss which datasets we use to quantify these skills.

Skill-dependent COs. Given the given definition, the CO for a skill s at a particular compute scale B is given by the parameter count p_s^* and training token budget t_s^* in the IsoFLOP group that optimises the loss on the dataset $\mathcal{D}_s$ quantifying the skill, rather than on APEs. As mentioned before, a primary aim of our work is to determine whether COs may be skill dependent. To quantify this, we consider how the COs for skills compare to the optima computed using APEs. If a skill fares better with comparatively more *data*, we call this skill *data-hungry*, while if a skill prefers more parameters than the APE optimum we call it *capacity-hungry*.

3.2 Data

In our experiments, we focus on two overarching skills: knowledge QA and code. In this section, we describe which datasets we use to measure models' abilities on those skills and describe how we infer their corresponding COs.

3.2.1 Skill data

For each of the two skills we evaluate, we identify a number of evaluation datasets designed to evaluate these respective skills. We split those datasets into a 'hypothesis' and 'held-out' split. In our analyses, we first form hypotheses on the hypothesis split of knowledge QA and code skills without accessing the held-out split, and then we validate our hypotheses on the held-out split. Our hypothesis and held-out splits include knowledge QA-based splits from Trivia QA [Joshi et al., 2017], NQ [Kwiatkowski et al., 2019], SQuAD [Rajpurkar et al., 2016], MMLU [Hendrycks et al., 2021a], and code-based splits from RepoBench [Liu et al., 2024], SWE-Bench [Jimenez et al., 2024], BigCodeBench [Zhuo et al., 2025], CAT-LM [Rao et al., 2024], CrossCodeEval [Ding et al., 2023], MultiPL-E [Cassano et al., 2023], HumanEval [Chen et al., 2021a], MBXP [Athiwaratkun et al., 2022], and MBPP [Austin et al., 2021]. Additional details of the the datasets and splits that we use in our experiments can be found in table 3.1.

Some of our datasets consist of compound datasets spanning multiple topics, such as MMLU. We categorise the subtasks of these datasets separately and then assign half of each to the hypothesis and held-out splits, respectively. We label a subtask as being knowledge-based if the model would have required prior knowledge of the subject area in order to answer the question accurately and remove all non-knowledge-based subtasks.

3.2.2 Pretraining data

Our canonical pretraining datamix comprises roughly 58.4% documents that are high in factual knowledge, 19.9% documents containing code, and the remaining 21.7% did not fall under either of these categories. To construct our training datasets for a given token budget t, we randomly sample documents from a larger pool of data from each of these categories, according to the proportions for each category. We vary these proportions in section 3.3.2 and section 3.3.3 by increasing the proportions of code or by increasing the proportion of knowledge. We include visualisations of the proportions for our canonical datamix, along with the proportions used in section 3.3.2 and section 3.3.3, in section 3.E.

3.3 Experiments

Now, we present the main results for our experiments, starting from our finding that COs are skill specific for our canonical datamix (section 3.3.1), followed by a deeper exploration

into the impact of the datamix on these conclusions (section 3.3.2), and finishing with a small investigation of how these results may impact model pretraining decisions (section 3.3.3).

3.3.1 Can COs differ between skills?

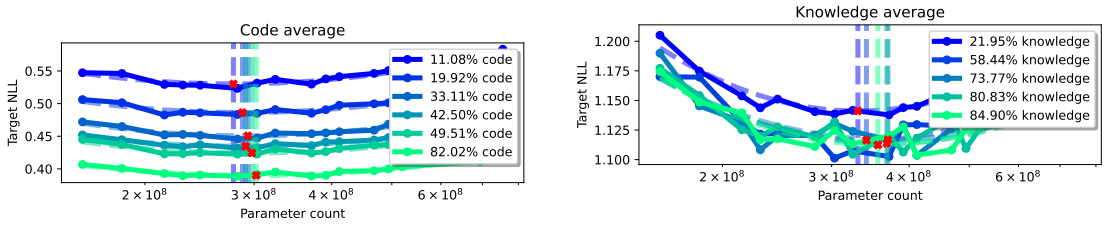
First, we consider the difference in COs for different skills for our canonical datamix.

Methodology. Roughly following the approach used by Grattafiori et al. [2024], we train models for compute budget between 6×10^{18} and 3×10^{21} FLOPs. At each compute scale, we pretrain models ranging in size between 40M and 8B parameters.[2] Using these models, we compute IsoFLOP curves and corresponding COs for the APE (validation loss) and the two skills under consideration – as measured by NLL of the target answer – using the methodology described in section 3.1.1. Here, in order to compare APE COs with the COs for skills, we first obtain the APE CO parameter counts, p_c from the power law fit given by Grattafiori et al. [2024], for each compute scale. Next, for a given skill s, for each dataset $\mathcal{D}$ belonging to that skill, we fit degree-2 polynomials for each compute scale and identify optimal parameter counts for p_s. We then project the APE COs onto the polynomial fits for s, and fit power law curves to the APE COs and the skill-dependent COs for s. In all our experiments, we first analyse the results on our hypothesis split and confirm our findings on the held-out split.

Results. In figs. 3.1(a) and 3.1(b), respectively, we show average results for the held-out code and knowledge QA datasets (in red). For comparison, we overlay the APE CO curve reported by Grattafiori et al. [2024] (in black). The figure shows a clear difference between knowledge QA and code: while knowledge QA prefers capacity compared to the APE curve – especially at lower compute scales – code tends to be more data-hungry. This pattern is present across all datasets belonging to the hypothesis split on which we first observed this difference, as well as for the held-out split. We show individual plots for all datasets in figs. 3.11 to 3.14.

To make the pattern more explicit across different scales, we consider the distribution of parameter count 'residuals,' or log-scale differences between the APE optima and the skill-optima for each skill. If at compute scale B, the residual is larger than 0, we say that the

[2]Details about model training can be found in section 3.B.



(a) Code ablation experiments, code mix.

(b) Knowledge QA ablation experiments, knowledge QA mix.

Figure 3.2: 6×10^{18} IsoFLOP curves for various code and knowledge QA datamixes. In (a), we scale the proportion of code pretraining data from $\sim 11\%$ to $\sim 82\%$ and see the losses improve and the COs shift toward capacity-hunger. For (b), we scale the proportion of knowledge from $\sim 22\%$ to $\sim 85\%$ and while knowledge tasks appear to be noisier than code, losses improve and COs shift.

skill is capacity-hungry at B; if it is smaller than 0, we call it data-hungry at B. The resulting distributions, shown in fig. 3.1(c), confirm the pattern we observed before: the parameter count residuals for code are shifted substantially to the left with respect to the parameter count residuals for knowledge QA, indicating that the latter skill is more capacity-hungry. The same pattern holds for all datasets in the hypothesis and held-out splits. This strongly supports the claim that there is a difference in COs between knowledge QA and code, and shows that this observation is a generalisable phenomenon.

3.3.2 Is this an artifact of the data distribution?

For our canonical datamix, we have seen a clear difference between the COs of knowledge QA and code datasets in their respective capacity- or data-hunger. What is not clear from our first experiments is whether these dissimilarities are the result of a *fundamental difference between the skills,* or if there is a disparity between the amount of data present for these skills in the canonical datamix.

Methodology. To begin to address this question, we first investigate if we can *shift* the skill COs by changing the proportion of relevant data for that skill, which are identified heuristically by tagging the relevant data sources in the pretraining data. Starting from the proportion of our canonical dataset (see fig. 3.7(a)), we create four additional datasets for knowledge and five additional datasets for code in which we up- or downsample the

proportion of skill relevant data. Specifically, for knowledge, we scale the original proportion of 58% down to 22% and up to 85%. Similarly for code, we scale the code proportion from their original 20% down to 11% and up to 84%. With these new datamixes, we train 16 models each with a budget of 6×10^{18} FLOPs, which is the smallest compute scale represented in our IsoFLOP curves, and recompute the skill CO for that compute scale.

Results. In fig. 3.2, we show the results for the code and knowledge held-out datasets. Unsurprisingly, increasing the amount of skill-dependent data (lighter colors in the plot), improves performance for that specific skill.[3] This pattern is much clearer for code than for knowledge. We hypothesise that this is because code datapoints are easily identified and contain only code, whereas knowledge datasets are usually more dispersed as well as more difficult to label automatically. More relevant to our research questions *is the fact that not only the losses, but also the COs shift.* Again specifically for code, this pattern is clear in both the hypothesis and held-out sets – the more code data is included, the more capacity-hungry the CO becomes. For reference, in section 3.E, we provide individual plots for all code and knowledge QA datasets of all splits, across all ablations. This implies a fundamental relationship between the skill proportions in the datamix and COs—for a fixed compute budget, if the data requirement for a skill is satisfied by increasing its proportion in the datamix, we can afford to train a larger model on less data.

However, we would like to assess whether or not the capacity- and data-hunger of knowledge and code are an artefact of the canonical datamix or if it is a more fundamental phenomenon. To do so, we use the same data to investigate the data-hunger of knowledge QA and code *independently* of the proportion of skill relevant data. To do so, we plot the CO parameter counts for each skill as a function of the proportion of skill relevant dta. In fig. 3.3, we show this for both knowledge QA and code in the same plot.[4] From this figure it is clear that even correcting for the proportion of skill relevant data, knowledge is substantially more capacity-hungry than code, and increases its capacity-hunger more quickly as the proportion of skill relevant data increases. We hypothesise that this phenomenon, *which*

[3]Increasing code data improved losses on code datasets *beyond the next compute scale on the canonical datamix.*

[4]The axis is aligned on proportions of skill-relevant data, not on the datamixes themselves. For knowledge QA datasets (purple lines), an x-value of 0.4 corresponds to a datamix with 40% knowledge, whereas for code datasets (green lines), it corresponds to results for the datamix with 40% *code.*

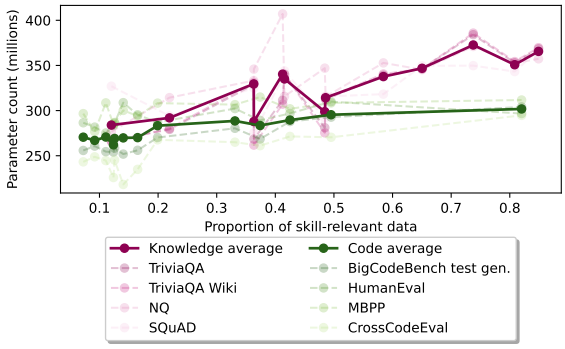


Figure 3.3: Optimal parameter count for 6×10^{18} models as a function of proportion of skill-relevant data. When the proportion of skill-relevant data is increased, both knowledge and code skills become more capacity-hungry. On average, knowledge-based tasks require more parameters than code given a particular proportion of skill relevant data, and the number of optimal parameters increases more quickly, implying a fundamental difference between the two skills.

demonstrates a fundamental difference between knowledge QA and code, is related to the fact that knowledge is harder to compress than code, requiring more capacity to memorise facts. The observation that COs for knowledge QA shift at a faster rate than code suggests a fundamental difference in their scaling, rather than an artefact of the datamix.

In line with this hypothesis, we observe a larger difference between knowledge and code when only Python code datasets are considered, as can be seen in fig. 3.4. We hypothesise that models at this scale may not be able to successfully compress 'low-resource' programming languages, though further research is required to confirm this.[5]

3.3.3 What does this imply for LLM training?

Having empirically shown and investigated the existence of skill-dependent COs, we now explore the practical implications of these findings for pretraining LLMs. While the research questions that could be explored in this realm are numerous, we focus on two concrete questions that can be feasibly tested within our compute budget:

[5]We also see in figs. 3.1 and 3.2 and sections 3.E and 3.I that datasets quantifying code have a lower loss value than those quantifying knowledge QA– across the board – despite large proportions of knowledge, which could suggest that code has *lower unmodelable entropy* compared to knowledge.

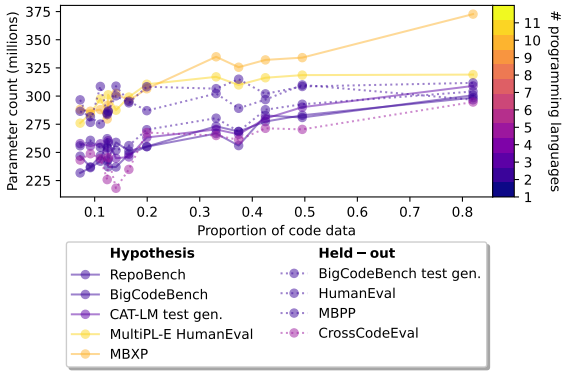
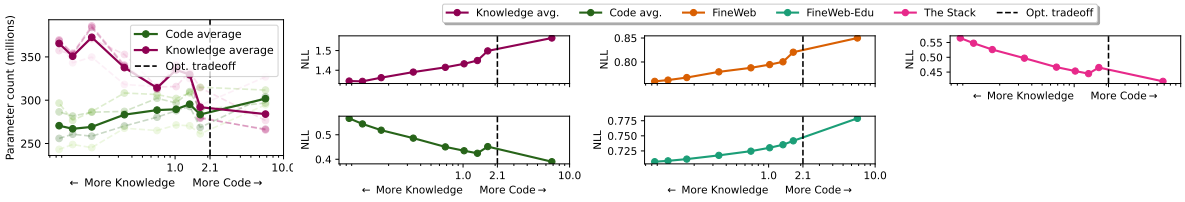


Figure 3.4: Optimal parameters for different coding benchmarks. Coding skills that contain more and thus less frequent programming languages, shown in yellow, appear to become more capacity-hungry, across all coding skills in our hypothesis and held-out splits.



(a) COs alignment.

(b) The impact of COs alignment on knowledge QA, code, and validation sets.

Figure 3.5: Optimal parameter count and loss as a function of code/knowledge ratio. (a) Optimal parameter count for 6×10^{18} models, as a function of the ratio between code and knowledge in the pretraining data. Dashed lines indicate the 'crossover point,' indicating the ratio for which the optimal parameter counts align. (b) Evaluation and validation loss for the same models, with the same optimal cross-over point. Every validation set that we consider exhibits losses that closely follow *either* the average knowledge QA or average code curves.

1. Can we align COs of skills via data selection to improve the validation loss?

2. What is the parameter count impact of a validation set measuring the wrong skills?

3.3.3.1 Aligning skill-COs

Given that COs of knowledge-related skills are more capacity-hungry while code related skills instead benefit more from data, we now ask the question if we can improve the overall aggregate performance of a model by shifting the datamix such that the COs of the two skills

are aligned. In fig. 3.5(a), we plot the optimal parameter count for the 6×10^{18} models we trained for the results reported in the previous subsection. We can see that there is indeed a point where the optimal parameter counts for code and knowledge roughly match, which is when code data is approximately 2.1 times as prevalent as knowledge data.[6] This is in line with our earlier results: as we add more code data, the CO for code will very gradually shift towards being more parameter-hungry, whereas decreasing the amount of knowledge data shifts the CO for knowledge to require fewer parameters.

Next, we look at how this trade-off impacts the validation loss, indicative of the aggregate model performance. Since our original APE COs were projected from the scaling law parameters given by Grattafiori et al. [2024], we would ideally conduct this analysis using the Llama 3 validation set. However, details of this validation set are unreleased, so, instead, we sample three different validation datasets from different publicly available pretraining datasets: FineWeb, FineWeb-Edu [Penedo et al., 2024a] comprising educational webpages filtered from FineWeb, and The Stack Kocetkov et al. [2022], a code-based pretraining dataset. We randomly draw 20K samples from each of these datasets to produce a much smaller validation set. For FineWeb[7] and FineWeb-Edu,[8] we draw the samples from their 10B token sample splits. For The Stack, we draw the samples from the subsampled The Stack Smol, containing a small subset of approximately 0.1% of The Stack.[9]

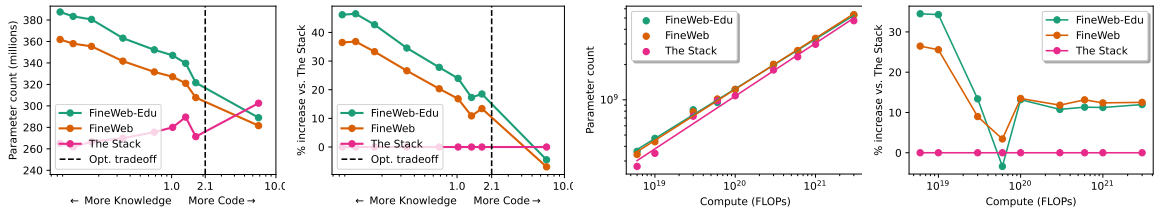
In fig. 3.5(b), we show the NLL for all validation sets, using the same axis as in fig. 3.5(a). The vertical dashed line indicates the optimal point from fig. 3.5(a). A first striking observation is that all the chosen validation sets track either knowledge QA or code quite closely: while the loss on FineWeb and FineWeb-Edu monotonically increases with the amount of code data, the opposite is true for The Stack. We do not see evidence of a mix of knowledge QA and code in the validation sets, which might have presented itself as a U-shaped curve, rather than the monotonically increasing or decreasing curves that we observe. To compute appropriate scaling laws, none of these validation sets would thus have resulted in COs aligned with *both* skills. This result implies that it is possible to align COs between skills, and that this should be done in practice to balance desired skills.

[6] We obtain this value of 2.1 by computing the crossover point on the hypothesis set and overlaying it onto the held-out splits – we include a comparison between the crossover points found on the hypothesis and held-out splits in section 3.G.

[7] `https://huggingface.co/datasets/HuggingFaceFW/fineweb`

[8] `https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu`

[9] `https://huggingface.co/datasets/bigcode/the-stack-smol`



(a) **(b)** **(c)** **(d)**

Figure 3.6: Fluctuation in optimal parameter count as function of validation set. In (a), we show the COs of the validation sets that we evaluate, showing that at a high ratio of knowledge QA to code, two of the three validation sets have COs with substantially higher parameter counts. In (b), we see this information in terms of a relative increase in parameter count compared to The Stack, and see that the increase can be up to roughly 50%. In (c), we show the CO scaling behaviour across compute scales between validation sets. In (d), we see the relative parameter count increases over The Stack, and see that at high compute scales, the increase is smaller, yet it still exceeds 10%.

3.3.3.2 The impact of the validation set

Lastly, we study more explicitly how much variation in the predicted COs is to be expected depending on the choice of validation set. In fig. 3.6(a), considering The Stack validation set as 'baseline,' we can see that for some datamixes, the optimal parameter count is off by almost 50%. In fig. 3.6(c), we see that, for the smallest compute scale, for the canonical datamix there is over a 30% difference in optimal parameter count between the different validation sets. While the relative difference is not as stark at higher compute scales, it still exceeds 10% for the 3 largest compute scales. This experiment shows that choosing a validation set that adequately represents a mix of what the final model should capture – or perhaps even a validation set that more directly measures those skills, such as in our work – is critical to finding the right COs. While the difference is more pronounced at smaller scales, it persists even on the largest compute scale in our experiments.

On the other hand, Grattafiori et al. [2024] points out that IsoFLOP curves get flatter at high compute scales, so the tradeoff between model capacity and dataset size might not matter as much. In one of their other results, Grattafiori et al. [2024] uses a fitted sigmoid of the loss to estimate *accuracies*. If one were to take their analysis a step further by applying a sigmoid to an entire IsoFLOP curve to produce an IsoFLOP curve of accuracy rather than NLL, we hypothesise that this could make curves appear *less flat* if low compute scales are in the flat region of the sigmoid and high compute scales are in the linear region. We leave

this intriguing question to future work.

3.4 Related work

Scaling laws and compute optimality. The first large-scale empirical study of statistical scaling laws of neural networks was done by Hestness et al. [2017], who studied scaling laws for a variety of tasks using the LSTM Hochreiter and Schmidhuber [1997b] and CNN Lecun et al. [1998] architectures. Their paper used power law fits to their experimental data, and they found that the only way to materially change the power law exponent was to *change the task*, whereas other choices such as hyperparameters changed only the scaling factor. Later works extended these ideas to joint model and dataset size scaling, such as those by Rosenfeld et al. [2019], Kaplan et al. [2020a], and famously, Hoffmann et al. [2022c], who introduced the notion of compute-optimal scaling measured using the loss on the training set. More recently, several efforts aim to predict downstream performance using scaling laws [Isik et al., 2025, Bhagia et al., 2024a], but they focus on forecasting downstream metrics, whereas our work asks a fundamental question about how different skills exhibit different scaling behaviour. Related to these efforts is a concurrent work that characterises LLM skills as latent variables of the scaling model fit to benchmark scores of existing pretrained models, but differs from our work in its treatment of skills and crucially, that it does not study compute optimality Polo et al. [2025].

Data selection. In addition to scaling laws, our work is related to the rich literature on data selection and optimising the pretraining datamix. Inspired by scaling laws, the recent Data Mixing Laws Ye et al. [2025] aim to model the loss as a function of the proportions of different types of data in the pretraining datamix. Other work attempts to improve dataset quality and therefore downstream performance by optimising the datamix Chen et al. [2025], Xie et al. [2023a], Held et al. [2025], as well as Skill-It Chen et al. [2023a], which dynamically updates the dataset proportions throughout training using downstream validation losses and an online mirror descent algorithm. However, these works do not focus on how COs might shift as a function of the pretraining datamix.

Skill frameworks. Skill-It formalises the downstream losses that it targets as 'skills,' which we draw inspiration from in our work. In Skill-It, a skill is defined as a unit of

behaviour associated with a dataset and a metric, such that if a model is trained on samples from the dataset, its metric improves on unseen samples. However, there are other related definitions of skills that include more general notions of behaviour such as reasoning abilities Xia et al. [2024], Arora and Goyal [2023] and the evaluation thereof Yu et al. [2024].

3.5 Conclusion

In this work, motivated by the impact that scaling laws have had on the landscape of training LLMs, we examine the existence of *skill-dependent scaling laws*. Specifically, we investigate how the optimal trade-off between capacity (in the form of parameter count) and training data varies across skills. Focusing on two different skills – code generation and knowledge-based QA – we find pronounced differences in how *data-* or *capacity-hungry* they are: where knowledge QA is capacity-hungry, code instead prefers data. These differences hold across tasks and splits, and are true even if the proportion of *skill-dependent data* in the pretraining datamix is factored out. Furthermore, as the proportion of skill-dependent data grows, the capacity-hunger of knowledge-based skills grows more rapidly than that of code, indicating that models are able to compress the latter better than the former.

In the second part of our experiments, we focus on the impact of these findings to LLM training. We find that both the use of validation set and the specific datamix used to do scaling experiments have a substantial impact on the estimated compute optimal points. In our 6×10^{18} FLOP experiments, we find that varying the ratio of code and knowledge data can result in a difference of 30% or more in the estimated optimal parameter count for that budget for a single validation set, and almost 50% depending on the validation set. Similarly, the use of validation set can result in a mismatch of over 30% on the smallest scale, and over 10% on the largest three scales (up to 3×10^{21}). As such, our experiments show that choosing a validation set that adequately represents a mix of what the final model should capture is critical to finding the right COs as well as that doing data ablations *after* selecting COs may result in suboptimal parameter counts for the specific datamix.

Our analysis also opens intriguing directions for future study, such as understanding the relationship between compression and skills through the lens of COs, as well as understanding the flatness or curvature of IsoFLOP curves computed on the NLL compared to accuracy-based metrics. Crucially, we note that the bulk of our analysis can be easily

extended to additional skills using open source scaling suites, as it only requires running evaluations on pretrained models.

3.A Architecture details

All of the models used in our experiments are based on the Transformer architecture Vaswani et al. [2017a], using a similar configuration to that used by Grattafiori et al. [2024]. We scale the models to along various axes to increase the parameter counts: embedding dimension, number of attention heads, and number of layers. The number of layers that we use in our models ranges from 8 to 50, the number of heads ranges from 8 to 40, and the embedding dimensions range from 512 to 5120.

3.B Model pretraining

We follow the training recipe described in Grattafiori et al. [2024] for their scaling law experiments. Specifically, we use a cosine learning rate schedule with 2000 steps of linear warmup from 0 and a maximum learning rate of $\eta \in [2 \times 10^{-4}, 4 \times 10^{-4}]$, which is tuned for different model sizes, that decays to a final value of $\frac{\eta}{10}$. Again following Grattafiori et al. [2024], we use a weight decay rate of 0.1. Similarly, we vary batch sizes across compute scales from 250K to 4M tokens, keeping it constant within each scale.

3.C Definitions

Here, we provide formal definitions of the notions of IsoFLOP groups, APE COs, skill COs, and capacity- vs data-hungry skills that we discussed or introduced in section 3.1.

3.C.1 IsoFLOP groups and APE COs.

Formally, IsoFLOP groups, which serve as the building block of IsoFLOP curves, are defined as follows.

Definition 3.1 (IsoFLOP group)**.** *An IsoFLOP group at a compute scale of* B *FLOPs is an isomorphism class on the set* $G_B = \{f_t^{(p)} \in \mathcal{F} : \textit{training } f_t^{(p)} \textit{ for } t \textit{ tokens required } B \textit{ FLOPs}\}$ *with*

the following two isomorphisms: for a given $f_t^{(p)} \in G_B$ *and decreasing* p *to* $p' : 0 < p' < p$*, there is a unique* $f_{t'}^{(p')} \in G_B$ *such that* $0 < t < t'$*, and for* $f_t^{(p)} \in G_B$ *and increasing* p *to* $p' : 0 < p < p'$*, there is a unique* $f_{t'}^{(p')} \in G_B$ *such that* $0 < t' < t$.

Next, we define the *APE optimum* as the CO as measured by performance on a general validation set $\mathcal{X}_{\text{validation}}$ drawn from the same distribution as the training dataset $\mathcal{X}_{\text{train}}$, as follows.

Definition 3.2 (APE CO)**.** *A parameter count* p^* *and* t^* *training tokens is called APE CO for a dataset* $\mathcal{X}_{validation}$ *at a compute scale of* B *FLOPs if* $f_{t^*}^{(p^*)} \in G_B$ *has improved metric* $\mathcal{L}$ *on samples from* $\mathcal{X}_{validation}$ *on average compared to any other element of* G_B.

3.C.2 Skill COs and capacity- vs data-hunger.

Next, we move on to definitions of notions that are specifically introduced in our work. First, we define COs for skills, and then we define capacity- and data-hunger, which relate skill COs to APE COs.

Definition 3.3 (Skill CO)**.** *A parameter count* p^* *and* t^* *training tokens is called compute-optimal for a dataset* $\mathcal{D}$ *that quantifies a skill* s *at a compute scale of* B *FLOPs if* $f_{t^*}^{(p^*)} \in G_B$ *has improved metric* $\mathcal{L}$ *on samples from* $\mathcal{D}$ *on average compared to any other element of* G_B.

We call a skill capacity- or data-hungry at a particular compute scale B depending on how it compares to the APE optimum as reported by Grattafiori et al. [2024] for B. Formally, we define these as follows.

Definition 3.4 (Capacity-hungry at B)**.** *A CO skill* s *with* p_s *parameters and* t_s *training tokens is called capacity-hungry at a compute scale of* B *FLOPs if, compared to the APE CO at* B*, specified by* p_c *and* t_c*, we have that* $p_s > p_c$ *and correspondingly,* $t_s < t_c$.

Conversely, if a skill CO is biased toward more data than the APE CO, we have the following definition.

Definition 3.5 (Data-hungry at B)**.** *A CO skill* s *with* p_s *parameters and* t_s *training tokens is called data-hungry at a compute scale of* B *FLOPs if, compared to the APE CO at* B*, specified by* p_c *and* t_c*, we have that* $t_s > t_c$ *and correspondingly,* $p_s < p_c$.

3.D Evaluation dataset details

In this section, we describe the usage of MMLU and SQuAD in our hypothesis and held-out splits.

3.D.1 MMLU splits

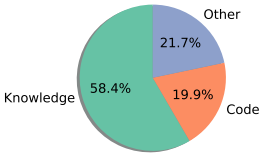
Our hypothesis split of MMLU Hendrycks et al. [2021a] includes Human Sexuality, Jurisprudence, Machine Learning, Marketing, Misc., Nutrition, Prehistory, Professional Psychology, Security Studies, US Foreign Policy, and World Religions, The held-out MMLU split contains Astronomy, Clinical Knowledge, College Chemistry, College Physics, Conceptual Physics, Electrical Engineering, High School Biology, High School European History, High School Government and Politics, High School Microeconomics, High School US History, Human Aging, International Law, Logical Fallacies, Management, Medical Genetics, Moral Disputes, Philosophy, Professional Medicine, Public Relations, Sociology, and Virology. We excluded all other MMLU topics, as they were either reasoning-based, or it was unclear whether they primarily evaluated knowledge QA.

3.D.2 SQuAD without context

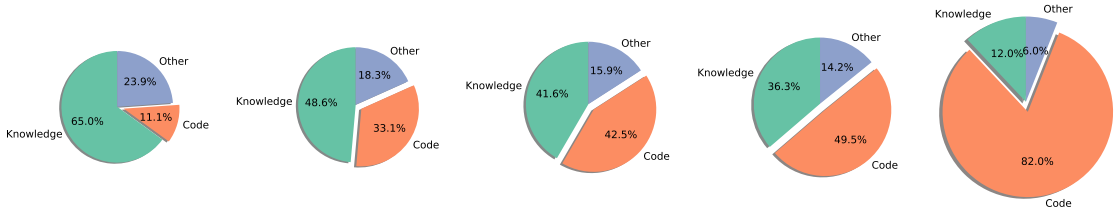
The SQuAD dataset Rajpurkar et al. [2016] involves retrieving knowledge from a context. However, in this work, we are primarily interested in knowledge QA-based skills that retrieve memorised facts from the training set, and not from a provided context (which may constitute an entirely different skill, such as in-context learning). In order to convert the SQuAD questions into a useable form for our purposes, we omit the context from our prompts and rely only on information learned during training and stored in the weights to answer the questions. We also discard all samples for which the answer was not contained in the context. We include a subsampled portion of the SquAD training set with 12000 samples in our hypothesis split and the standard SQuAD dev set in the held-out split.

3.E Datamix ablation proportions

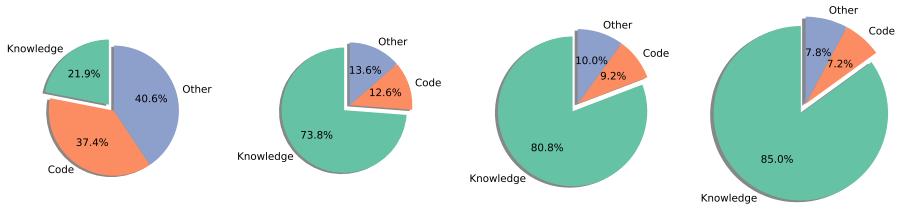
We provide our canonical datamix proportions in fig. 3.7(a), the different proportions used during code scaling in fig. 3.7(b), and the proportions used for knowledge scaling in fig. 3.7(c).



(a) Canonical datamix proportions.



(b) Code scaling proportions.

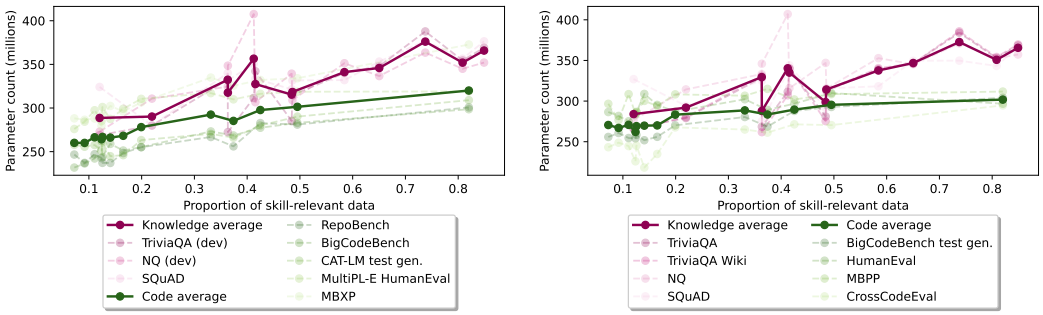


(c) Knowledge scaling proportions.

Figure 3.7: Pretraining datamix proportions used in our experiments. In (a), these are the canonical datamix proportions, (b) shows the code scaling ordered by increasing code, and in (c), our knowledge QA scaling proportions ordered by increasing knowledge.

3.F Hypothesis split skill-relevant data scaling

In fig. 3.8, we include a comparison of our skill-relevant data scaling analysis between the hypothesis and held-out splits. Our conclusion from the hypothesis split indeed held on the held-out split, which we reported in the main text.



(a) Hypothesis split.

(b) Held-out split.

Figure 3.8: Comparison between hypothesis and held-out skill-relevant data scaling analyses. For both the hypothesis and held-out splits, we find that knowledge QA exhibits a faster increase in optimal parameter count as the proportion of skill-relevant data increases, implying that the capacity-hunger of knowledge QA is fundamental and not an artefact of the datamix.

3.G Hypothesis and held-out crossover points for aligning the COs

In addition to the held-out set results in the main text, we also include hypothesis set results for our analysis in which we computed the optimal alignment of knowledge QA and code COs in fig. 3.9. We note that the crossover points that align the two COs are similar between the hypothesis and held-out sets. We report the hypothesis crossover point in the main text to avoid a priori access to the held-out split.

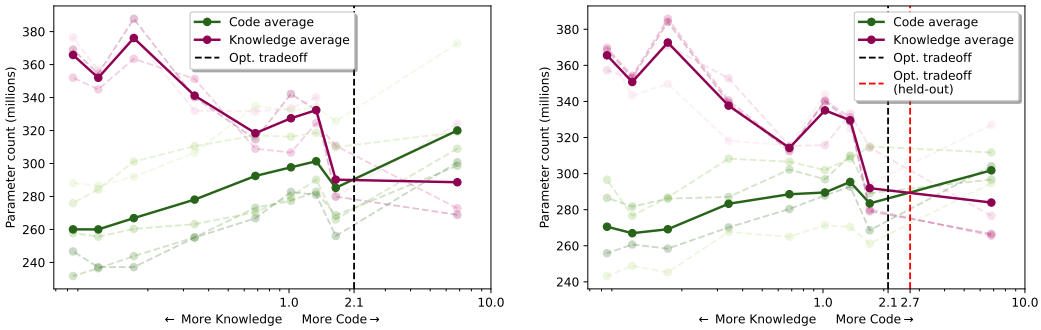
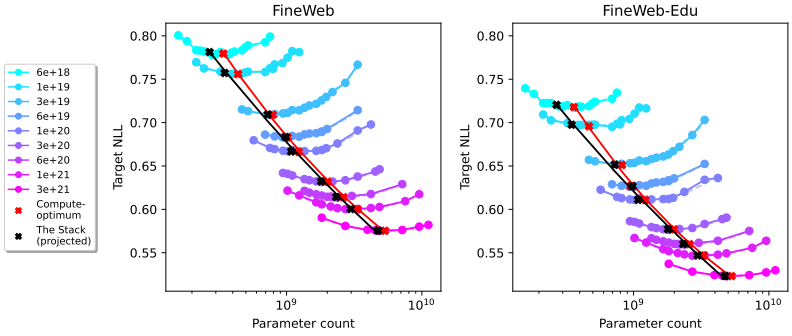


Figure 3.9: Comparison between hypothesis set and held-out set code/knowledge scaling ratios. We compute the crossover point that aligns the COs of the two skills on the hypothesis split (2.1) and find that the crossover point for the held-out split is similar (2.7). In the main text, we report the hypothesis set crossover point since we do not assume a priori access to the held-out set.

3.H IsoFLOP curves for validation sets

In fig. 3.10, we show the IsoFLOP curves of the alternative validation sets – FineWeb and FineWeb-Edu – compared to The Stack. We find that they are all capacity-hungry compared to The Stack, which supports our findings that The Stack is more code-aligned while the others are mode knowledge QA aligned.



(a) FineWeb validation. **(b)** FineWeb-Edu validation.

Figure 3.10: APE COs and IsoFLOP curves of alternative validation sets. We show the APE optima for various validation sets. All validation sets are subsampled from open source pretraining datasets, and compared to a validation set subsampled from The Stack.

3.I Per-benchmark IsoFLOP curves

We include per-benchmark IsoFLOP curves and residual distribution plots in Figures figs. 3.11 to 3.14. Every benchmark in our hypothesis and held-out sets were capacity-hungry if they were knowledge QA-based and were data-hungry if they were code-based. We note that we excluded SWE-Bench (Oracle) from our final averages because its lowest two compute scales were highly skewed towards data-hunger (which is supported by SWE-Bench being code-based), but the skew was so extreme that the optima was outside of the parameter count ranges of the models used to estimate the curves.

3.J Per-benchmark datamix ablation curves

We provide the per-benchmark plots for our datamix proportion scaling experiments in Figures figs. 3.15 to 3.18. As with our main results, we find that when we increase the proportion of knowledge QA in the datamix, we see improved losses for knowledge QA-based datasets and when we increase the proportion of code in the datamix, we see improved losses for code-based datasets, and vice versa. We also see that the COs shift toward capacity-hunger as we increase the proportion of skill-relevant data. Again, we note that we removed SWE-Bench (oracle) from the averages because the 6×10^{18} compute scale was highly skewed outside of the empirical range.

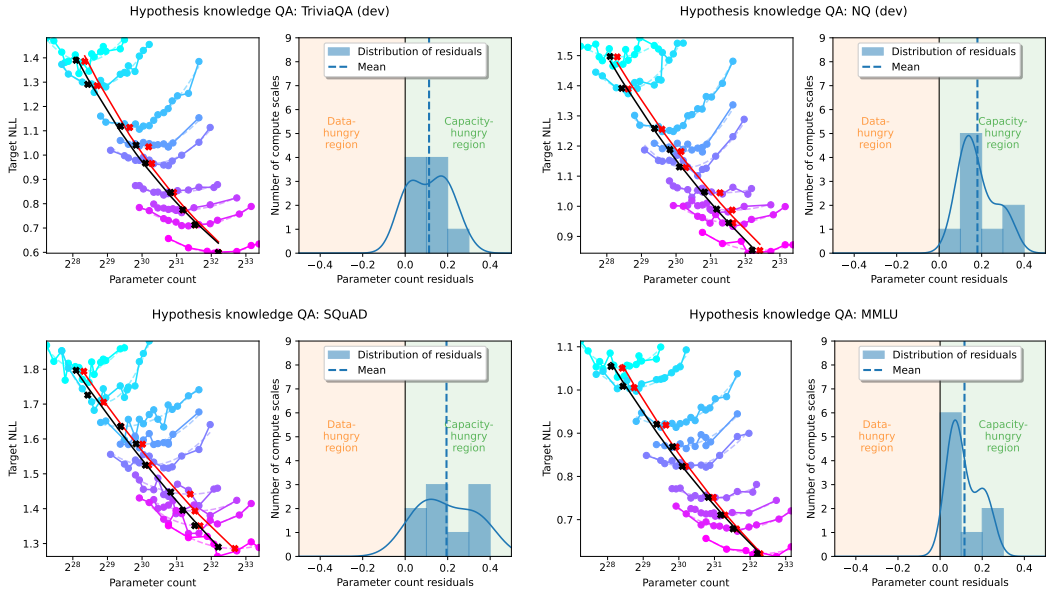


Figure 3.11: Hypothesis split knowledge QA skills. On the hypothesis split, on which we formed our initial hypotheses about knowledge QA vs code scaling, we found that every knowledge QA dataset exhibited capacity-hungry scaling.

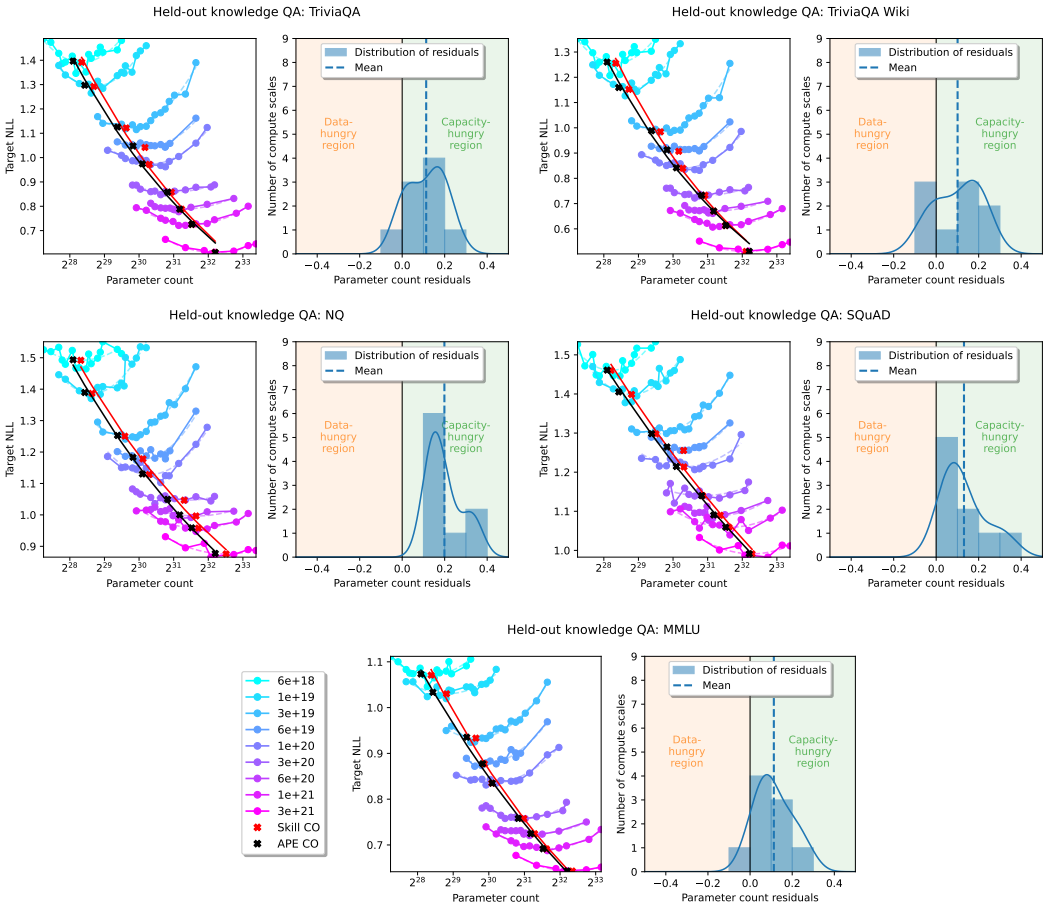


Figure 3.12: Held-out split knowledge QA skills. On the held-out split, which we did not access while forming our hypotheses, we found that every knowledge QA dataset also exhibited capacity-hungry scaling.

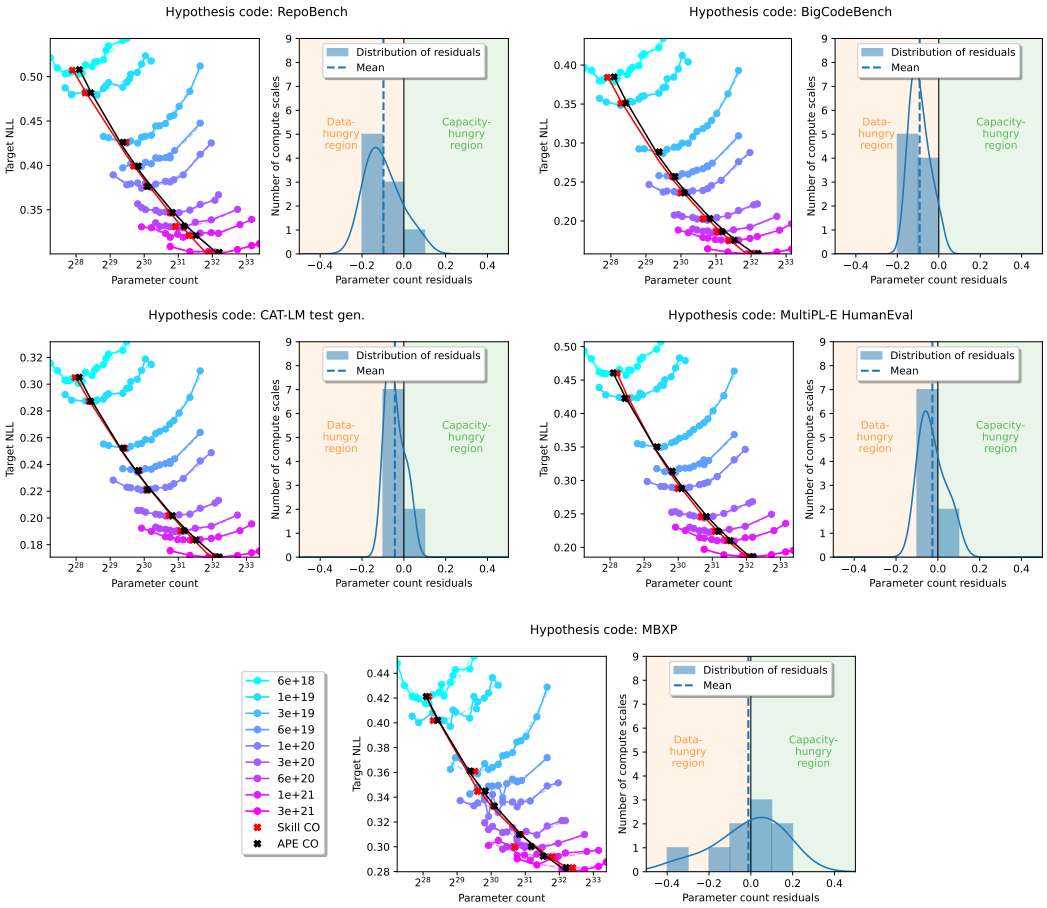


Figure 3.13: Hypothesis split code skills. On the hypothesis split, on which we formed our initial hypotheses about knowledge QA vs code scaling, we found that every code dataset exhibited data-hungry scaling.

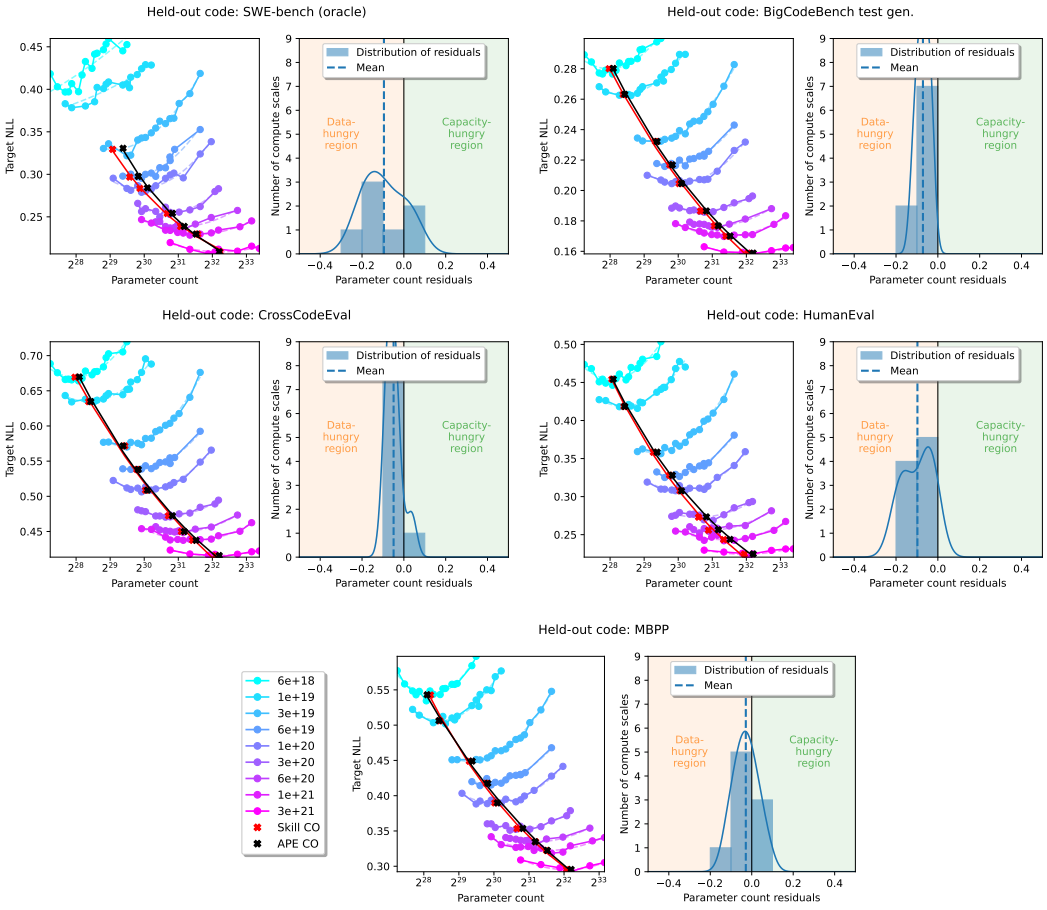


Figure 3.14: Held-out split code skills. On the held-out split, which we did not access while forming our hypotheses, we found that every code dataset also exhibited data-hungry scaling.

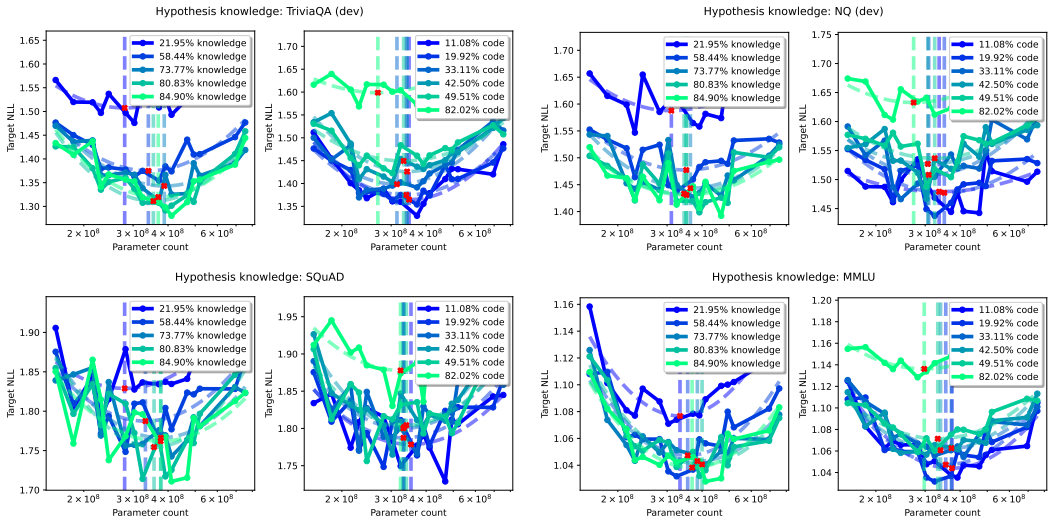


Figure 3.15: Data mix scaling curves for hypothesis split knowledge QA skills. For hypothesis knowledge QA datasets, loss improves and COs shift to higher parameter counts with more knowledge and vice versa with code.

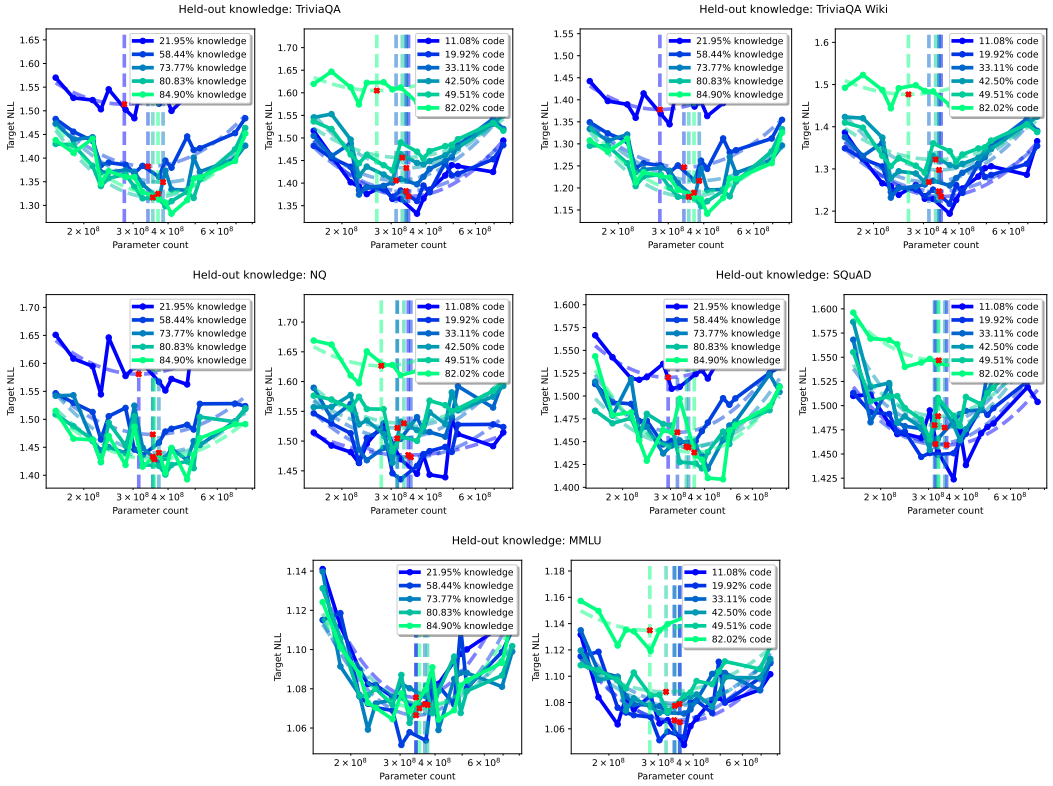


Figure 3.16: Data mix scaling curves for held-out split split knowledge QA skills. For held-out knowledge QA datasets except for MMLU, loss improves and COs shift to higher parameter counts with more knowledge and vice versa with more code. We attribute the noise in the MMLU results to the fact that losses are averaged across MMLU categories.

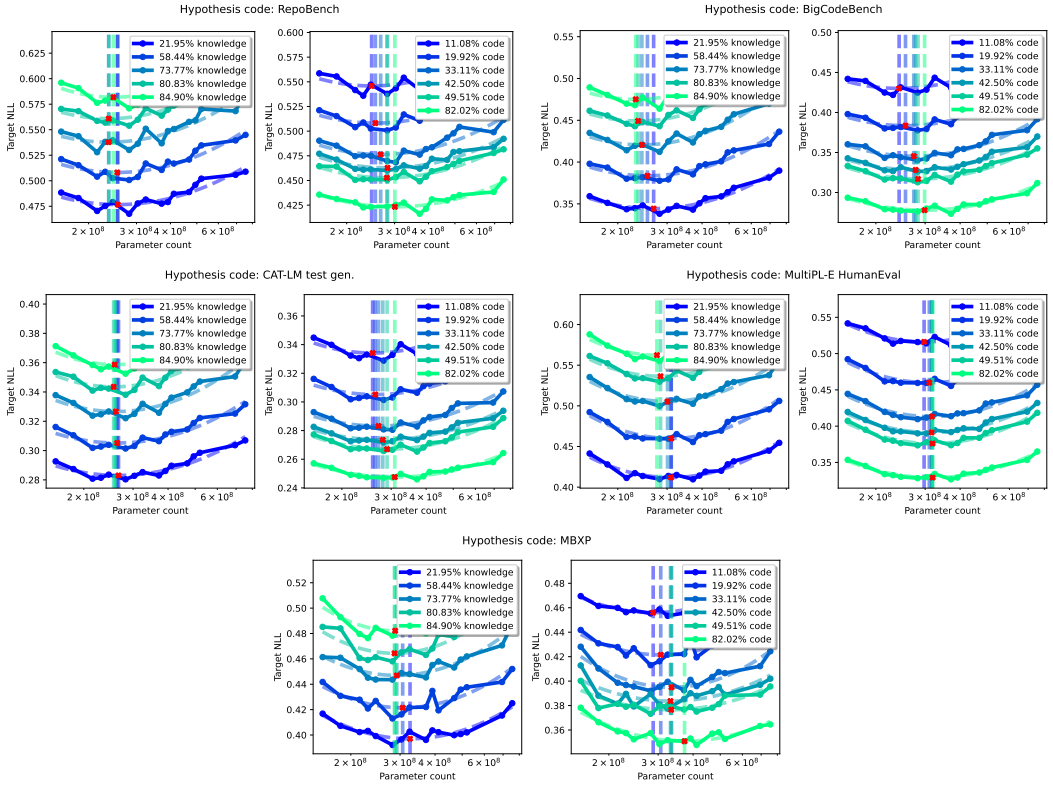


Figure 3.17: Data mix scaling curves for hypothesis split code skills. For hypothesis code datasets, loss improves and COs shift to higher parameter counts with more code and vice versa with knowledge. This pattern is much more clear on code datasets than on knowledge QA datasets.

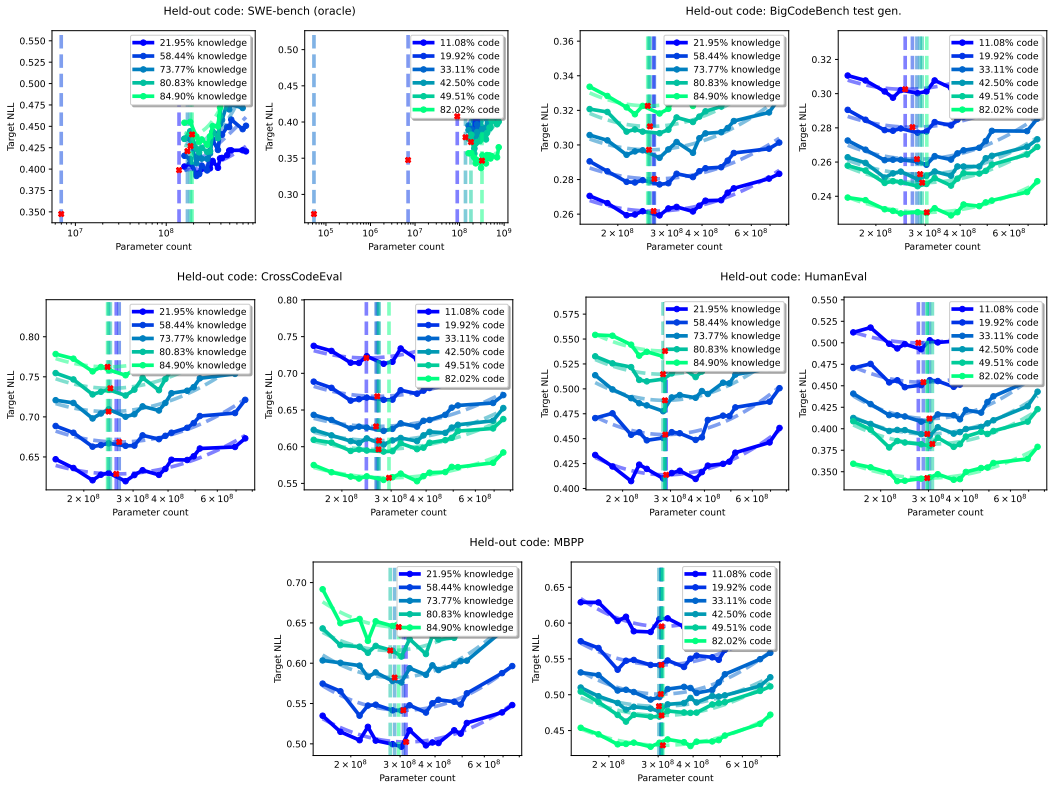


Figure 3.18: Data mix scaling curves for held-out split code skills. For held-out code datasets except for SWE-Bench (oracle), loss improves and COs shift to higher parameter counts with more code and vice versa with knowledge. Note that SWE-Bench (oracle) was ultimately excluded from this analysis, as its lowest compute scale was so skewed that the estimated optima were typically outside of our empirical range.

Chapter 4

Pretrained Hybrids with MAD Skills

Transformers are the workhorse architecture for large language models and beyond, powering a vast collection of foundation models. While for years it appeared that the Transformers family would remain the undisputed standard, a recent *Cambrian explosion* of proposed architectures has taken place. Many of the new architectures achieve subquadratic complexity—in contrast to the quadratic complexity of self-attention in Transformers—by using local or linear attention [De et al., 2024, Botev et al., 2024, Arora et al., 2024, Zhang et al., 2024], resurrecting and scaling recurrent networks [Botev et al., 2024, De et al., 2024, Peng et al., 2023], or by building on state-space modeling principles [Gu and Dao, 2023, Poli et al., 2023b,a, Fu et al., 2023, Gu et al., 2022]. These approaches potentially promise to overturn the dominance of Transformers through more efficient training and inference.

However, no single new model is a clear overall winner when varying data modalities, tasks, and model sizes. Comparing architectures on a fixed task is fraught with difficulties [Amos et al., 2024]. Even if these are overcome, practitioners would have to experiment with and evaluate every architecture for each new task—an expensive proposition. Instead, seeking a best-of-all-worlds approach, researchers have proposed the use of *hybrid models* that mix multiple architectures. These hybrids, such as the MambaFormer [Park et al., 2024]—a mix of the popular SSM Mamba architecture with a standard Transformer—have shown potential in maintaining the desirable properties of multiple model classes.

While promising, hybrids suffer from two main obstacles that stymie their adoption:

- Manual Design. Hybrid architectures are hand-crafted, either by manually exploring the large search space of hybrids or by relying on often unreliable intuition and heuristics.

- Failure to Use Pretrained Models. It is unclear how to integrate *pretrained* components from models with different architectures. Pretrained models are a key advantage of foundation models, but due to compatibility issues, hybrids are often trained from scratch, which is both limiting and costly.

A potential solution to the latter challenge is the use of *model merging* [Yadav et al., 2023, Yu et al., 2023, Wortsman et al., 2022, Ilharco et al., 2023, Davari and Belilovsky, 2023, Jang et al., 2024], some of which can operate cross-architecture [Akiba et al., 2024, Goddard et al., 2024]. Unfortunately, such tools are embryonic–they are expensive and it is unclear how well they work with the diverse architectures a user may seek to build a hybrid from.

We propose a framework for automatically designing hybrid architectures that overcomes these obstacles. Our approach is inspired by principles from neural architecture search (NAS), but applies these at the level of *LM blocks* rather than convolutional cells [Liu et al., 2019c, Li et al., 2021b] or operations [Shen et al., 2022, Roberts et al., 2021c]. The resulting framework is simple, tractable, and it sidesteps merging different architectures by using simple projectors to translate between the "languages" spoken by various architectures. This enables us to include blocks from many different architectures/models with no changes required. Furthermore, inspired by the mechanistic architecture design framework (MAD) [Poli et al., 2024], we show how to learn hybrids via MAD that transfer to new tasks.

Concretely, with our proposed system, Manticore, we:

1. Automatically select language models, without training several models from scratch,
2. Automatically construct pretrained hybrids without evaluating the entire search space,
3. Explore when it is possible to program hybrids without full training.

Experimentally, our automatically designed hybrids compete with existing hybrids and models on the MAD tasks [Poli et al., 2024] and Long Range Arena [Tay et al., 2021, LRA], we produce pretrained hybrids that improve downstream fine-tuning performance on a variety of language tasks, and we show that Manticore can be programmed using MAD.

4.1 Methods

We now describe Manticore, our framework for automatically designing hybrid architectures by mixing components of pretrained models. We rely on projectors to align features across architectures, then apply a convex combination to aligned features, as shown in Figure 4.1.

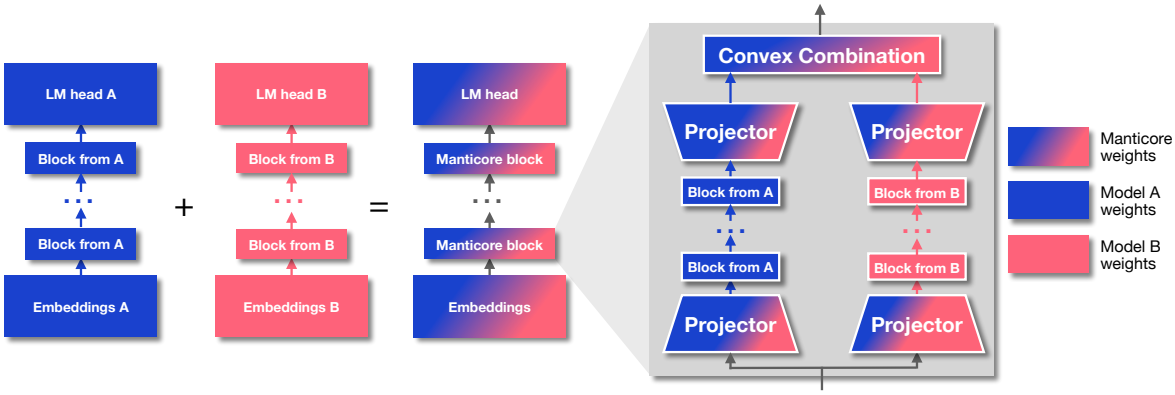


Figure 4.1: Manticore enables: (1) cross-architecture LM selection, (2) the construction of pretrained hybrids, and (3) the ability to program hybrids to have certain skills.

In Section 4.1.1, we discuss and formally define the structure of Manticore hybrids: the projectors and convex combination mixture weights, as well as how both of these components are used within Manticore. In Section 4.1.2, we detail the NAS-inspired search procedures and training routines involved in pretraining, fine-tuning, and programming hybrids. Finally, we provide the synthetic and real data settings that we use in our experiments in Section 4.2.

4.1.1 The Structure of Manticore Hybrids

Our framework comprises three main parts: the individual LMs that we combine to produce our overall hybrid, projectors that translate feature representations between LMs of different architectures, and convex combination mixture weights that specify how much the hybrid will use the features of each component architecture. We detail each of these in the following.

Component Models. We refer to a model that is used in Manticore as a *component model*. Any modern decoder-only LM can be used as a component model in our framework. In this section, we will formally define the general high-level structure of the component models that we support. For an LM M with model embedding dimension d_M on a sequence of t tokens from a set $\mathcal{V}$, denoted $x = (x_1, ..., x_t) \in \mathcal{V}^t$, a forward pass $M(x)$ is typically computed using the following recipe:

1. Apply an embedding function, $M_{\text{embed}} : \mathcal{V}^t \to \mathbb{R}^{t \times d_M}$ to the tokens, resulting in a sequence of embeddings denoted $x_{\text{embed}} = M_{\text{embed}}(x)$.

2. Take forward passes through L_M 'blocks'–we denote the ℓ^{th} block as $M^{(\ell)}_{\text{Block}} : \mathbb{R}^{t \times d_M} \to \mathbb{R}^{t \times d_M}$. Specifically, for all $\ell \in [L_M]$, we obtain $x_{\ell+1} = M^{(\ell)}_{\text{Block}}(x_\ell)$, where $x_1 := x_{\text{embed}}$.

3. Finally, we pass x_{L_M+1} into a language modeling head, $M_{\text{head}} : \mathbb{R}^{t \times d_M} \to (\Delta^{|\mathcal{V}|-1})^t$, where $\Delta^{|\mathcal{V}|-1}$ is the probability simplex of dimension $|\mathcal{V}|$.

This recipe applies to virtually all transformer-based LMs, recurrent models, and state-space models. Manticore supports all of these and any architecture that follows this recipe.

Projectors. Suppose we have pretrained component models M and M'. Assuming that the model dimensions are the same for both models ($d_M = d_{M'}$), blocks from M and M' may not be compatible, as their input and output features are distributed differently. It is also possible that $d_M \neq d_{M'}$, in which case composing blocks from M and M' is not well defined.

To overcome this issue, we apply projectors to both the inputs and the outputs of a block (or a sequence of blocks, discussed in Section 4.1.1) that we wish to combine in Manticore hybrids. Overall, our goal in designing projectors is to enable the blocks of M and M' to *share a common representation*, such that their features are compatible and can be reused in the resulting hybrid model. This is conceivably challenging—the mapping between feature spaces could be highly nonlinear and might require a lot of task-specific data to adequately learn the mapping. If the mapping is indeed highly nonlinear, we might need heavyweight multi-layer projectors with a large number of parameters. This could substantially increase parameter counts, inference cost, and could increase the data requirement for learning them. So do projectors need to be heavyweight, data-hungry, highly nonlinear objects? Fortunately, we find that the answer is no—we find that a simple linear transformation with a gated residual, pretrained on general language data, is sufficient.[1]

Suppose that we want to create a Manticore hybrid from K different pretrained component models, denoted $M_{(1)}, ..., M_{(K)}$ with model dimensions $d_{M_{(1)}}, ..., d_{M_{(K)}}$. We define $d_{\max} := \max_{k \in [K]} d_{M_{(k)}}$, then want *input* and *output* projectors for the blocks of each model that convert their features to a common feature space of dimension $d_{\max}$. For any sequence

[1] When this gating is combined with Equation 4.1, we see that the use of gated residuals ensures that the component architectures are still in our search space. This is a convenient property that allows Manticore to fall back on a component model when it outperforms hybrids.

of blocks of length $(n+1) < L_{d_{M_{(k)}}}$ from model $M_{(k)}$ and length-t input,

$$\left(M^{(\ell+n)}_{(k)\text{Block}} \circ \ldots \circ M^{(\ell)}_{(k)\text{Block}}\right) : \mathbb{R}^{t \times d_{M_{(k)}}} \to \mathbb{R}^{t \times d_{M_{(k)}}},$$

we want $\text{Proj-in}^{(\ell)}_{(k)} : \mathbb{R}^{t \times d_{\max}} \to \mathbb{R}^{t \times d_{M_{(k)}}}$ and $\text{Proj-out}^{(\ell+n)}_{(k)} : \mathbb{R}^{t \times d_{M_{(k)}}} \to \mathbb{R}^{t \times d_{\max}}$, so that

$$\left(\text{Proj-out}^{(\ell+n)}_{(k)} \circ M^{(\ell+n)}_{(k)\text{Block}} \circ \ldots \circ M^{(\ell)}_{(k)\text{Block}} \circ \text{Proj-in}^{(\ell)}_{(k)}\right) : \mathbb{R}^{t \times d_{\max}} \to \mathbb{R}^{t \times d_{\max}}.$$

For input $x \in \mathbb{R}^{t \times d_{M_{(k)}}}$ we parameterize projectors as linear layers with gated residuals:

$$\text{Proj-in}^{(\ell)}_{(k)}(x; \alpha) := (1-\alpha) \cdot \text{Linear}_{d_{\max} \to d_{M_{(k)}}}(x) + \alpha \cdot \text{Trunc}(x; d_{M_{(k)}})$$

$$\text{Proj-out}^{(\ell)}_{(k)}(x; \alpha) := (1-\alpha) \cdot \text{Linear}_{d_{M_{(k)}} \to d_{\max}}(x) + \alpha \cdot \text{Pad}(x; d_{\max}).$$

Respectively, $\text{Trunc}(\cdot; d)$ and $\text{Pad}(\cdot; d)$ truncate and zero-pad input to dimension d, and $\text{Linear}_{d \to d'} : \mathbb{R}^d \to \mathbb{R}^{d'}$ is a learnable linear layer with gating weights $\alpha \in [0, 1]$. In total, where $\alpha \in \Delta^{K-1}$ and I_k is a length-n_k vector of block indices from component model k, we define the output of the block sequence defined by I_k as

$$h_k(x; \alpha_k, I_k) = \left(\text{Proj-out}^{(I_{k,n_k})}_{(k)} \circ M^{(I_{k,n_k})}_{(k)\text{Block}} \circ \ldots \circ M^{(I_{k,1})}_{(k)\text{Block}} \circ \text{Proj-in}^{(I_{k,1})}_{(k)}\right)(x; \alpha_k).$$

Mixture Weights. Next, we would like to mix the activations of different component models' block sequences, in a way that allows us to learn how much influence the blocks from each component model will have on the overall hybrid model. Learning the amount of influence that each block sequence should have on the overall hybrid is critical—if certain blocks produce less helpful features, we need a way to down-weight them. Conversely, we want to use the best blocks in our hybrid as much as possible—we want to up-weight helpful blocks. Overall, a parameterization that allows us to learn these weights should lead to better hybrids. We do this by taking a convex combination of the projectors' outputs: given the projected features $h_k(x; \alpha_k, I_k)$ for each component model $k \in [K]$, we output a

convex combination of projected features

$$\text{Mix}_\alpha(x; I_1, ..., I_K) = \sum_{k \in [K]} \alpha_k h_k(x; \alpha_k, I_k). \tag{4.1}$$

We reuse the convex combination weights as the gating weights in the projectors. This choice yields the convenient property that when the mixture weights α are set to one in index k and zero everywhere else, the Mix function exactly computes a sequence of blocks from component model k while completely ignoring the projectors and the blocks from other component models. We adopt a popular parameterization for mixture weights from the NAS literature [Liu et al., 2019c]: we parameterize α as a softmax of a parameter vector—that is, $\alpha_k := \frac{\exp(a_k)}{\sum_{j \in [K]} \exp(a_j)}$ for all $k \in [K]$.

Manticore. We are now ready to define our overall hybrid architecture. We seek to create a hybrid from K component models, $M_{(1)}, ..., M_{(K)}$, each with a potentially different number of blocks, denoted $L_{M_{(k)}}$ for component model k. We fix L to be the number of *Manticore blocks*, where L is a common factor of each of the depths $L_{M_{(k)}}$, for all $k \in [K]$—we treat this choice of factor as a hyperparameter. For each of the L Manticore blocks, we want to mix a sequence of blocks from each of the K component models. We also want the number of blocks from each model $k \in [K]$ that are allocated to a single Manticore block to be evenly spread throughout the L Manticore blocks—this is why we require L to be a factor of $L_{M_{(k)}}$. For each component model $k \in [K]$, divide the indices of the blocks $[L_{M_{(k)}}]$ evenly into L contiguous parts, denoted as $[L_{M_{(k)}}] = (I_{k,1}, ..., I_{k,L})$. Then, adopting the notation from our component models, a Manticore block is defined as

$$\text{Manticore}^{(\ell)}_{\text{Block}}(\cdot) := \text{Mix}_{\alpha^{(\ell)}}(\cdot; I_{1,\ell}, ..., I_{K,\ell})$$

with $\text{Manticore}^{(\ell)}_{\text{Block}} : \mathbb{R}^{t \times d_{\max}} \to \mathbb{R}^{t \times d_{\max}}$, for each $\ell \in [L]$, and $\alpha^{(\ell)}$ being the mixture weights at ℓ. Next, we initialize a new set of embedding weights and a new task specific (or language modeling) head, and we can finally illustrate a forward pass with a Manticore hybrid model, denoted using the shorthand notation $\text{Manticore}(\cdot) := \text{Manticore}[M_{(1)}, ..., M_{(K)}](\cdot)$. Let $x = (x_1, ..., x_t) \in \mathcal{V}^t$ be a sequence of t tokens from a set $\mathcal{V}$. The forward pass is computed as follows:

1. Apply the new embedding function $\text{Manticore}_{\text{embed}} : \mathcal{V}^t \to \mathbb{R}^{t \times d_{\max}}$ to the tokens, result-

ing in a sequence of embeddings denoted $x_{\text{embed}} = \text{Manticore}_{\text{embed}}(x)$.

2. Take forward passes through L Manticore blocks, each with dimension d_{max}, concretely, we compute $x_{\ell+1} := \text{Manticore}^{(\ell)}_{\text{Block}}(x_\ell)$, where $x_1 := x_{\text{embed}}$.

3. Pass x_{L_M+1} into a new task-specific or language modeling head, $\text{Manticore}_{\text{head}} : \mathbb{R}^{t \times d_M} \to \mathbb{T}$, where $\mathbb{T}$ is the appropriate output space for the learning task.

In NAS terms, our search space is over the set of $L \ni \ell$ mixture weights $\alpha^{(\ell)} \in \Delta^{K-1}$. However, *our search space differs from typical gradient-based NAS techniques* in the sense that we do not require *discretization* to derive a final architecture after we obtain our mixture weights. Typically, NAS would involve selecting a single sequence of component architecture blocks at each of the Manticore blocks, usually by taking the arg max of the mixture weights. Instead, the mixtures themselves are what characterize Manticore hybrids. Nonetheless, if we were to replace the mixture weights $\alpha^{(\ell)}$ with discrete one-hot vectors, we could derive any of the following: the component model architectures themselves, existing hybrid architectures, and 'frankenmerged' models [Goddard et al., 2024].

4.1.2 How To Use Manticore

With Manticore, we can automatically select language models without training every model in the search space, automatically construct pretrained hybrid architectures without significant trial-and-error, and program pretrained hybrids without full training. In this section, we discuss the details of how Manticore can be used in each of these three usage scenarios.

Training hybrids from scratch. *Manticore can be used to automatically select LMs without training all of the LMs in the search space.* Our selection technique is simple: inspired by gradient-based NAS techniques [Liu et al., 2019c] and treating the mixture weights as our 'architecture parameters,' we proceed in two steps: 1. train mixture weights along with all other parameters, and 2. freeze the mixture weights and retrain the rest of the parameters from scratch. Unlike NAS, we found that in many pretraining settings, it was sufficient to stop at 1. and forgo retraining. In our pretraining experiments, we use randomly-initialized GPT-Neo [Black et al., 2021] and Mamba [Gu and Dao, 2023] as component models without projectors, and separately experiment with a subset of blocks from MAD [Poli et al., 2024].

Fine-tuning pretrained hybrids. *Manticore can be used to create and fine-tune pretrained hybrids.* We create pretrained hybrids as follows: begin with a set of pretrained models, replace their LM heads and embeddings with a single randomly initialized LM head and embedding layer, and pretrain the projectors on a small amount of general language data such as FineWeb [Penedo et al., 2024b] while keeping the original component model weights frozen.[2] To fine-tune the pretrained hybrids on downstream task data, we first search for mixture weights by training all of the parameters simultaneously, we freeze the mixture weights, rewind the component models and projectors to their pretrained state, and fine-tune. This procedure completely sidesteps large-scale pretraining of new hybrids.[3] In our synthetic experiments, we create pretrained Manticore hybrids from pretrained GPT-Neo-125M [Black et al., 2021] and Mamba-130M [Gu and Dao, 2023] models, while for our experiments on real natural language data, we opt for pretrained Pythia-410M [Biderman et al., 2023] and Mamba-370M [Gu and Dao, 2023] as component models.

Programming hybrids. Excitingly, there are cases in which we can program Manticore mixture weights by using external information to predict them. We consider two scenarios. If we know that a component model has blocks that are incompatible with the target task—e.g. resulting from sequence length constraints—we can omit these blocks by setting their mixture weights to 0. Otherwise, we can predict good mixture weights by searching on a fixed set of proxy tasks. For this, we use MAD tasks [Poli et al., 2024]. The MAD tasks are synthetic unit tests that are predictive of hybrid LM scaling laws, but within our framework, we find that MAD can also be useful for finding pretrained hybrids. We use the following procedure for programming mixture weights using the MAD tasks. First, run search on the MAD tasks using a smaller, randomly initialized version of our pretrained hybrid. For each MAD task, our search procedure returns a set of mixture weights—we simply average the resulting mixture weights, freeze them, and fine-tune on downstream task data.

4.1.3 Discussion and Design Considerations

Manticore features several intentional design decisions that we make concrete in this section.

[2] We found that 100M tokens sufficed for projector pretraining.

[3] We include an extensive FLOPs analysis and a discussion of comparable baselines in the Appendix.

Where Manticore hybrids excel. It is known that hybrids excel at compositional tasks like finding a token arbitrarily far in the past and then performing a local copy operation—this for instance, necessitates a tradeoff in SSM [Gu et al., 2022] state size and transformer context. Results like these motivate the study of tools like Manticore. For this reason, we expect that Manticore excels at tasks in which the component models are specialized for certain data sources or aspects of the dataset. As a result, many of our experiments in Section 4.2 feature heterogeneous data sources.

Design tradeoffs in Manticore. Manticore requires taking a forward pass with each of its component models, which increases inference cost over the use of a single component model. This increased inference cost is an explicit tradeoff for not having to pretrain a hybrid from scratch. In Appendix 4.E, we motivate this tradeoff by showing that the total FLOPs required to produce a Manticore hybrid is dominated by component model pretraining, and that this can be avoided by reusing existing pretrained models. Due to the simplicity of our projector architecture, we also show that the inference cost of Manticore is dominated by forward passes of its component models. This further motivates our comparison to ensembles in Section 4.2, due to their similar inference and training FLOPs requirements. Finally, Manticore can be scaled to larger component models without significant overhead, as the inference costs scale linearly in the size of its component models.

Flexibility of search algorithm. Our search space works best with NAS algorithms that support continuous-valued mixture weights, such as DARTS [Liu et al., 2019c], GAEA [Li et al., 2021b], and other gradient-based NAS algorithms. This makes our framework particularly flexible in its support for this broad class of NAS algorithms, while leaving room for specialized algorithms to be developed later. In Appendix 4.B, we include an ablation comparing DARTS to the DASH [Shen et al., 2022] search algorithm, along with various other components of the NAS pipeline. These ablations help characterize the desirable traits of NAS search algorithms for Manticore. For the purposes of our experiments, we mainly rely on DARTS [Liu et al., 2019c]—an entirely off-the-shelf NAS algorithm—and leave the development of tailor-made hybrid search algorithms to future work.

4.2 Experimental Results

We provide experimental evidence that validates the following claims about Manticore:

- C1. Pretrained hybrids can outperform their component models on fine-tuning tasks,
- C2. Trained from scratch, Manticore is competitive with existing hybrids and LMs, and
- C3. In certain cases, we can program mixture weights without search on the task data.

4.2.1 Fine-Tuning Pretrained Hybrids

We evaluate C1, first on a synthetic task, and then on natural language fine-tuning tasks.

Setup. We consider a synthetic LM dataset comprising GPT-Neo and Mamba generated completions of text from Penn Treebank [Marcus et al., 1993b]. Naturally, we also use pretrained GPT-Neo-125M and Mamba-130M models as component models, creating a single Manticore block with projectors that were pretrained on 100M tokens from FineWeb [Penedo et al., 2024b]. We search using DARTS, and afterward, we rewind the model weights and projectors to their pretrained states for retraining.

Results. Our results are shown in Figure 4.2 (left). We compare our search results to a sweep over a range of possible mixture weights and find that our search procedure returns the optimal mixture weights, outperforming both Mamba and GPT-Neo. This confirms our claim that Manticore hybrids can outperform their component models on *synthetic* fine-tuning tasks. Given that this task comprises two slices that each of our component models should be good at—GPT-Neo should be good at predicting GPT-Neo outputs, and vice versa—we hypothesize that Manticore hybrids are especially well suited to the component models having complementary 'skills' [Chen et al., 2023b].

Setup. We evaluate on three natural language fine-tuning datasets: Penn Treebank [Marcus et al., 1993b], the Alpaca instructions dataset [Taori et al., 2023], and ELI5 [Fan et al., 2019]. We use Pythia-410M and Mamba-370M as our component models, and create a single Manticore block from the blocks of the two models with projectors that were pretrained on 100M tokens from FineWeb [Penedo et al., 2024b]. As before, we first search for mixture weights, and then we retrain with the fixed mixture weights found by search.

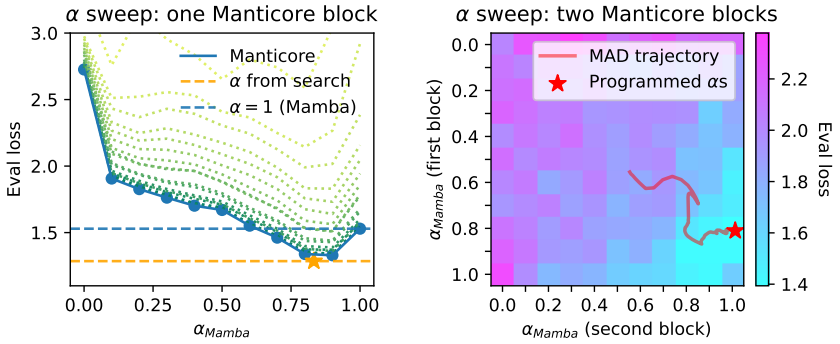


Figure 4.2: Mixture weight sweeps on Penn Treebank completions using pretrained GPT-Neo-125M and Mamba-130M as our component models. (Left) When we create one Manticore block, there is a region of the search space where we improve over Mamba. Here, we denote the loss value and mixture weights found via search using a yellow star and track the loss throughout training in green. (Right) The same holds for two Manticore blocks, and our technique for hybrid programming using MAD discovers this region.

Results. Our results are shown in Table 4.1. Manticore outperforms its component models on Alpaca and ELI5, while it achieves performance between its two component models on Penn Treebank. This confirms our claim that Manticore can outperform component models on *real* natural language tasks. The fact that Mamba-370M outperforms Manticore in this setting is not a failure of our framework, as Mamba-370M is included as part of our search space—improving the search procedure beyond off-the-shelf NAS algorithms in order to obtain these high performing models is an interesting direction for future work.

Task	Pythia-410M (A)	Mamba-370M (B)	Manticore[A, B]
PTB	0.9099	**0.8397**	<u>0.8600</u>
Alpaca	2.5011	<u>2.2999</u>	**2.1779**
ELI5	4.1260	<u>3.9414</u>	**3.9331**

Table 4.1: Manticore on language tasks using Pythia-410m and Mamba-370m component models. The best test losses are bolded and the second-best are underlined.

Setup. Building on the previous setup for natural language tasks, we perform a sweep over the α parameter corresponding to Mamba in our search space, and compare the results of the sweep to off-the-shelf NAS algorithms: DARTS [Liu et al., 2019c] (Manticore's search

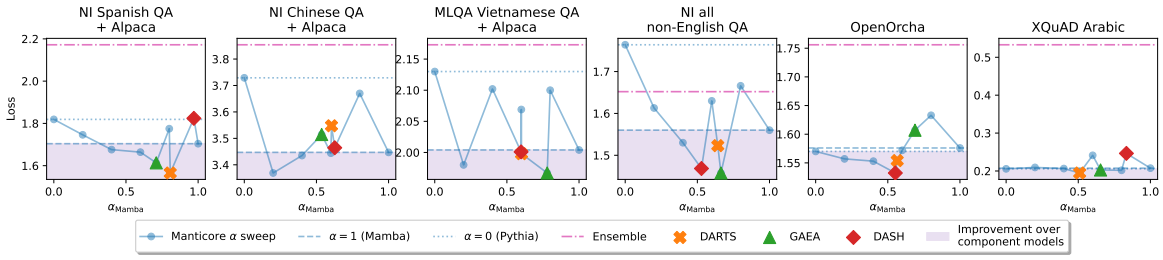


Figure 4.3: Mixture weight sweeps using Pythia-410M and Mamba-370M component models. NAS algorithms often locate regions of the search space that outperform component models and a learned ensemble baseline.

procedure), GAEA [Li et al., 2021b], and DASH [Shen et al., 2022]. In order to compare Manticore to a method with comparable inference cost, we also consider an ensemble baseline where the ensemble weights are learned during training. For three datasets, 50% of the documents are drawn from the Alpaca [Taori et al., 2023] dataset to artificially induce heterogeneity–we hypothesize that Manticore hybrids are well-suited to such settings—if Manticore's component models specialize in different subsets of a dataset, then Manticore should achieve improved overall performance on the combined dataset.

Results. Our results are shown in Figure 4.3. We find that in all but one setting (NI Chinese QA + Alpaca), at least two of the NAS algorithms that we evaluate recover a model that outperforms its component models. Furthermore, on five of the datasets, at least one NAS algorithm outperforms or matches the best model found during the sweep. Manticore also substantially outperforms the ensemble on all tasks. This is further evidence for our claim that Manticore outperforms component models on natural language, and demonstrates that NAS algorithms can find performant pretrained hybrids in our search space.

4.2.2 Training Hybrids from Scratch

For C2, we compare to prior hybrids on MAD and non-hybrid models on LRA and MAD.

Setup. We compare training Manticore from scratch to training existing hybrid architectures on MAD tasks. We begin with two hybrid architectures from the literature: Mambaformer [Park et al., 2024], which combines Mamba and attention blocks, and the striped multi-head Hyena + Mixture-of-Experts (MoE) MLP architecture that was shown to perform

well on the MAD tasks [Poli et al., 2024]. We compare these two baselines to a Manticore hybrid combining three component models: striped multi-head Hyena + MoE-MLP, a transformer, and Mamba. We use two blocks for each of these architectures, creating two Manticore blocks. Again, we search for mixture weights and then retrain.

Results. The results of this experiment are shown in Table 4.2 (left). We outperform the striped multi-head Hyena + MoE model from Poli et al. [2024], and we approach the performance of Mambaformer on all but one task. This validates the claim that Manticore hybrids, trained from scratch, compete with existing hybrids. Despite Mambaformer not being a component model, it is in our search space, and we again speculate that improvements in search would lead to its recovery.

	Starting from existing hybrids			Starting from non-hybrids		
Task	**SMH Hyena + MoE-MLP (A)**	**Mamba-former (B)**	**Manticore**	**GPT-Neo (C)**	**Mamba (D)**	**Manticore [C, D]**
Ctx. Recall	3.7153	**0.0020**	<u>0.0048</u>	<u>4.0771</u>	4.1858	**4.0768**
Fuzzy Recall	**4.1714**	**4.1712**	<u>4.1750</u>	<u>4.4384</u>	4.8097	**4.2797**
Noisy Recall	<u>4.1643</u>	4.1646	**4.1607**	<u>4.1843</u>	4.2605	**4.1823**
Sel. Copy	1.8021	**0.0005**	<u>0.0171</u>	<u>1.0470</u>	3.7765	**0.9478**
Mem.	<u>8.8353</u>	**5.2179**	8.9254	<u>4.6110</u>	5.2281	**4.1367**

Table 4.2: Results for training from scratch on MAD tasks. (Left) Manticore matches the performance of existing hybrids on all but one task. (Right) Manticore improves over non-hybrid component models. (Both) best losses are bolded and second best are underlined.

Setup. We compare Manticore hybrids to their component models on LRA, when trained from scratch. We use GPT-Neo and Mamba component models of similar sizes to those in Tay et al. [2021] to create Manticore hybrids, while keeping the number of blocks the same between the component models. In these experiments, we create a Manticore block for every block in the component models, ranging from 3 to 6 Manticore blocks.

Results. Our results are shown in Table 4.3. We outperform component models on all tasks except for IMDb. This validates the claim that Manticore hybrids, trained from scratch, compete with existing LMs.

Task	GPT-Neo (A)	Mamba (B)	Manticore[A, B]
ListOps	37.90	20.65	**38.70**
IMDb	59.62	**87.74**	72.44
CIFAR10	39.37	20.81	**43.15**
Pathfinder32	89.41	85.76	**91.45**
Pathfinder-X	N/A*	**75.50***	**75.50***

Table 4.3: Manticore trained from scratch on LRA using GPT-Neo and Mamba component models. Best accuracies are bolded. *GPT-Neo does not support the Pathfinder-X sequence length requirement, so its mixture weight is 0 and Manticore reduces to Mamba.

Setup. Next, we compare Manticore to non-hybrid architectures trained from scratch on the MAD tasks. For these experiments, our component models use the default architecture and training settings used in MAD. We compare two-block GPT-Neo and Mamba models to a Manticore hybrid using a single Manticore block.

Results. Our results are shown in Table 4.2 (right). Manticore outperforms GPT-Neo and Mamba on all of the MAD tasks in this setting. This provides further evidence for our claim that Manticore hybrids compete with existing LMs when trained from scratch. It is conceivable that our larger Manticore hybrids simply perform better than component models due to their size—however, we find that post-search discretization and retraining tends to result in similar performance, but reduces the model size by roughly half. We include an ablation of post-search discretization in the Appendix.

4.2.3 Programming Hybrids

We evaluate C3 with two types of external data: task metadata such as sequence length requirements, and the use of the MAD tasks as a proxy for search on downstream task data.

Setup. As in many of our previous experiments, we used the GPT-Neo and Mamba architectures as component models to our Manticore hybrid. However, this time, we set out to train from scratch on the extremely long-range Pathfinder-X task from LRA, which requires sequence length support greater than that of GPT-Neo. Using this external

information about the task, we set the mixture weights for GPT-Neo to 0, which in this case, means that Manticore reduces to Mamba.[4]

Results. The results of this experiment are shown in the last row of Table 4.3. In the simple case of having access to task metadata, this validates the claim that we can program mixture weights to exclude incompatible blocks. At the time of writing, we are not aware of prior published Mamba results on LRA despite community interest, which would make our evaluation in Table 4.3 the first such result. Note that we did not thoroughly tune hyperparameters, so we view this result as a preliminary starting point for the community to build off of, rather than a final answer.

Setup. Finally, in the case in which we can actually run all of our component models on our learning task, we explore when we can program the mixture weights using the MAD tasks as a proxy for search, which are intended to be predictive of scaling laws on The Pile [Poli et al., 2024, Gao et al., 2020a]. We set out to fine-tune a pretrained hybrid comprising GPT-Neo-125M and Mamba-130M, which were both pretrained on The Pile, with two Manticore blocks on our Penn Treebank completions synthetic. We train a scaled-down version of this Manticore hybrid with randomly initialized weights and two blocks per component model on the MAD tasks. This yields mixture weights for each of the MAD tasks—we average them across the tasks, and then fine-tune our pretrained hybrid on Penn Treebank completions using the predicted mixture weights.

Results. Our results are shown in Figure 4.2 (right). We superimpose the predicted mixture weights and mean search trajectory from MAD onto the architecture loss landscape computed on Penn Treebank completions. We find that this procedure recovers a hybrid that outperforms the component models (Mamba, lower right; GPT-Neo, upper left) and substantially outperforms the naive frankenmerges in our search space (upper right and lower left) [Goddard et al., 2024]. This is a scenario in which it is possible to program mixture weights using external sources without performing search on the task data. Intriguingly, search on the MAD tasks appears to follow the architecture gradient on the *different* downstream fine-tuning task, even though the architecture is scaled-down and trained

[4]Mamba on the LRA is open: `https://github.com/state-spaces/mamba/issues/282`.

from scratch on MAD. We hypothesize that programming Manticore hybrids becomes more difficult as the fine-tuning distribution is further from the pretraining distribution, and that the architecture loss landscapes become less similar. This evaluation was carried out on our synthetic PTB completions task, so the fine-tuning dataset should be fairly similar to the pretraining distribution. In our evaluation in Table 4.1, we find that Mamba outperforms the Pythia component model on English natural language tasks that are further from the pretraining distribution than our synthetic (while both models were trained on The Pile [Gao et al., 2020a] which is largely in English, we are not training on completions produced by the models themselves). Finally, our evaluations in Figure 4.3 use non-English text, which is further from the pretraining data distribution, and we observe no discernible pattern between their loss landscapes—programming α parameters in this scenario is likely challenging.

4.3 Conclusion

We present Manticore, a framework that automates the creation of hybrid models from pretrained models by using projectors and convex combinations to align and combine features from multiple different component models, as well as NAS-inspired search procedures. Manticore is efficient and flexible in its usage; hybrids can be trained from scratch, fine-tuned from pretrained component models, and even programmed with external information and/or proxy tasks. In our experiments with several real/synthetic language modeling datasets and existing component/hybrid models, we find that Manticore hybrids match or outperform existing handcrafted hybrid models in these settings. Any requisite fine-tuning and evaluation is performed with a single, large Manticore model rather than designing new hybrids by hand and pretraining them from scratch, which dramatically reduces the computational cost of designing hybrids.

4.A Related work

Language Model Architectures: Transformers and Beyond. Transformers are currently the dominant LM architecture. The success of the "vanilla" architecture introduced by Vaswani et. al. [Vaswani et al., 2017b] has led to many proposed variations. The quadratic

complexity of the base self-attention operation has inspired the search for alternative architectures that offer comparable performance with subquadratic complexity. One line of work builds off *state-space models*, with variations made to enable language modeling [Poli et al., 2023a,b, Gu and Dao, 2023, Arora et al., 2024]. Another line of work involves linear-complexity attention by formulating transformers as RNNs and expressing self-attention as a kernel dot-product [Katharopoulos et al., 2020]. Other approaches increase the expressivity of this formulation with data-dependent gating [Yang et al., 2024]. Our work does not propose a new architecture. Instead, we focus on the idea *that practitioners should be able to take advantage of new architectures in a transparent way.*

Neural Architecture Search and Mechanistic Search. Neural architecture search (NAS) techniques are used to automatically search for optimal architectures. These techniques have produced state-of-the-art models in several different architectures and data domains. Much of the challenge in NAS is the complexity of the search procedures; in the most standard form, NAS involves a difficult bilevel optimization over a large search space. Much effort has been aimed at reducing these costs, often via continuous relaxations of the large search spaces, with efficient, end-to-end differentiable search techniques like DARTS [Liu et al., 2019c], GAEA [Li et al., 2021b], and DASH [Shen et al., 2022].

Using NAS to discover architectures for language modeling—and especially those that may rival Transformers—has thus far been hard. A promising approach is the MAD framework [Poli et al., 2024] , which uses "*mechanistic* tasks" (synthetic tasks organized around simple principles) to search for high-quality subquadratic architectures. While we do not seek to discover *new* architectures, we are inspired by this approach in our effort to search for *hybrid* architectures.

Hybrid Architectures. Perhaps unsurprisingly, there is no single dominant architecture among either standards, like Transformers, or emerging subquadratic architectures. While there are some insights that can be converted into heuristics for model selection, generally, to take advantage of new models, practitioners must exhaustively evaluate all of them on each of their tasks. The cost of doing so has inspired the idea of crafting hybrid architectures that mix components from different approaches, with the goal being to obtain best-of-all-worlds behavior.

Unfortunately, the space of hybrid architectures is already large and only grows with each new proposed approach. Manually crafting hybrids is costly; users must either brute-force the enormous search space or alternatively hand-craft a small candidate set of hybrids in the hope that it includes a reasonably performant choice. Our work provides an efficient alternative to this process.

Model Merging. A final prospective approach to using multiple models is *merging*. Merging pretrained models (of the same architecture) has shown promising results [Yadav et al., 2023, Yu et al., 2023, Wortsman et al., 2022, Ilharco et al., 2023, Davari and Belilovsky, 2023, Jang et al., 2024], creating powerful large-scale merges such as SOLAR-10.7B [Kim et al., 2023a] and Goliath-120B[5] from two fine-tuned Llama2-70B [Touvron et al., 2023b] models. The former two were produced using a trial-and-error-based technique called 'franken-merging,' introduced in MergeKit [Goddard et al., 2024]. Frankenmerging involves stitching together different fine-tuned versions of the same model or, hypothetically, different models. This has inspired efforts to merge models of different architectures using large-scale evolutionary search [Akiba et al., 2024]. However, such efforts are still embryonic, with substantial computational drawbacks, requiring many training runs. *Manticore, on the other hand, does not require training a large number of models.*

4.B Ablations

Choice of search algorithm. By default, we use a form of the single-level DARTS [Liu et al., 2019c] search algorithm in all of our experiments requiring search. We optionally evaluate whether or not to take *alternating* update, that is, we alternately take gradient steps in the architecture and model parameters—we treat this choice as a task-dependent hyperparameter. However, there are many alternative NAS algorithms that we could have used for search. In our ablation of the choice of search algorithm, we also evaluate DASH [Shen et al., 2022] on our Penn Treebank completions synthetic—the results of which are shown in Table 4.4. In general, we found that using DASH was unable to recover strong architectures in our search space. We postulate that this is because DASH simply aims to solve a different problem, and is not suited to our search space: namely, DASH is used to

[5] `https://huggingface.co/alpindale/goliath-120b`

search for lower-level operations, rather than LM blocks. We also found that alternating DARTS updates was somewhat helpful, compared to simultaneously updating all of the parameters at once—for our experiments, we treated this choice as a hyperparameter.

Alternating?	DARTS	DASH
Yes	1.2854	2.5899
No	1.3635	2.5968

Table 4.4: Comparison of NAS search methods on our Penn Treebank completions synthetic.

Whether or not to discretize after search. We perform an ablation of whether or not to perform discretization on our MAD task experiments in which we compare to existing hybrids. We find that while discretization can sometimes improve performance, the performance differences are often marginal. If final parameter count is a concern, then discretization is beneficial.

Task	**Manticore (non-discretized)**	**Manticore (discretized)**
Context Recall	**0.0068**	0.0081
Fuzzy Recall	4.1764	**4.1729**
Noisy Recall	4.1628	**4.1614**
Selective Copying	0.0849	**0.0006**
Memorization	8.9416	**8.9402**

Table 4.5: A comparison of non-discretized vs. discretized Manticore.

Amount of projector pretraining. Finally, we ablate over the amount of projector pretraining. We re-ran our α sweep on our PTB completions synthetic with different amounts of projector pretraining, ranging from 0 to 100M tokens sampled from FineWeb [Penedo et al., 2024b]. The results of this ablation are shown in Figure 4.4. We found that the optimal value of the α parameter stabilizes around 70M tokens used to pretrain the projectors.

4.C Additional MAD results

In the main text, we presented results comparing Manticore hybrids trained from scratch to existing hybrids from the literature—namely Mambaformer and the Striped MH Hyena +

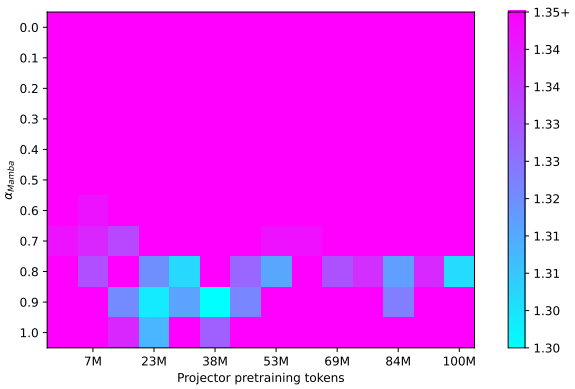


Figure 4.4: As evaluated on our PTB completions synthetic with Mamba-130M and GPT-Neo-125M, we find that the optimum stabilizes at around 70M tokens of projector pretraining.

MOE architecture from Poli et al. [2024]. Notably, the Striped MH Hyena + MOE architecture was only the second best architecture presented by Poli et al. [2024]. We found that their best architecture, the Striped Hyena Experts + MOE model, performed slightly worse on the harder versions of the MAD tasks that we evaluated. We present these results in Table 4.6.

Task	Striped Hyena Experts + MoE-MLP	Striped MH Hyena + MoE-MLP	Mambaformer	Manticore
In-context Recall	4.0315	3.7153	**0.0020**	<u>0.0048</u>
Fuzzy In-context Recall	<u>4.1749</u>	**4.1714**	**4.1712**	<u>4.1750</u>
Noisy In-context Recall	<u>4.1640</u>	4.1643	4.1646	**4.1607**
Selective Copying	2.1731	1.8021	**0.0005**	<u>0.0171</u>
Memorization	8.8537	<u>8.8353</u>	**5.2179**	8.9254

Table 4.6: Trained from scratch on MAD tasks, Manticore beats or matches the performance of existing hybrids on all but one task. The best test losses are bolded and the second best are underlined.

4.D Additional Pathfinder Results

We ran several additional variants of the pathfinder task for which the required sequence length exceeded the maximum supported sequence length of GPT-Neo. We report these results in Table 4.7.

Pathfinder task	GPT-Neo (A)	Mamba (B)	Manticore [A, B]
64×64, 6 paddles	N/A	80.40	80.40
64×64, 9 paddles	N/A	90.01	90.01
64×64, 14 paddles	N/A	86.87	86.87
128×128, 6 paddles	N/A	75.50	75.50

Table 4.7: Additional Pathfinder results. Note that since these variants of Pathfinder exceed the maximum sequence length of GPT-Neo, we set its mixture weight to be 0 and evaluate using Mamba.

4.E On Baselines

The correct set of baselines for Manticore is an interesting and somewhat challenging question. In the main text, we compare to the set of component models used to construct a Manticore hybrid—in other words, in order for Manticore to be at least as performant as its component models on a task, it must match or beat the performance of the best component model, which implies that both component models need to be fine-tuned. This would roughly match the total amount of fine-tuning FLOPs used to train the corresponding Manticore hybrid. However, there are other potential ways to make a comparison; in this section, we will discuss the fairness and availability of baselines corresponding to different metrics of comparison, and provide a new set of baselines involving ensembles of component models. Specifically, we will address the question of whether the correct comparison is one involving parameter count, training FLOPs, or inference FLOPs.

4.E.1 Parameter Count

One proposal is to compare a Manticore hybrid of size N to a pretrained model that is also of size N. Manticore combines the weights of existing *pretrained* models to produce a hybrid that is drastically cheaper to generate compared to pretraining a hybrid of the same size from scratch. Off-the-shelf pretrained models of size N are often pretrained up to D tokens corresponding to its Chinchilla optimum [Hoffmann et al., 2022b], but information about the amount, mixture, or quality of pretraining data is often unavailable. This makes comparison along the axis of the parameter count alone somewhat challenging—a larger model may well have been trained on more total data than the two smaller component models making up

Manticore. In other words, Manticore should not be expected to follow the same pretraining scaling laws as models that were trained from scratch. Therefore, comparing a Manticore hybrid and a pretrained model of the same size is not necessarily a fair comparison, when considering model size alone. Furthermore, pretrained models of a specific predefined size N are not even guaranteed to exist.

4.E.2 Training FLOPs

Another option is to make a comparison along the axis of total training FLOPs, which would include pretraining FLOPs, fine-tuning FLOPs, and any additional FLOPs incurred when generating a Manticore hybrid. Suppose we create a Manticore hybrid from two component models of sizes N_1 and N_2, which have been pretrained using T_1 and T_2 tokens, incurring roughly $6N_1T_1$ and $6N_2T_2$ FLOPs, respectively [Kaplan et al., 2020b]. With Manticore, we incur FLOPs from two sources: projector pretraining and fine-tuning. In our experiments, we use $T_{\text{proj}} = 100\text{M}$ tokens of general data for projector pretraining, and saw in Figure 4.4 that we likely didn't even need this much. Nonetheless, 100M tokens is substantially smaller than the typical amount of pretraining data, so we can assume that $T_{\text{proj}} = 100\text{M} << \min\{T_1, T_2\}$, and since the pretrained projectors can be reused, this cost can be amortized over many future fine-tuning runs. Manticore then involves fine-tuning on some small amount of downstream tasks-specific data comprising $T_{\text{ft}} << \min\{T_1, T_2\}$ tokens. So then, the total amount of training FLOPs involved end-to-end in producing a Manticore hybrid is

$$6N_1T_1 + 6N_2T_2 + (6N_1 + 6N_2)T_{\text{proj}} + (6N_1 + 6N_2)T_{\text{ft}} = O(6N_1T_1 + 6N_2T_2),$$

meaning that the total training FLOPs is dominated by the pretraining of the component models. Our experiments in the main text compare Manticore to the better of the two component models, which means that both component models need to be fine-tuned (i.e., the baseline comprises 'both' component models). Therefore, if the projector pretraining FLOPs are amortized over many fine-tuning runs, Manticore roughly matches the baseline in terms of training FLOPs. That is, this baseline and Manticore effectively requires $6N_1T_1 + 6N_2T_2 + (6N_1 + 6N_2)T_{\text{ft}}$ FLOPs.

4.E.3 Inference FLOPs

It is true that our baselines in the main text (which are pairs of component models) are cheaper in terms of inference FLOPs compared to Manticore. In fact, Manticore effectively doubles the inference FLOPs by requiring forward passes through both component models. Here, we include an analysis of inference FLOPs showing that the contribution of the projectors is negligible, and we present an additional baseline—combining the component models into an ensemble that is fine-tuned simultaneously using the same fine-tuning budget as Manticore.

Inference FLOPs analysis. First, we will compute the general form of the inference FLOPs requirement for a component model. Let d be the embedding dimension, let t be the sequence length, let L be the number of blocks, let $v = |\mathcal{V}|$ be the size of the vocabulary set for our downstream task, and let $B(d,t)$ be the inference FLOPs requirement for the blocks in the component model. Then the inference requirement for a single token prediction from the component model is computed by summing the FLOPs requirements from looking up an embedding, computing forward passes through a sequence of blocks, and generating the final logits. That is, we obtain the following:

$$O(1 + LB(d,t) + dv) = O(LB(d,t) + dv).$$

For a Manticore hybrid, assume that we have $K = 2$ component models, M_1 and M_2, as well as their projectors. Without loss of generality, assume that the embedding dimensions, d, and the number of blocks, L_M, in the component models are the same. Let $L << L_M$ be the number of *Manticore* blocks, which is typically constant with respect to the number of blocks in each of the component models L_M (in our experiments, L was set to 1 or 2). Let $B_{M_1}(d,t)$ and $B_{M_2}(d,t)$ be the FLOPs requirements of individual blocks from M_1 and M_2 respectively, and let $B_{\text{proj}}(d,t) = O(td^2)$ be the FLOPs requirement of projector usage. Note that typically, $B_{\text{proj}}(d,t) = O(td^2) \leqslant B_{M_*}(d,t)$, as many types of blocks involve a dimension-mixing operation such as an MLP, which has a larger FLOPs requirement than $O(td^2)$, or a sequence mixer that has quadratic or log-linear dependence on t, rather than the linear dependence of $B_{\text{proj}}(d)$. Then the FLOPs requirement of each Manticore block is

as follows:

$$O\left(\frac{L_M}{L}(B_{M_1}(d,t)+B_{M_2}(d,t))+4B_{\text{proj}}(d,t)\right),$$

and along with the token embedding and the logits output, we have

$$\begin{aligned}
&O(1)+L*O\left(\frac{L_M}{L}(B_{M_1}(d,t)+B_{M_2}(d,t))+4tB_{\text{proj}}(d,t)\right)+O(dv)\\
=&O\left(L_M B_{M_1}(d,t)+L_M B_{M_2}(d,t)+LB_{\text{proj}}(d,t)+dv\right)\\
=&O\left(L_M B_{M_1}(d,t)+L_M B_{M_2}(d,t)+td^2L+dv\right)\\
=&O\left(L_M B_{M_1}(d,t)+L_M B_{M_2}(d,t)+dv\right),
\end{aligned}$$

where the final step comes from $L << L_M$ and the assumption that $B_{\text{proj}}(d,t) = O(td^2) \leqslant B_{M_*}(d,t)$. This inference cost is the same as inference with both component models. This motivates another baseline: ensembles of component models, which we evaluate next.

Comparison to ensembles. We compare the fine-tuning performance of Manticore to ensembles of component models on the six tasks shown in Figure 4.3. Starting with pretrained Pythia-410M and Mamba-370M models, we construct our ensemble as follows: for each token prediction, we mix the output probabilities from Pythia-410M and Mamba-370M with equal weighting of 0.5, and then we fine-tune the entire mixture end-to-end on the downstream task. We present the results in Table 4.8. The ensemble baseline underperforms Manticore and the best component model on all tasks—we suspect that this could be related to overfitting.

4.F Hyperparameters

In this section, we discuss our hyperparameters and our experimental setup. Code implementing our experiments can be found at `https://anonymous.4open.science/r/manticore-anon`.

4.F.1 Fine-Tuning Pretrained Hybrids

Penn Treebank completions synthetic. For model weights, we use the AdamW [Loshchilov and Hutter, 2019] optimizer with a linear learning rate schedule with an initial learning rate

Task	Pythia-410M (A)	Mamba-370M (B)	Ensemble [A, B]	Manticore [A, B]
Es. + Alpaca	1.819	1.704	2.172	**1.664**
Ch. + Alpaca	3.729	3.447	3.854	**3.369**
Vi. + Alpaca	2.130	2.004	2.173	**1.980**
NI non-En.	1.764	1.560	1.652	**1.530**
OpenOrcha	1.570	1.576	1.756	**1.553**
XQuAD Ar.	0.205	0.207	0.533	**0.201**

Table 4.8: Comparison between Manticore, its component models, and an ensemble of its component models on the tasks from Figure 4.3. For Manticore, we show the best performance achieved across our sweep from Figure 4.3. Ensembling the component models does not improve performance, but creating a Manticore hybrid does lead to improved performance.

of $5e-5$. For mixture weights, we use the AdamW [Loshchilov and Hutter, 2019] optimizer with a linear learning rate schedule with an initial learning rate of 0.005 and use alternating updates.

Fine-tuning on language tasks. For model weights, we use the AdamW [Loshchilov and Hutter, 2019] optimizer with a linear learning rate schedule with an initial learning rate of $5e-5$. For mixture weights, we use the AdamW [Loshchilov and Hutter, 2019] optimizer with a linear learning rate schedule with an initial learning rate of 0.005 and use simultaneous updates.

4.F.2 Training Hybrids from Scratch

Comparison to existing hybrids on MAD.

We provide the hyperparameters and training details for our MAD evaluations from Section 4.2.2

Existing hybrids were trained with a hyperparameter grid search over the space $[1e-4, 5e-4, 1e-3]$ for learning rate and $[0.0, 0.1]$ for weight decay, similar to the procedure in MAD [Poli et al., 2024].

Manticore is trained in two stages. In the first stage, we train the model and architecture weights in the alternating schedule utilized in DARTS [Liu et al., 2019c]. In this stage, we perform a hyperparameter grid search of the space $[1e-4, 5e-4, 1e-3]$ for model weight

learning rate, $[1e-4, 1e-4]$ for architecture weight learning rate, and $[0.1]$ for weight decay. In the second stage, the architecture weights are frozen and we train only the model weights using the best learning rate found in the first stage.

Evaluation on LRA. We provide the hyperparameters and training details for our LRA evaluations.

- ListOps. We trained all models for 5000 steps. GPT-Neo used 8 attention heads, 6 blocks, an embedding dimension of 512, and a feed-forward network (FFN) dimension of 2048. Mamba used 12 blocks with a model dimension of 512. The vocabulary size was 18.
- IMDb. We trained all models for 25 epochs with a batch size of 32. GPT-Neo used 8 attention heads, 6 blocks, an embedding dimension of 512, and an FFN dimension of 2048. Mamba used 12 blocks with a model dimension of 512. The vocabulary size was 129.
- CIFAR10. We trained all models for 10 epochs. GPT-Neo used 4 attention heads, 3 blocks, an embedding dimension of 64, and an FFN dimension of 128. Mamba used 6 blocks with a model dimension of 64. The vocabulary size was 256, corresponding to the pixel value range of the grayscale image.
- Pathfinder32. We trained all models for 10 epochs. GPT-Neo used 8 attention heads, 4 blocks, an embedding dimension of 128, and an FFN dimension of 128. Mamba used 8 blocks with a model dimension of 128. The vocabulary size was 256, corresponding to the pixel value range of the grayscale image.

Comparison to non-hybrids on MAD.

We use two blocks each from GPT-Neo and Mamba, each with a model dimension of 128. We train for 200 epochs and select the best performance during training, as all of the models overfit across the board. We use the AdamW [Loshchilov and Hutter, 2019] optimizer with a linear learning rate schedule with an initial learning rate of $5e-5$.

4.F.3 Programming Hybrids

Mamba evaluation on long Pathfinder tasks. Due to our limited computation resources, we did not conduct a hyperparameter sweep for the result we presented. We used Mamba with models of a similar size as Pathfinder32, which has 8 layers, 128 as the hidden dimension size, and 256 as the vocab size. The 64×64, 6 paddles version is trained by 10 Epoch with

default HP. The result for other versions is trained with 200 epochs with default HP in Huggingface trainer.

MAD tasks as a search proxy. For model weights, we use the AdamW [Loshchilov and Hutter, 2019] optimizer with a linear learning rate schedule with an initial learning rate of $5e-5$. For mixture weights, we use the AdamW [Loshchilov and Hutter, 2019] optimizer with a linear learning rate schedule with an initial learning rate of 0.01 and use simultaneous updates. For search on the MAD tasks, we train scaled-down versions of GPT-Neo and Mamba each with four blocks, model dimensions of 128, and no projectors.

4.F.4 Pretraining Projectors

For all non-frozen weights (i.e., projectors, mixture weights, embeddings, and the LM head), we use the AdamW [Loshchilov and Hutter, 2019] optimizer with a linear learning rate schedule with an initial learning rate of $5e-5$.

4.G Data and MAD Task Parameters

We provide a more detailed description of the datasets that we use in our experiments. We perform our experiments on a range of synthetic and real tasks that measure various aspects of modern LM capabilities. We discuss the specific datasets that we use in our experiments below. MAD synthetics. The MAD synthetic datasets are a set of tasks introduced by Poli et al. [2024] to systematically evaluate the design space of LMs. These tasks are designed to serve as proxy unit tests for rapidly prototyping of new hybrid LM architectures. In our experiments, we use harder variants of the MAD tasks, in which we use a larger vocabulary size of 128 instead of the default 16 for most of the tasks, along with fewer training examples. For simplicity, we omit the compression task as it requires the use of encoder-decoder architectures.

- In-context recall. MAD utilizes a multi-query associative recall task, challenging models to retrieve values linked to keys within input sequences, testing their in-context learning ability across randomly shuffled mappings. We use a vocab size of 128 and 800 training examples.

- Fuzzy in-context recall. This is a variant of in-context recall to assess a model's ability to semantically group adjacent tokens. Variable-length keys and values are randomly paired, testing the model's capacity for fuzzy recall. We use a vocab size of 128 and 800 training examples.
- Noisy in-context recall. This is an adaptation of in-context recall to evaluate a model's capacity to disregard irrelevant information. This involves inserting tokens from a separate vocabulary randomly among key-value pairs, enhancing the memorization challenge. We use a vocab size of 128, a noise vocab size of 16 with 80% noise, and 800 training examples.
- Selective Copying. MAD employs a selective copying task to evaluate a model's ability to remember and replicate specific tokens from an input sequence while disregarding randomly inserted noise tokens, emphasizing the preservation of token order. We use a vocab size of 128 with 96 tokens to copy, and 800 training examples.
- Memorization. MAD assesses language models' factual knowledge retention through a memorization task, where models learn fixed key-value mappings without in-context computation, testing pure memorization ability. For this task, we use a vocab size of 8192.

Long Range Arena. Long Range Arena (LRA) [Tay et al., 2021] is a benchmark consisting of various tasks of different modalities that evaluate how well models can learn long-context data. For simplicity, we omit byte-level document retrieval as it requires two forward passes per example.

- Long ListOps. This task is designed to understand whether the architecture is able to model hierarchically structured data in a long-context [Nangia and Bowman, 2018].
- Byte-level text classification. This task attempts to test the model's ability to deal with compositionality as in the real world, the model needs to compose characters into words and words into higher-phrases in not so well defined boundaries making it a challenging task, we use IMDB dataset[Maas et al., 2011a] in the LRA paper [Tay et al., 2021].
- Image classification on a sequence of pixels. This task aims to understand whether a model is able to capture the 2D spatial structure when presented with a flattened 1D version of an image to classify, we use pixel information from CIFAR10[Krizhevsky, 2009] dataset.

- Pathfinder. This task helps to understand whether a model can reason about whether the given 2 dots in an image are connected by a path having dashes or not. The sequence length is 1024 i.e a 32x32 image is flattened and provided as input to the model [Linsley et al., 2018, Kim et al., 2020].

- Pathfinder-X. An extreme version of Pathfinder with a higher resolution, such as 64x64 and 128*128, which results in a sequence length of up to 16K

Penn Treebank completions. We generate a synthetic dataset of generated text from pretrained GPT-Neo-125M [Black et al., 2021] and pretrained Mamba-130M models [Gu and Dao, 2023]. We prompt both models using the first four words of every example in the Penn Treebank [Marcus et al., 1993b] validation set, which yields two natural slices of our dataset: sentence completions generated by GPT-Neo and those generated by Mamba.

Natural language tasks. We evaluate the ability to fine-tune Manticore on natural language datasets. Specifically, we evaluate on Penn Treebank [Marcus et al., 1993a], the Alpaca instruction tuning dataset [Taori et al., 2023], and an i.i.d. split of the ELI5 training set [Fan et al., 2019]. Additionally, we use 100M tokens from the FineWeb dataset [Penedo et al., 2024b] to pretrain our projector weights. We describe all other natural language datasets that we use in our evaluations below.

- NI Spanish QA + Alpaca. This is from the Natural Instruction dataset v2.8 downloaded from `https://github.com/allenai/natural-instructions/releases`, we picked task 1610 and mixed it with equal numbers of randomly selected samples from the Alpaca dataset to create a bilingual dataset that contains Spanish Q&A along with English instructions.

- NI Chinese QA + Alpaca. This is similar to the previous dataset, except we pick task1570, which is Q&A that input/output language are Chinese.

- MLQA Vietnamese + Alpaca. This dataset is a subset of MLQA (MultiLingual Question Answering)(`https://huggingface.co/datasets/facebook/mlqa`) in which both the inputs and outputs are in Vietnamese, and mixed with equal numbers of randomly selected samples from Alpaca dataset to create a bilingual dataset.

- OpenOrcha. We randomly sample 10,000 samples from the OpenOrcha dataset containing Japanese translations from `https://huggingface.co/datasets/atsushi3110/cross-lingual-openorcha-830k-en-ja`, to form a Japanese Q&A dataset.
- NI all non-English QA. There are six Q&A tasks in the Natural Instructions dataset such that both their input and output language is non-English—we combine all of them to form a new dataset containing non-English Q&A.
- XQuAD Arabic. The Arabic Q&A part from XQuAD (Cross-lingual Question Answering Dataset), from `https://huggingface.co/datasets/google/xquad`.

4.H A Call for Action & Community Recommendations

Throughout our research process, we noted a handful of opportunities that help to democratize LM research. Should these opportunities be taken up by the research community, we believe they could help to democratize and help to decentralize community-driven LM research, all which enabling further research on pretrained hybrids.

A search engine for pretrained models. Surprisingly, we were unable to easily search for pretrained LMs of certain sizes or with certain properties (using Huggingface or otherwise). Tools like this should exist: this would not only significantly democratize LMs, but it would help to reduce monopolies on LM releases and usage, and thereby decentralize LM research.

Standardized, block-structured LM implementations. We found that standard tools such as Huggingface and PyTorch were insufficient to cleanly access intermediate activations across several model implementations. This could be resolved by adopting standard implementations or structures for LMs that share the common block structure that we describe in Section 4.1.1. Instead, our solution was to fork implementations of several Huggingface models, which is time-consuming, error-prone, and non-scalable. A solution to this problem would enable and encourage further research on pretrained hybrid models, which in turn helps to democratize LM research.

Removing tokenizers from LM pipelines. We believe that there are too many possible tokenizers, and that tokenizers have a significant potential to introduce merge conflicts in model merging/pretrained hybrid pipelines. In response to this challenge, in our work, we simply chose an arbitrary tokenizer and relearned our embeddings and LM head from scratch in all of our experiments. Possible solutions to this problem would be: as a community, we agree on a standard (small) set of tokenizers, or we eliminate tokenizers altogether by learning character or byte-level LMs.

4.I Limitations

At various points in Section 4.2, we described limitations with using DARTS (the off the shelf NAS search algorithm that we used) for search, in that it was not always able to recover the best architecture in the search space. A potential limitation of Manticore is that it relies on the existence of good gradient-based NAS search algorithms, potentially tailored to our search space. However, we postulate that this is possible, and we leave the task of developing new search techniques to future work.

4.J Compute Resources

We ran our experiments on the following GPU hardware:

- 2x Nvidia RTX A6000 GPUs with 48GB GPU memory hosted locally in a nook in the lead author's house and in a friend's basement.
- 2x Nvidia RTX 4090 GPUs with 24GB GPU memory each hosted locally in other friends' basements.
- 2x Nvidia Tesla V100 GPUs with 16GB GPU memory each hosted on AWS (p3.2xlarge instances).

In total, we estimate that our total number of GPU hours across all experiments (those which failed as well as those included in this chapter) amounted to roughly 750 GPU hours. We estimate that less than half of these hours accounted for experiments that were not ultimately included in this chapter.

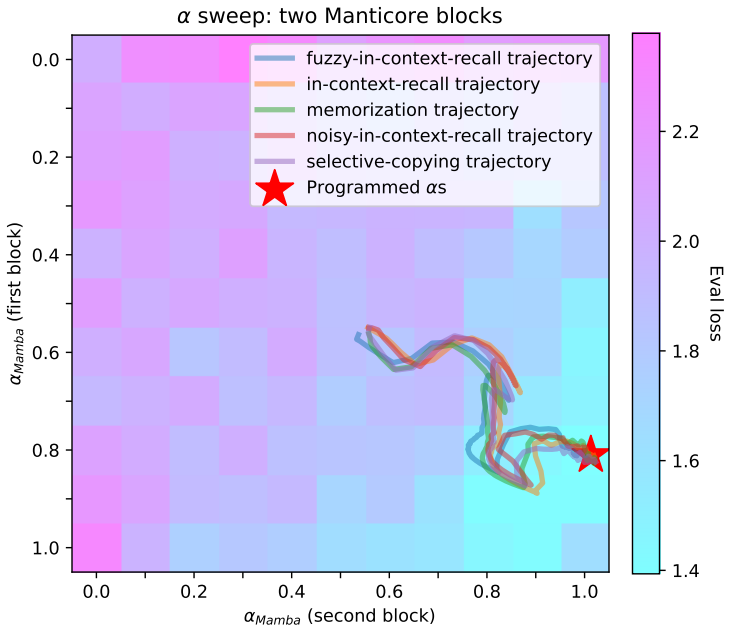


Figure 4.5: Mixture weight sweeps on Penn Treebank completions using pretrained GPT-Neo-125M and Mamba-130M as our component models. There is a region of the search space where we improve over Mamba when using two Manticore blocks, and our technique for hybrid programming using MAD discovers this region.

4.K Expanded Version of Figure 4.2 (Right)

To show how the architectures evolve over search on all of the MAD tasks in our mixture weights programming experiment, we provide a more detailed version of Figure 4.2 (Right) – this is shown in Figure 4.5. Here, we plot the architecture trajectories throughout training on all of the MAD tasks, and superimpose them onto the architecture-loss landscape of the Penn Treebank completions task. The trajectories roughly follow what appears to be a gradient in the loss landscape, and all of the trajectories are roughly similar. We derive our final 'programmed' alphas by taking the average of the final alpha values on each of the MAD tasks, after training.

Part II

Data Efficient Learning

Chapter 5

Skill-it! A Data-Driven Skills Framework for Understanding and Training Language Models

This chapter is based on joint work with Mayee F. Chen, Kush Bhatia, Jue Wang, Ce Zhang, Frederic Sala, and Christopher Ré, published at NeurIPS 2023 [Chen et al., 2023a]. The work was led by Mayee F. Chen.

Contributions: The author contributed to the design of the Skill-it algorithm including the skills graph component and in the running of final experiments for the continual pretraining evaluations.

Large language models (LMs) exhibit remarkable capabilities, including producing creative content [Stevenson et al., 2022], writing source code [Chen et al., 2021b], or chatting with users [Brown et al., 2020b]. A key ingredient in enabling models to perform such tasks is the data on which the models are trained [Google, 2023, Gururangan et al., 2020, Touvron et al., 2023a]. A natural way to unlock particular capabilities is to improve this training data. However, it is unclear how to select data from a large corpus for these capabilities given a fixed budget of training tokens, as data selection methods for current state-of-the-art LMs mostly rely on heuristics for filtering and mixing together different datasets [Lee et al., 2022, Touvron et al., 2023a]. We lack a formal framework for capturing how data influences the model's capabilities and how to utilize this data effectively for improving LM performance.

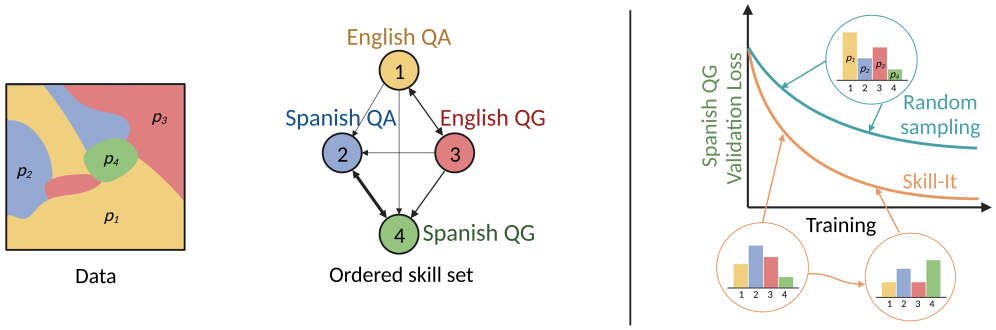


Figure 5.1: Inspired by how humans acquire knowledge, we hypothesize that LMs best learn skills in a particular order and that this can help improve our understanding and training of LMs. We show that these ordered skill sets exist in real data, which enables skills to be learned with less data given that we train on their prerequisite skills. We then propose Skill-it, an online data selection algorithm that learns skills quickly by exploiting their ordering.

To develop such a framework, we take inspiration from how humans acquire knowledge. A classic idea in education literature is the concept of *skills* that form a learning hierarchy [White, 1973]. For example, one study found that students learned mathematical and scientific skills most quickly when these skills were presented in a particular order [Gagne, 1962]. We seek to understand the extent that similar skill-based orderings characterize LM training. Such orderings, if they exist, may provide a better understanding of LMs as well as a mechanism for data-efficient training. For instance, to train an LM for Spanish question generation, we wish to know if training first on related but simpler tasks, such as Spanish grammar and English question generation, helps.

We study if the idea of skill orderings can help us build a framework that relates data to LM training and behavior. This requires addressing two challenges revolving around the connection between skills and data. First, in order to show that there exist sets of skills that the LM learns most efficiently in some particular order, *an operational definition of LM skill and skill ordering must be developed and validated on data.* In initial experiments, we investigated if semantic groupings of data, such as metadata attributes or embedding clusters, were sufficient to represent a skill and characterize how models learn. For instance, we partitioned the Alpaca dataset [Taori et al., 2023] by instruction type—a technique used to capture dataset diversity [Wang et al., 2022a]—but we found that sampling based on instruction types and random sampling resulted in similar model performance, suggesting that not just any existing notion of data groups can characterize skills.

Second, *these definitions of skills must be used to construct sampling distributions to actually improve model training*. To develop criteria for a data selection algorithm that learns skills efficiently, we identify challenges that naive selection approaches face. The standard approach of random uniform sampling over data fails to learn skills optimally due to not accounting for skill imbalance and ordering. Skills can be distributed unevenly in the data, with more complex skills being rare—for instance, Spanish and question generation (QG) are 5% and 4% of the Natural Instructions dataset [Wang et al., 2022b], respectively, but Spanish QG is only 0.2%. Random sampling also provides no mechanism for taking into account a particular training order and dependency structure on skills. More sophisticated techniques like curriculum learning account for sample-level ordering, but not skills or their dependencies. Our goal framework must account for these issues of imbalance and ordering.

Skill-based framework We define a *skill* as a unit of behavior that a model can learn using an associated slice of data (Definition 5.1). An *ordered skill set* is a collection of skills with a directed *skills graph* that is neither complete nor empty, where an edge from a prerequisite skill to a skill exists if the amount of training it takes to learn the skill can be reduced if the prerequisite skill is also learned (Definition 5.2, Figure 5.1 left, center). We show that ordered skill sets exist in synthetic and real datasets using this operational definition. Interestingly, the existence of these ordered skill sets unveils that one can learn a skill quickly not by training solely on that skill, but on a mixture of that skill and prerequisite skills. For instance, in Figure 5.3 we observe that Spanish QG can be learned more efficiently when the model also learns English QG and Spanish—we can achieve 4% lower validation loss than training on only Spanish QG over a fixed budget of overall training steps.

Next, given an ordered skill set to train on, we use our framework to propose methods for how to select data so that the LM learn skills faster: skill-stratified sampling and an online generalization, Skill-it. We address the issue of unevenly distributed skills in datasets by proposing skill-stratified sampling, a simple approach that allows us to explicitly optimize for learning skills by uniformly sampling relevant skills (such as a target skill and its prerequisite skills in fine-tuning). Skill-stratified sampling uses the construction of the ordered skill set but is static, which does not incorporate the ordering as training proceeds and results in oversampling skills that may be already learned early on in training. We address this issue by proposing an online data selection algorithm, Skill-it, for selecting mixtures of training skills that allocates more weight towards learning skills that are not

yet learned or towards prerequisite influential skills (Figure 5.1 right). SKILL-IT is derived from an online optimization problem over the training skills for minimizing loss on a set of evaluation skills given a fixed budget of data and the skills graph. SKILL-IT is inspired by online mirror descent and can be adapted for continual pre-training, fine-tuning, or out-of-domain evaluation depending on the relationship between the evaluation skill set and the training skill set.

We evaluate SKILL-IT on synthetic and real datasets at two model scales, 125M and 1.3B parameters. For the continual pre-training setting, we show on the LEGO synthetic [Zhang et al., 2022c] that we obtain a 35.8 point improvement in accuracy over randomly selecting training data and curriculum learning [Bengio et al., 2009]. For the fine-tuning setting, we show that on the widely-used Natural Instructions dataset [Mishra et al., 2022, Wei et al., 2021], our algorithm over a mixture of skills is able to achieve up to 13.6% lower loss on that skill than solely training on that skill, given the same overall training budget. For the out-of-domain setting when our training skills do not align perfectly with evaluation skills, our algorithm is able to achieve the lowest loss on 11 out of 12 evaluation skills corresponding to task categories in the Natural Instructions test tasks dataset over random and skill-stratified sampling on the training data. We finally apply our framework to a case study on the recent RedPajama 1.2 trillion token dataset [Together, 2023]. We use the data mixture produced by SKILL-IT to continually pre-train a 3B parameter model. We find that SKILL-IT achieves higher accuracy with 1B tokens than uniform sampling over data sources with 3B tokens.

5.1 Skills framework

First, we propose definitions of skills and ordered skill sets in order to formalize our intuition around how models learn skills, and we demonstrate that not just any existing notion of data groups can characterize an ordered skill set in the dataset. Then, we demonstrate the existence of ordered skill sets on synthetic and real data, which show how viewing data through a skills-based framework can help with training and understanding model performance. Finally, we explore unsupervised skill recovery from data, finding that embedding-based approaches do not adequately recover synthetic skills.

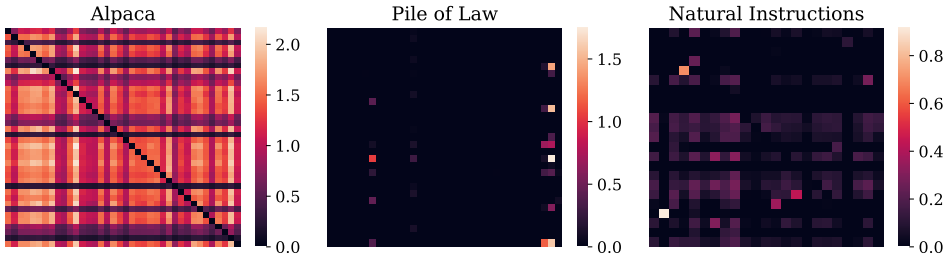


Figure 5.2: Heatmaps of adjacency matrices we compute for skill graphs for Alpaca, Pile of Law, and Natural Instructions. Negative elements and diagonals are thresholded to 0 for clarity. See Appendix 5.C.2 for descriptions of how they were constructed and larger versions.

5.1.1 Definitions

We first present a definition of an individual skill. Let the input space of all possible text data be $\mathcal{X}$, where $x \in \mathcal{X}$ is an individual text sample that a next-token-prediction LM $f \in \mathcal{F} : \mathcal{X} \to \mathcal{X}$ is trained on. We quantify learning via a metric $L : \mathcal{F} \times \mathcal{X} \to \mathbb{R}$, which maps from a model and evaluation data to a scalar quantity. In our setup, we use the cross-entropy validation loss applied over next-token predictions as our metric L.

Definition 5.1 (Skill)**.** *A* skill s *is a unit of behavior with associated data* $\mathcal{X}_s \subseteq \mathcal{X}$ *such that if* f *is trained on an dataset* $\mathcal{D}_s \subset \mathcal{X}_s$*, then* f *has improved metric* L *on samples belonging to* $\mathcal{X}_s \backslash \mathcal{D}_s$ *on average.*

This definition of a skill is flexible—it simply means that given a training dataset associated with the skill, a model f has an improved metric (e.g., decreasing validation loss) when evaluated on validation data associated with this skill. Under this definition, a skill could be a granular task, such as Spanish question generation for a subset of Wikipedia articles, or can be defined over a data source, such as next-token prediction of legal data from tax court rulings. However, our next definition, the ordered skill set, has a more specific construction and provides a framework for how models learn across dependent skills.

Definition 5.2 (Ordered skill set, skills graph)**.** *An* ordered skill set *for* f *is a collection of skills* $\mathcal{S} = \{s_1, \ldots, s_k\}$ *over which there is a directed skills graph* $G = (\mathcal{S}, E)$ *on the skill set that is neither complete or empty, where* $(s_i, s_j) \in E$ *if the amount of data needed to learn* s_j *when uniformly sampling from* $\mathcal{D}_{s_i} \cup \mathcal{D}_{s_j}$ *is no more than the amount of data needed when sampling*

only from $\mathcal{D}_{s_j}$. *We equate learning a skill* s_j *to* f *attaining a certain value of* L *or lower on average over* $\mathcal{X}_{s_j} \setminus \mathcal{D}_{s_j}$.

This definition isolates complete and empty graphs as extrema that do not capture meaningful sets of skills. We discuss the three types of skill graphs—complete, empty, intermediate—and their implications for data selection. In particular, we discuss how several initial attempts of defining skills over datasets via semantic groupings resulted in the extrema cases (see Appendix 5.C.2 for full results):

- The complete graph demonstrates that all skills influence each other. A random partition is an example of a skill set that yields a complete graph. This graph suggests that the best approach for learning any skill or set of skills is random sampling on the dataset. This is not a setting where we can gain much with skill-based sampling. For example, using instruction types as skills on the Alpaca dataset results in a nearly complete estimated skills graph (97.4% dense), and we find that stratified sampling on these skills only improves validation loss per skill by 0.007 points over random sampling on average (Figure 5.2 left), suggesting that utilizing skills does not improve model performance in this case.
- The empty graph demonstrates that each skill is independent. This can occur if skills are too granular; for instance, learning Spanish math problems is unlikely to help with English poem generation. This graph suggests that the best approach for learning an individual skill is to train on the skill itself. We see that empty graphs exist in real data; in Figure 5.2 (center), using data sources as skills on the Pile of Law [Henderson* et al., 2022] results in a nearly empty skills graph (3.9% dense).
- Graphs that are neither empty nor complete thus suggest a nontrivial order of how skill influence each other. *This is the setting in which we expect that identifying skills and exploiting their ordering will help the most.* In Figure 5.2 right, we use task categories, which capture broader reasoning patterns, as skills on Natural Instructions and find that the estimated graph has intermediate density (42.7% dense). We show concrete examples of how skills can be learned more efficiently on Natural Instructions in Section 5.1.2.

While these intuitive groupings result in ordered skill sets on some datasets (e.g., task categories on NI), this is not always the case (e.g., instruction types on Alpaca and sources on Pile of Law). Even though these groupings capture some notion of diversity in the dataset, our findings suggest that not just any semantic grouping induces an ordered skill set. We

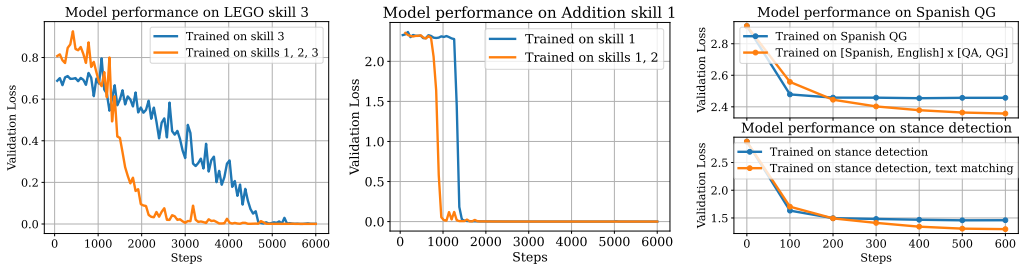


Figure 5.3: On the LEGO synthetic, 3-digit addition, and Natural Instructions, we identify examples of ordered skill sets in which training on a mixture of skills helps learn an individual skill faster than just training on that skill itself, given a fixed training budget.

now empirically demonstrate that our definition of ordered skill sets aligns with how models learn and can be exploited for more data-efficient training.

5.1.2 Examples of skills and ordered skill sets

We provide examples of ordered skill sets on the LEGO synthetic dataset, an addition synthetic dataset, and subsets of the Natural Instructions dataset. On these datasets, we find that certain skills are better learned when trained along with their prerequisite skills rather than in isolation.

LEGO skills. The LEGO synthetic, first introduced in [Zhang et al., 2022c], can evaluate a model's ability to follow a chain of reasoning. In this synthetic, the letters of the alphabet, $\mathcal{A}$, are variables each with some binary label in $\{0, 1\}$. An individual sample consists of k clauses for some fixed k across the dataset, each of the form $a = gx$ where $a, x \in \mathcal{A}$ and g is either a negation ("not") or assertion ("val"), e.g. we assign a to the value of x, or we assign a to the opposite label. At the end of the sentence, we prompt the model for what the value of one of these variables is. Two samples $x \in \mathcal{X}$ are given below for $k = 5$:

> Input: b = not y, r = val 1, m = val b, q = val m, y = not r. Output: b = 1.
>
> Input: c = val x, p = val f, x = val k, f = not c, k = val 0. Output: k = 0.

These samples each correspond to a chain of reasoning; for instance the first sample has the chain r, y, b, m, q, where knowing q's label requires the most reasoning steps. We define the ith skill s_i as the model's ability to know the ith variable of the chain. From our example above, the first sample belongs to $\mathcal{X}_{s_3}$ and the second sample belongs to $\mathcal{X}_{s_1}$. To

demonstrate the existence of ordered skill sets, we continually pre-train the 125M parameter GPT-Neo model [Gao et al., 2020b, Black et al., 2021] over various mixtures of LEGO skills with $k = 5$. In Figure 5.3 (left), we find that in 35.9% fewer training steps, training on a balanced mixture of $\mathcal{X}_{s_1}, \mathcal{X}_{s_2}$, and $\mathcal{X}_{s_3}$ resulted in the same validation loss of 0.01 as training solely on $\mathcal{X}_{s_3}$. This suggests that s_1, s_2 helped unlock performance on s_3 and that there exist edges from s_1 or s_2 to s_3 in the skill graph. Additional observations are available in Appendix 5.D.1, where we examine other edges as well as more complex reasoning chains, and the full skills graph corresponding to the ordered skill set for LEGO with $k = 5$ is in Figure 5.10.

Addition skills. We consider a variant of a synthetic 5-digit addition dataset analyzed in [Nanda et al., 2023]. We show the existence of ordered skill sets for a simplified 3-digit addition dataset where we treat each digit prediction as a skill—the outputs, in this case, are the integers $\{0, 1, ..., 9\}$. Examples are of the following form:

> Input: A = 1 0 6 + 0 7 1 , A 0 = ? Output: 7 Input: A = 6 0 6 + 8 7 9 , A 2 = ? Output: 4

where 'A 0' refers to the ones digit of the output (s_1) and 'A 2' refers to the hundreds digit (s_3). In Figure 5.3 (center), we find that in 32% fewer training steps, training on a balanced mixture of $\mathcal{X}_{s_1}$, and $\mathcal{X}_{s_2}$ resulted in the same validation loss of 0.01 as training solely on $\mathcal{X}_{s_1}$. That is, the ones digit addition skill can be improved by simultaneously learning the tens digit addition skill, even though the former should not require information from the latter—this is in line with observations from prior work that models do not always learn the ones digit addition first [Nanda et al., 2023]. The full skills graph corresponding to the ordered skill set over 3-digit addition is in Figure 5.11.

Natural Instructions (NI) skills. We show that ordered skill sets exist in NI [Wang et al., 2022b] when we treat task categories as skills.

- In Figure 5.3 (top right), we show that ordered skill sets exist over crosslingual task categories. Training on Spanish question generation (QG) along with equal parts of English QG, Spanish question answering (QA), and English QA results in 4.1% lower validation loss than training only on Spanish QG. Remarkably, the former only uses 25% of

the latter's Spanish QG data. This suggests that there are edges from Spanish QA, English QA, and English QG to Spanish QG.

- In Figure 5.3 (bottom right), we see that training on the task category Text Matching along with Stance Detection helps decrease the loss on Stance Detection by 11%. This suggests that these categories, which both involve understanding the relationship between two input texts, share an edge.

The full skills graphs corresponding to the ordered skill sets over these task categories are in Figure 5.13. While equating task categories to skills may be noisy, these examples suggest that there is signal within real data that suggests that ordered skill sets can improve data efficiency.

5.1.3 Skill recovery

A final component of characterizing skills is unsupervised recovery of ordered skill sets. We consider embedding-based clustering approaches and a loss-based clustering approach for recovering LEGO skills. When clustering data using various trained and pre-trained embeddings, we find that they were unable to achieve above 39% accuracy on LEGO. Instead, we find that taking 10 random training runs and clustering data by their *loss* per timestep per run recovers the skills with 61% accuracy (Table 5.3). The intuition behind this method is that the validation losses on points from the same skill have similar trajectories as models learn. We discuss this approach more in Appendix 5.D.2.

5.2 Skills-based data selection

Now that we have established the existence of ordered skill sets, we discuss how to use them for data selection. We state the data selection problem for learning across skills in Section 5.2.1. We discuss how to learn the skills graph that will be exploited in our data selection methods in Section 5.2.2. We then introduce two sampling methods that utilize the graph, a simple skill-stratified sampling method and the online sampling method Skill-it, in Section 5.2.3.

5.2.1 Problem statement

We are given an ordered training skill set $\mathcal{S}_{\text{train}} = \{s_{\text{train},1}, \ldots, s_{\text{train},k}\}$ on the training data, each with associated support set $\mathcal{X}_{s_{\text{train},1}}, \ldots \mathcal{X}_{s_{\text{train},k}}$, and an ordered evaluation skill set

Table 5.1: Summary of three settings—continual pre-training, fine-tuning, and out-of-domain. These settings are determined by how $\mathcal{S}_{\text{eval}}$ is defined and result in different skills graphs used for our sampling methods.

Setting	$\mathcal{S}_{\text{eval}}$	Skills graph
Continual pre-training	$\mathcal{S}_{\text{eval}} = \mathcal{S}_{\text{train}}$	$A \in \mathbb{R}^{k \times k}$, edges among all $\mathcal{S}_{\text{train}}$
Fine-tuning	$\mathcal{S}_{\text{eval}} \subset \mathcal{S}_{\text{train}}$	$A \in \mathbb{R}^{k \times m}$, edges from all training skills to target skill subset
Out-of-domain	$\mathcal{S}_{\text{eval}} \cap \mathcal{S}_{\text{train}} = \emptyset$	$A \in \mathbb{R}^{k \times m}$, edges from all training skills to separate evaluation skill set

$\mathcal{S}_{\text{eval}} = \{s_{\text{eval},1}, \ldots, s_{\text{eval},m}\}$ of m evaluation skills on a separate evaluation dataset. We aim to select n samples from $\mathcal{S}_{\text{train}}$ via a mixture of training skills, $p \in \Delta^{k-1}$, to achieve three goals depending on how $\mathcal{S}_{\text{eval}}$ is constructed:

- Continual pre-training: when $\mathcal{S}_{\text{eval}} = \mathcal{S}_{\text{train}}$, our goal is select a mixture of training skills to learn all of them.
- Fine-tuning: when $\mathcal{S}_{\text{eval}} \subset \mathcal{S}_{\text{train}}$, our goal is to select a mixture of training skills to learn an individual target skill or subset of these skills.
- Out-of-domain: when $\mathcal{S}_{\text{eval}} \cap \mathcal{S}_{\text{train}} = \emptyset$, our goal is to select a mixture of training skills to learn a disjoint set of evaluation skills we cannot train on. This can arise when we have a separate downstream validation dataset or the skills identified in the training dataset are noisy.

Furthermore, we have a skills graph $G = (\mathcal{S}_{\text{train}} \cup \mathcal{S}_{\text{eval}}, E)$, where $E \subseteq \mathcal{S}_{\text{train}} \times \mathcal{S}_{\text{eval}}$ and $A \in \mathbb{R}^{k \times m}$ is a weighted adjacency submatrix, where A_{ij} describes the strength of the edge from $s_{\text{train},i}$ to $s_{\text{eval},j}$. In Table 5.1, we summarize how the three different settings are constructed and how A varies across them. Next, we discuss how A can be estimated from the data.

5.2.2 Skills graph learning

The skills graph is important for determining how to sample from the ordered skill set for training efficiently. We present two approaches for learning the skills graph—brute-force and linear approximation. Algorithms are provided in Appendix 5.B.2. By definition 5.2, the brute-force way of identifying edges involves fixing an overall training budget of H steps and 1) training and evaluating the model on each s_i and 2) training the model on each pair of (s_i, s_j) and evaluating on s_i and s_j. If the loss on s_j when trained on both s_i and s_j is lower, there exists an edge from s_i to s_j. This approach has runtime $\mathcal{O}(Hk^2)$, which

is feasible for small k. When k is large, we can approximate this approach in linear time by training on each s_i for $h < H$ steps and setting $A_{ij} > 0$ if the loss on s_j decreases over h steps for a runtime of $\mathcal{O}(hk)$. This linear approach is necessary in the out-of-domain setting when $\mathcal{S}_{\text{eval}}$ and $\mathcal{S}_{\text{train}}$ are disjoint, as we do not train on data associated with $\mathcal{S}_{\text{eval}}$. In addition, both graph learning approaches can be performed on a smaller model, and the learned graph can be used for data selection for training a larger model (Appendix 5.D.4).

5.2.3 Skills graph-aware sampling

We present two approaches for sampling over the mixture of training skills according to the skills graph: skill-stratified sampling, which samples uniformly over relevant training skills according to A, and SKILL-IT, which is an online generalization that incorporates knowledge of how skills are being learned throughout training.

5.2.3.1 Skill-stratified sampling

A straightforward sampling approach is to discard training skills that do not benefit the evaluation skills and sample uniformly over the set of relevant training skills, which we call *skill-stratified sampling*. For continual pre-training, the relevant skills are the entire training skill set; for each $s_{\text{train},i} \in \mathcal{S}_{\text{train}}$, $\Pr(s_{\text{train},i}) = \frac{1}{k}$. This enables each skill to have sufficient training data. For fine-tuning, the relevant skills are the target skills and prerequisite skills, which can be identified via positive entries of the ith column of A with $\mathcal{S}_{\text{prereq}} = \{s_{\text{train},i} : \exists\, s_{\text{eval},j} \text{ s.t. } A_{ij} > 0\}$. We then set $\Pr(s) = \frac{1}{|\mathcal{S}_{\text{prereq}} \cup \mathcal{S}_{\text{eval}}|}$ for $s \in \mathcal{S}_{\text{prereq}} \cup \mathcal{S}_{\text{eval}}$. For the out-of-domain setting, skill-stratified sampling is over the set of prerequisite skills. For each $s \in \mathcal{S}_{\text{prereq}}$, we set $\Pr(s) = \frac{1}{|\mathcal{S}_{\text{prereq}}|}$. Next, we propose our online algorithm that exploits the graph dynamically for more efficient training.

5.2.3.2 SKILL-IT online data selection algorithm

Despite accounting for prerequisite skills, one shortcoming of skill-stratified sampling is that even if a skill has already obtained sufficiently low validation loss early during training, we will continue to allocate the same weight to that skill throughout training. Therefore, we formulate our data selection problem as an online learning problem and propose SKILL-IT, which both prioritizes prerequisite skills and skills that are not yet learned.

Algorithm 1 SKILL-IT Online Data Selection Algorithm

Input: Ordered training skill set $\mathcal{S}_{\text{train}}$, ordered evaluation skill set $\mathcal{S}_{\text{eval}}$. Learning rate η, T rounds, n samples, H training steps per run for graph learning, model f_1, window parameter w.
$A \leftarrow \text{LEARNGRAPH}(\mathcal{S}_{\text{train}}, \mathcal{S}_{\text{eval}}, H, f_1)$ (Alg. 2, 3).
Initialize $p_1^i = \exp(\eta \sum_{j=1}^{m} A_{ij})$ for all $i \in [k]$, the softmax over A.
for $t = 1, \ldots, T-1$ **do**
    Observe losses $L_{\text{eval},j}(f_t)$ for all $s_{\text{eval},j} \in \mathcal{S}_{\text{eval}}$.
    Train model f_t with n/T samples from mixture p_t over $\mathcal{S}_{\text{train}}$. Update model $f_{t+1} = \Phi(f_t, p_t)$.
    Set $p_{t+1}^i = \exp(\eta \sum_{\tau=t-w+1}^{t} \sum_{j=1}^{m} A_{ij} L_{\text{eval},j}(f_\tau))$.
end for

We are given a budget of T rounds and n total samples to train on. At round t, we select a mixture $p_t \in \Delta^{k-1}$ from the k-dimensional unit simplex, and for each training skill $s_{\text{train},i} \in \mathcal{S}_{\text{train}}$, we sample from $\mathcal{X}_{s_{\text{train},i}}$ with proportion p_t^i for a total of $\frac{n}{T}$ samples per round. Let f_t be the model at at the start of round t. We can define f_t recursively as a function of the previous round's model f_{t-1} and mixture p_{t-1} via a dynamics function $\Phi : \mathcal{F} \times \Delta^{k-1} \to \mathcal{F}$; that is, $f_t = \Phi(f_{t-1}, p_{t-1})$. Let $L_{\text{eval},j}(f_t)$ be the validation loss of f_t on $s_{\text{eval},j}$. Our goal is to select $p_1, \ldots, p_T$ to minimize loss per evaluation skill at the end of training:

$$\underset{p_1,\ldots,p_T \in \Delta^{k-1}}{\text{minimize}} \frac{1}{m} \sum_{j=1}^{m} L_{\text{eval},j}(f_T). \tag{5.1}$$

This optimization problem is challenging to solve without additional assumptions. In order to make the problem tractable, we impose an explicit dynamics rule for the each evaluation skill's loss $L_{\text{eval},j}$ in terms of the current loss and data mixture. Assuming for simplicity that $\mathcal{S}_{\text{eval}} \subseteq \mathcal{S}_{\text{train}}$, a simple rule would be $L_{\text{eval},j}(f_t) = L_{\text{eval},j}(\Phi(f_{t-1}, p_{t-1})) := L_{\text{eval},j}(f_{t-1})(1 - \alpha p_{t-1}^j)$ for $\alpha \in [0, 1]$. That is, we expect that allocating more data to skill j should result in the validation loss on skill j decreasing. However, such an expression assumes that only training on the jth skill will help learn the jth skill. Instead, Section 5.1.2 suggests that there are other skills that may help with the jth skill. We propose the following dynamics:

$$L_{\text{eval},j}(f_t) = L_{\text{eval},j}(f_{t-1})(1 - A_{:,j}^\top p_{t-1}), \tag{5.2}$$

where $A_{:,j}$ is the column with weights of all skills that influence $s_{\text{eval},j}$, and we absorb the scalar α into A. The optimization problem in (5.1) can thus be simplified as follows:

$$\begin{aligned} &\underset{p_1,\ldots,p_T \in \Delta^{k-1}}{\text{minimize}} \ \frac{1}{m} \sum_{j=1}^{m} L_{\text{eval},j}(f_T) \\ &\text{s.t} \quad f_t = \Phi(f_{t-1}, p_{t-1}) \ \forall t = 1, \ldots T \\ &\qquad L_{\text{eval},j}(f_t) = L_{\text{eval},j}(f_{t-1})(1 - A_{:,j}^{\top} p_{t-1}) \ \forall j \in [m] \end{aligned} \tag{5.3}$$

In Appendix 5.B, we derive the following update rule via online mirror descent [Nemirovskij and Yudin, 1983] for learning rate $\eta > 0$:

$$p_{t+1}^{i} = p_t^{i} \exp\left(\eta \sum_{j=1}^{m} A_{ij} L_{\text{eval},j}(f_t)\right). \tag{5.4}$$

In addition, when equation 5.4 is expanded, we have that

$$p_{t+1}^{i} = p_1^{i} \exp\left(\eta \sum_{\tau=1}^{t} \sum_{j=1}^{m} A_{ij} L_{\text{eval},j}(f_\tau)\right).$$

Since this summation over τ results in diminishing strength of updates, we change it to a moving window of size w. Our full method is in Algorithm 1.

Intuitively, at each step we adjust the weight on skill i based on the losses of skills that i influences, with the assumption that more training data helps decrease loss. Note that when we use our algorithm with a complete graph or empty graph, we achieve expected behavior discussed in Section 5.1.1. For the complete graph, our algorithm reduces to stratified sampling. When we have a skill set with an empty graph, the update rule reduces to sampling proportional to each skill's validation loss.

5.3 Experimental results

Given an ordered skill set, we aim to validate Skill-it's ability to select data for efficiently learning skills in the continual pre-training, fine-tuning, and out-of-domain settings. We provide full tables of results in Appendix 5.D.3.1 and results where we learn the skills graph

on the 125M model and use it for the 1.3B parameter model in Appendix 5.D.4. Skills graphs are in Appendix 5.C.2, weight trajectories for SKILL-IT are in Appendix 5.D.3.2, and ablations on the graph and online components of SKILL-IT are in Appendix 5.D.5.

5.3.1 Continual pre-training

Setup. We evaluate the ability of SKILL-IT to select data for efficiently learning over all skills. We measure average validation loss per skill after a fixed number of training steps. We construct the LEGO synthetic and addition synthetic with $k = 5$ and 3, respectively, and an imbalanced dataset over the skills. On the Natural Instructions dataset, we use 23 of the task categories as skills.

Baselines. We compare SKILL-IT against three baselines that do not account for skills: random sampling, curriculum learning, and anticurriculum learning. Random sampling is a standard procedure for selecting samples given no additional information. Curriculum learning [Bengio et al., 2009] and anticurriculum learning [Wu et al., 2020] score the samples from easiest to hardest and vice versa, respectively, and sample over an expanding set of the lowest scored samples at every epoch; we use the pre-trained model's loss to rank points. We evaluate skill-stratified sampling, which uses knowledge of the skills but is not online, and include an additional skills curriculum baseline in Appendix 5.D.3.1

Analysis. Our results are shown in Figure 5.4. Across our experiments we find that SKILL-IT outperforms baselines that do not use skills as well as skill-stratified sampling. On the LEGO dataset, all three baselines that do not utilize a notion of skills exhibit plateauing loss on four of the skills. Both skill-stratified sampling and SKILL-IT are able to significantly reduce loss on all skills, but the former is slower. Halfway through training, SKILL-IT exhibits an accuracy improvement between 9.9 and 25.9 points over other approaches, reaching a final accuracy of 99.4 (Figure 5.19). SKILL-IT outperforms skill-stratified sampling by initially allocating more weight to prerequisite skills and eventually allocating more weight to skills that are learned more slowly (Figure 5.20). On the addition synthetic with $k = 3$, SKILL-IT converges to near-zero validation loss faster than the baselines on skills 1 and 2. While the random baseline may seem competitive at first glance, it fails to learn skill 1 (adding together the ones digits), which hurts its average loss per skill. On NI, the validation loss

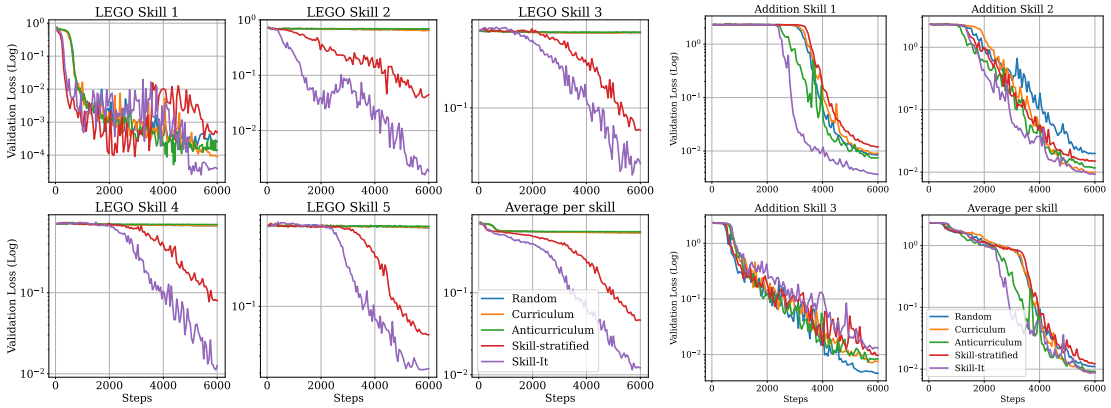


Figure 5.4: Performance of Skill-it on each skill in the continual pre-training setting (learning over all skills in the ordered training skill set) on the LEGO synthetic (left) and addition synthetic (right).

from Skill-it is 3.2% lower than from random sampling (Table 5.7). Our results suggest that exploiting the construction and ordering of skills is critical to learning skills quickly.

5.3.2 Fine-tuning

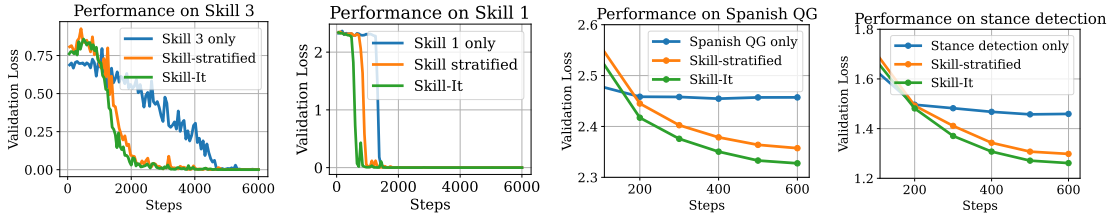


Figure 5.5: Performance of Skill-it in the fine-tuning setting (learning a target skill using the ordered training skill set) on LEGO, addition, and NI.

Setup. We evaluate the ability of Skill-it to select data from an ordered training skill set for learning a target skill. Mirroring Figure 5.3, we evaluate on LEGO target skill 3 (third in reasoning chain), on the addition synthetic's skill 1 (ones place digit addition), and on NI's Spanish QG and Stance Detection.

Baselines. We compare Skill-it against training on the target skill only and skill-stratified sampling over prerequisite skills and the target skill. The skill-stratified sampling approach

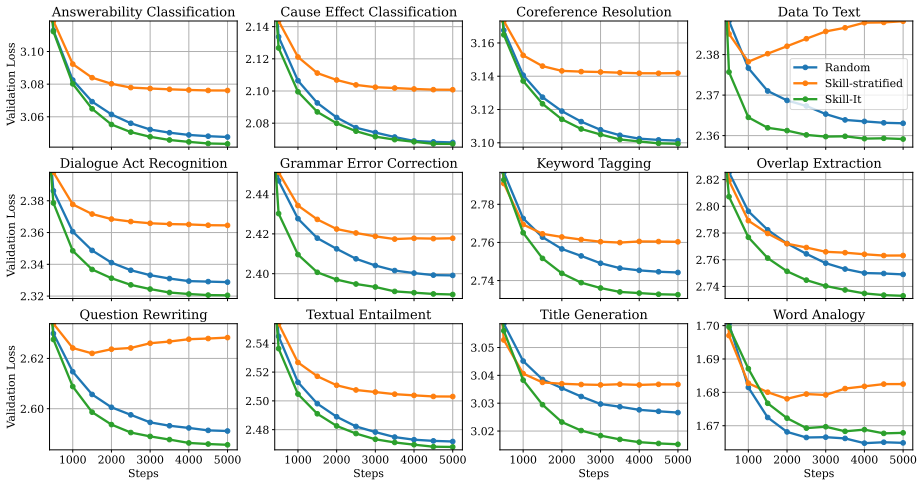


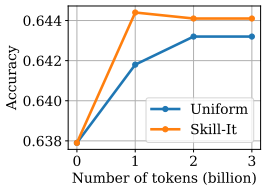
Figure 5.6: Performance of Skill-it in the out-of-domain setting for the NI test task split. Skill-it uses the graph between the train and evaluation skills to produce an online mixture on the training dataset.

uses the ordered skill set to identify prerequisite skills, but does not exploit them dynamically.

Analysis. Our results are shown in Figure 5.5. On LEGO, Skill-it results in the same validation loss of 0.01 as training only on the target skill in 38.1% fewer steps. We observe a similar trend on addition, with Skill-it converging to a validation loss of 0.01 in 59% fewer steps required to do so when training only on the target skill. Finally, on NI, Skill-it improves validation loss on Spanish question generation by 5.3% and Stance Detection by 13.6% over just training on the respective target skill only. In this setting, a significant portion of the improvement over training only on the target skill comes from identification of prerequisite skills through the learned graph in the skill-stratified sampling method. Skill-it is further able to improve performance with finer-grained dynamic weighting on prerequisite skills.

5.3.3 Out-of-domain setting

Natural Instructions. We evaluate the ability of Skill-it to select data from a set of training skills for learning a disjoint set of evaluation skills that we cannot train on. We use all 59 task categories in the NI train tasks split as the training skills and the 12 task categories in the test tasks split as our evaluation skills. We compare Skill-it against random and



RedPajama source	Skill-it mixture
ArXiv	0.1370
Books	0.0437
C4	0.4195
CommonCrawl	0.0732
GitHub	0.189
StackExchange	0.0892
Wikipedia	0.0484

Figure 5.7: Left: Accuracy on LM Evaluation Harness for continual pre-training of a 3B parameter model using Skill-it on the RedPajama dataset. We achieve higher accuracy at 1B additional tokens than uniform at 3B tokens. Right: Skill-it mixture over RedPajama sources.

skill-stratified sampling, both of which do not exploit the relationships between training skills and evaluation skills. Skill-it achieves the lowest loss on 11 out of 12 task categories over random and skill-stratified sampling (Figure 5.6, tables in Appendix).

RedPajama. We use Skill-it to produce a data mixture on the RedPajama dataset. The training skills are the data sources comprising the dataset, and the evaluation skills are several tasks from the Language Model Evaluation Harness [Gao et al., 2021a]. Skill-it with $\mathsf{T} = 1$ (i.e. a static, graph-based mixture) yields the mixture in Figure 5.7 (right). We continually pre-train a 3B parameter model trained on one trillion tokens for three billion additional tokens using this mixture, and see that it outperforms uniform sampling over the data sources (Figure 5.7 left). In particular, Skill-it achieves higher accuracy with 1B additional tokens than uniform with 3B additional tokens.

5.4 Related work

Data selection for LMs. There have been several studies of large-scale data selection for LMs. Data deduplication [Lee et al., 2022, Abbas et al., 2023, Hernandez et al., 2022], in which identical or nearly identical samples are removed, is a method that enables LMs to be trained on smaller, cleaned datasets and has been increasingly used as a pre-processing step for training data [Touvron et al., 2023a, Biderman et al., 2023, Zhang et al., 2022b]. Other methods applied at scale involve ensuring high quality of data by explicitly filtering out samples or comparing the training dataset with a cleaned reference dataset [Brown et al., 2020b, Touvron et al., 2023a, Laurençon et al., 2023]. Importance reweighting approaches have also

been proposed for identifying training data from a large corpus that best approximates a smaller target distribution [Xie et al., 2023b], and influence functions have been used to select a subset of training data to improve performance on downstream tasks [Wang et al., 2023]. These approaches can identify data pertaining to a particular target distribution or filter out low quality data according to some heuristic, while our work aims to understand how the choice of data is related to the numerous skills that LMs learn.

Recent development of LMs has shifted focus from emphasizing the scale of the model to prioritizing the training data utilized. For example, models like Alpaca [Taori et al., 2023], Vicuna [Chiang et al., 2023], and Koala [Geng et al., 2023] are all based on the LLaMA model combined with instruction data generated by an existing LM. Palm 2's technical report states that the data mixture was a critical component of the final model [Google, 2023], and Mosaic ML's recent MPT model was trained on a hand-engineered mixture of the RedPajama dataset [MosaicML, 2023]. However, these works lack rigorous explanation for why their training datasets were constructed in this way.

Finally, perhaps most related to our approach is the contemporary work DoReMi [Xie et al., 2023a], which uses group distributionally robust optimization on a smaller LM to select data source mixtures for training a larger LM. Their approach focuses on selecting data at the data source level for optimizing worst-case performance across the training data sources, rather than at the more general skills level for a variety of target skill sets. Furthermore, we focus on understanding how skills are related to each other and induce some order in how LMs learn by explicitly modeling skill graph structure, which we find to be important for data-efficient LM training (see ablations in Appendix 5.D.5).

Data selection methods. Many data selection methods have been proposed for supervised, task-specific settings. In this setting, the most typical objective is dataset condensation, which aims to identify a small subset of data that captures the larger dataset's properties with respect to the model. Some approaches include constructing coresets [Langberg and Schulman, Phillips, 2016], identifying samples that the model forgets during training [Toneva et al., 2018]; identifying samples with the largest gradients [Paul et al., 2021] or gradients that approximate the overall gradient [Mirzasoleiman et al., 2019]; clustering in embedding space and selecting points farthest from cluster centers [Sorscher et al., 2022]; and selecting samples with the highest uncertainty or entropy [Lewis, 1995]. These approaches have also been shown to transfer from smaller models to larger models [Coleman et al., 2019]. Unlike

these methods, we study how to select data for learning one or many skills at the mixture level for LMs instead of the instance level.

Another area of interest is data selection for domain adaptation and multitask learning. For domain adaptation, there are a wide range of methods that select data to best match the target distribution. For example, the Moore-Lewis method matches data based on the difference in cross-entropy using a model trained on the target versus a model trained on the source data [Moore and Lewis, 2010]. Several other approaches suggest training a model to distinguish between source and target and selecting points with high uncertainty [Ruder et al., 2017], or selecting points based on some divergence in an embedding space [Ruder and Plank, 2017]. In comparison to these approaches, our work focuses on learning one or many skills and also finds that embedding-based heuristics do not fully identify skills.

Data attribution. Another perspective on understanding training data is data attribution, which seeks to identify what data is responsible for particular model behaviors. Influence functions [Koh and Liang, 2017] and shapley values [Ghorbani and Zou, 2019] are two ways to quantify the role of individual samples. Datamodels [Ilyas et al., 2022] fit a model to predict behavior given a subset of training data, providing a framework for understanding individual samples as well as dataset counterfactuals. Simfluence [Guu et al., 2023] fits a Markov process to a set of training trajectories for finer-grained understanding of how data impacts training. We focus on understanding how groups of data associated with skills elicit broader model capabilities, and utilize this understanding to select data for more efficient training.

Curriculum learning. Curriculum learning [Bengio et al., 2009] proposes to show the model data in order from easy samples to hard ones. Various criteria have been used to determine hardness, and anticurriculum as well as various pacing functions and mixing rates have been explored [Soviany et al., 2022]. Curriculum learning can also be performed at the group level [Varshney et al., 2022]. More sophisticated approaches include parametrizing each sample with a dynamic importance [Saxena et al., 2019], and also accounting for irrelevant and noisy data [Mindermann et al., 2021]. Our approach similarly utilizes a curriculum, but it is defined over a skills graph and does not necessarily align with training on easiest to hardest skills.

How LMs learn. Many different explanations for how LMs learn from data have been proposed. One hypothesis is that there exist discrete, universal building blocks of LM knowledge called quanta, and power law scaling emerges from a learning over a particular distribution of quanta in the right order [Michaud et al., 2023]. Another is that chain of thought reasoning emerges due to local clusters of latent variables that influence each other, which can be validated by studying the LM's ability to do conditional inference given intermediate variables [Prystawski and Goodman, 2023]. Others have provided theoretical analysis of how transformers learn topics by studying co-occurrences of words in the training data [Li et al., 2023]. Empirically, how models learn is still a mystery—for instance, models trained on code are found to perform fairly well at commensense reasoning [Madaan et al., 2022]. Our work initiates a study on how LMs learn various skills and how to exploit this for better data selection.

Task selection. In multitask auxiliary learning, the goal is to train a model to perform well on a target task(s) by selecting the most beneficial source tasks to train on. One can use feature similarity to select tasks [Kung et al., 2021], but we find in our synthetics that feature similarity does not always recover skills. In Taskonomy [Zamir et al., 2018], a hypergraph over a set of tasks is learned and used to select tasks. The methods used to develop the taxonomy can be applied to further expand our graph learning (e.g., studying transitive and higher-order properties). However, their focus is on task selection in computer vision rather than data selection for LMs to learn skills. Lastly, the contemporary work of TaskWeb [Kim et al., 2023b] builds a graph among 22 common NLP tasks in order to determine what the best source tasks are for a target task. Their definition of an edge in the task graph is less strict than ours (their comparison is on if training on additional data from s_i helps with s_j, while we fix the overall amount of training data over both s_i and s_j). Overall, our approach is similar in use of the skills graph, but we incorporate it into a dynamic sampling algorithm. Furthermore, we look more broadly at skills, rather than tasks, and characterize when we expect using the skills graph to improve model performance.

Education. The notion of skill has been studied in education. Classical research on learning hierarchies [White and Gagné, 1974] identify sets of skills that make up subordinate capabilities for students. For instance, [Gagne and Paradise, 1961] identified that in order for students to solve linear equations, there were many prerequisite skills, ranging from the

simplest being symbol recognition to the most complex being the ability to add, subtract, multiple, and divide from both sides of the equation. More recently, decision-making over lesson sequences based on skills, e.g., what the student already knows versus what the lesson teaches, has become an area of interest in personalized learning [Reddy et al., 2016].

5.5 Conclusion

Given a fixed budget of data, knowing what data to train on to induce various capabilities in an LM is challenging. As LMs continue to improve, it will become increasingly important to extract as much signal as possible from the data and to direct that signal towards acquiring a broad variety of capabilities. In this chapter, we introduced a skills-based framework for understanding how LMs learn and for selecting training data. We hope our study invites others to build on such a notion of skill and further explore how to align skills with data.

5.A Broader Impacts and Limitations

Broader Impacts. As more LMs are developed, a key criteria for their adoption and utility is if they exhibit a wide array of useful capabilities, such as generating harmless content, summarizing essays, and being conversational with the user. While improvements in other parts of the LM development pipeline such as training and architecture are important, many recent advances in building LMs with a wide array of useful capabilities have come from the data itself [Google, 2023, Taori et al., 2023, Chiang et al., 2023, Geng et al., 2023, MosaicML, 2023]. Our work is fundamental in investigating how LMs learn and how to select data to learn skills more efficiently. However, we recognize that data selection methods can always be utilized to optimize for particular skills that may be considered malicious or negatively target or exclude specific groups [Bai et al., 2022]. Furthermore, pre-trained LMs have been found to have various biases [Kirk et al., 2021, Nadeem et al., 2021, Liang et al., 2021, Bommasani et al., 2021b].

Limitations. The skills graph can either be provided (e.g., using a knowledge graph) or learned. Our work learns the skills graph using Algorithm 2 or Algorithm 3, which requires initial training runs on pairs of skills or each skill, respectively. This can be made more efficient by performing these training runs on a smaller model and for fewer

number of steps, but tradeoffs here have yet to be thoroughly investigated. SKILL-IT also assumes that the ordered skill set is provided; as discussed in sections 5.1.1 and 5.1.3, it is challenging to recover ordered skill sets simply via metadata attributes or embedding clustering. Otherwise, the best way to sample over collections of skills that form a complete or empty graph is random or stratified sampling with no ordering to exploit. Our loss-based clustering approach presented in section 5.1.3 demonstrates that grouping by losses can provide an explanation for how skills are defined over data. An important direction for future work is to use such a clustering approach or other unsupervised algorithms in an end-to-end pipeline for skill discovery, skill graph learning, and data selection based on such skills.

5.B Additional Algorithmic Details

5.B.1 Derivation of SKILL-IT Update Rule

First, we provide the derivation of our update rule from online mirror descent using the proximal point view [Gupta, 2020]. We restate our optimization problem from (5.3):

$$\begin{aligned} &\underset{p_1,\ldots,p_T \in \Delta^{k-1}}{\text{minimize}} \ \frac{1}{m}\sum_{j=1}^{m} L_{\text{eval},j}(f_T) \\ &\text{s.t} \ \ L_{\text{eval},j}(f_t) = L_{\text{eval},j}(f_{t-1})(1-\alpha A_{:,j}^{\top} p_{t-1}) \ \forall j \in [m], t = 1,\ldots,T \\ &\quad f_t = \Phi(f_{t-1}, p_{t-1}) \ \forall t = 1,\ldots T \end{aligned} \tag{5.5}$$

Let $\bar{L}_t(p) = \frac{1}{m}\sum_{i=j}^{m} L_{\text{eval},j}(f_{t+1}) = \frac{1}{m}\sum_{i=j}^{m} L_{\text{eval},j}(\Phi(f_t, p))$; that is, p is the mixture we must choose at time t and $\bar{L}_t$ is the average loss per skill of the model after it is trained on p at round t. A greedy approximation of (5.5) is $\underset{p\in\Delta^{k-1}}{\text{minimize}}\ \bar{L}_t(p)$, given the model and mixtures at previous rounds. A linear approximation of $\bar{L}_t(p)$ is

$$\bar{L}_t(p) \approx \bar{L}_t(p_{t-1}) + \langle \nabla \bar{L}_{t-1}(p_{t-1}), p - p_{t-1} \rangle \tag{5.6}$$

Then, the problem of minimizing $\bar{L}_t(p)$ can be approximated as

$$\text{argmin}_{p\in\Delta^{k-1}} \langle \eta \nabla \bar{L}_{t-1}(p_{t-1}), p \rangle \tag{5.7}$$

after we drop terms from (5.6) that do not depend on p. Note that the η is a constant and does not impact the solution. The optimal solution to this problem is selecting the p that has the most weight on the slice with the largest gradient (e.g., a folow-the-leader sort of algorithm). To improve stability and prevent overfitting, we introduce regularization via a Bregman divergence $D_h(p||p_{t-1}) = h(p) - h(p_{t-1}) - \langle \nabla h(p_{t-1}), p - p_{t-1} \rangle$. After dropping terms that do not contain p, our problem is now

$$\text{argmin}_{p \in \Delta^{k-1}} \langle \eta \nabla \bar{L}_{t-1}(p_{t-1}), p \rangle + h(p) - \langle \nabla h(p_{t-1}), p \rangle \tag{5.8}$$

Taking the gradient and setting it equal to 0 gives us

$$\eta \nabla \bar{L}_{t-1}(p_{t-1}) + \nabla h(p) - \nabla h(p_{t-1}) = 0 \tag{5.9}$$

Similar to in standard multiplicative weights, we set $h(p) = \sum_i p_i \ln p_i$ and $\nabla h(p) = [\ln p_i + 1]_i$. Then,

$$\begin{aligned} \ln p^i &= \ln p^i_{t-1} - \eta \nabla_i L_{t-1}(p_{t-1}) \\ \Rightarrow p^i_{t+1} &= p^i_t \exp(-\eta \nabla_i \bar{L}_t(p_t)) \end{aligned} \tag{5.10}$$

where ∇_i is the ith element of the gradient. Now we wish to compute $\nabla_i \bar{L}_t(p_t) = \frac{1}{m} \sum_{j=1}^m \nabla_i [L_{\text{eval},j}(f_{t+1})] =$
$\frac{1}{m} \sum_{j=1}^m \nabla_i [L_{\text{eval},j}(\Phi(f_t, p_t))]$. Recall the dynamics model for L_{eval}:

$$L_{\text{eval},j}(f_{t+1}) = L_{\text{eval},j}(f_t)(1 - A^\top_{:,j} p_t), \tag{5.11}$$

The gradient of this model with respect to each training skill s_i is

$$\begin{aligned} \nabla_i L_{\text{eval},j}(f_{t+1}) &= -A_{ij} L_{\text{eval},j}(f_t) \\ \Rightarrow \nabla_i \bar{L}_t(p_t) &= \frac{1}{m} \sum_{j=1}^m -A_{ij} L_{\text{eval},j}(f_t) \end{aligned} \tag{5.12}$$

Algorithm 2 LEARNGRAPH (Brute-Force)

Input: Ordered skill set $\mathcal{S} = \{s_1, \ldots, s_k\}$. Number of training steps H, base model f.
for $j \in [k]$ **do**
  Train f on samples from $\mathcal{X}_{s_j}$ for H steps and denote $f_{H,j}$ to be the model after training.
  Observe change in loss, $\delta_j^j = L_{\text{eval},j}(f) - L_{\text{eval},j}(f_{H,j})$.
end for
for $i, j \in [k]$ **do**
  Train f on samples from $\mathcal{X}_{s_i} \cup \mathcal{X}_{s_j}$ for H steps and denote $f_{H,i,j}$ to be the model after training.
  Observe change in loss, $\delta_j^{i,j} = L_{\text{eval},j}(f) - L_{\text{eval},j}(f_{H,i,j})$.
  if $\delta_j^{ij} > \delta_j^j$ **then**
    Draw edge $s_i \to s_j$ and set $A_{ij} > 0$.
  end if
end for
Return Adjacency matrix $A \in \mathbb{R}^{k \times k}$

Plugging this back into (5.10),

$$p_{t+1}^i = p_t^i \exp\left(\eta \sum_{j=1}^{m} A_{ij} L_{\text{eval},j}(f_t)\right), \tag{5.13}$$

where we can absorb the $\frac{1}{m}$ into η.

5.B.2 Graph Learning Method

We provide algorithms for learning the graph over an ordered skill set. In Algorithm 2, we discuss the brute-force approach for learning the adjacency matrix. This approach only works when $\mathcal{S}_{\text{eval}} \subseteq \mathcal{S}_{\text{train}}$ (e.g. pre-training and fine-tuning cases), so we denote $\mathcal{S} = \mathcal{S}_{\text{train}}$ in the algorithm box. In Algorithm 3, we discuss the linear approach for learning the adjacency matrix. This approach works even in the out-of-domain case when $\mathcal{S}_{\text{eval}}$ and $\mathcal{S}_{\text{train}}$ are disjoint.

In both approaches, the exact value of A_{ij} can vary, but we can typically set it proportional to $\delta_j^{i,j} - \delta_j^j$, the difference between the changes in loss, in the brute-force case or δ_j^i, the change in loss itself, in the approximate case. The exact constructions and methods for learning each A in our experiments are in Appendix 5.C.2.

Algorithm 3 LearnGraph (Approximate)

Input: Ordered skill sets $\mathcal{S}_{\text{train}}$ and $\mathcal{S}_{\text{eval}}$. Number of training steps H, base model f.
for $i \in [k]$ **do**
  Train f on samples from $\mathcal{X}_{s_{\text{train},i}}$ for H steps and denote $f_{H,i}$ to be the model after training.
  for $j \in [m]$ **do**
    Observe change in loss, $\delta_j^i = L_{\text{eval},j}(f) - L_{\text{eval},j}(f_{H,i})$.
    If $\delta_j^i > 0$, draw edge $s_{\text{train},i} \to s_{\text{eval},j}$ and set $A_{ij} > 0$.
  end for
end for
Return Bipartite adjacency submatrix $A \in \mathbb{R}^{k \times m}$

5.C Additional Experimental Details

5.C.1 Datasets

We present details about each dataset used, including information on the skills and the validation dataset. A summary is presented in Table 5.2.

Table 5.2: We list each dataset used as well as its corresponding skill. We include the number of skills in the training dataset, as well as details on how the validation dataset is constructed.

Dataset	Skill	# skills	Validation data
Alpaca	Instruction type	38	50 samples per skill
Pile of Law	Legal data source	31	645 samples per skill
LEGO	Reasoning chain depth	5	100 samples per skill
Addition	Digit	3	100 samples per skill
NI (pre-training)	Task category	23	50 samples per task
NI (Spanish QG)	Task category $\times$ language	4	100 samples per task
NI (stance detection)	Task category	2	50 samples per task
NI (out-of-domain)	Task category	59, 12	400 samples per task
RedPajama	Data source	7	LM eval harness

- Alpaca dataset [Taori et al., 2023]: the Alpaca dataset consists of 52K instruction examples that were generated from text-davinci-003. We applied the Berkeley Neural Parser [Kitaev et al., 2019, Kitaev and Klein, 2018] to each instruction, keeping 40777 samples it was able to parse successfully. If the sample began with a question, we annotated it with the skill "question", and otherwise we annotated it with the verb identified from the parser. We

grouped the data into a total of 38 skills, such as "list", "edit", "calculate", "describe" and "identify".

- Pile of Law [Henderson* et al., 2022]: the Pile of Law dataset consists of various sources of legal and administrative data, ranging from tax rulings to the world's constitutions. We evaluate on a subset of the Pile of Law validation dataset consisting of 13883 samples, where we selected max(645, source size) samples per source. We truncated each sample to be no more than 100K characters.
- LEGO [Zhang et al., 2022c]: for the LEGO synthetic, we set $k = 5$ and sample 192000 points across the skills. Our validation dataset consisted of 100 samples per skill.
- Addition: for the 3-digit addition synthetic, we set $k = 3$ and sample 192000 points across the skills. We use a validation dataset of 100 samples per skill.
- Natural Instructions [Wang et al., 2022b, Mishra et al., 2022]: the Natural Instructions dataset is a large collection of tasks and their definitions in natural language. For the pre-training setting, we used a set of 23 task categories that had the largest degree (in-degree + out-degree) in the learned skills graph, for a total of $1,232,437$ samples and 425 tasks to select from. We evaluated on 50 samples per task.

  For the fine-tuning setting with Spanish question generation, we select data over 4 skills (Spanish question generation, Spanish question answering, English question generation, English question answering) for a total of 513210 samples and 212 tasks to select from. We evaluated on 100 samples per task.

  For the fine-tuning setting with stance detection, we select data over 2 skills (stance detection, text matching) for a total of 50990 samples and 19 tasks to select from. We evaluated on 50 samples per task.

  For the out-of-domain setting, we select data over all 59 task categories for a total of $2,417,867$ samples and 753 tasks to select from. The test split consisted of 12 task categories and 119 tasks, and we evaluated on min(400, task size) samples per task.
- RedPajama [Together, 2023]: the RedPajama dataset is a 1-trillion token dataset that aims to reproduce the LLaMA [Touvron et al., 2023a] training dataset. We select over the 7 data sources and evaluate using the LM evaluation harness [Gao et al., 2021a].

5.C.2 Graph Learning Details

We describe how the skills graph was learned on each dataset.

- Alpaca (Figure 5.8): we use Algorithm 3 and train for $K = 150$ steps per skill. Each edge $i \to j$ has a weight of δ_j^i, the difference in loss on skill j before and after training on i. Next, we compare the average validation loss of skill-stratified sampling versus random sampling when we train for $K = 1000$ steps. We find that skill-stratified sampling only does 0.007 better than random sampling, confirming that Alpaca's dense skills graph suggests that random sampling is the best we can do.
- Pile of Law (Figure 5.9): we use Algorithm 3 and train for $K = 150$ steps. Each edge $i \to j$ has a weight of δ_j^i, the difference in loss on skill j before and after training on i.
- LEGO (Figure 5.10): we use both Algorithm 2 and Algorithm 3 and train for $K = 6000$ steps each. Each edge $i \to j$ has a weight of 0.5 if the amount of data associated with skill j that is needed to reach 0.01 validation loss is less when training on (i, j) than on j (edges are set to 0 if 0.01 validation loss is not reached, even if loss is decreasing). Each edge $i \to j$ is also set to 0.5 if training on i decreases loss directly on j. We set each diagonal entry of A to be 1.
- Addition (Figure 5.11): we use Algorithm 2 and train for $K = 6000$ steps. Each edge $i \to j$ has a weight of 0.5 if the amount of data associated with skill j that is needed to reach 0.01 validation loss is less when training on (i, j) than on j (edges are set to 0 if 0.01 validation loss is not reached, even if loss is decreasing). We set each diagonal entry of A to be 1.
- Natural Instructions (Figure 5.12, 5.13, 5.14): we use Algorithm 3. For the pre-training setting, we train for $K = 600$ steps and assign each edge $i \to j$ a weight δ_j^i equal to the change in loss on j in the first 100 steps for all $i, j \in [k]$, including diagonal entries. For the fine-tuning setting, we train for $K = 600$ steps and assign each edge $i \to j$ a weight δ_j^i equal to the change in loss before and after training. For the out-of-domain setting, we train for $K = 600$ steps and assign each edge $i \to j$ a weight δ_j^i equal to the change in loss before and after training in the first 100 steps.
- RedPajama (Figure 5.15): we use Algorithm 3 and train for 1 billion tokens per data source. We assign each edge $i \to j$ a weight δ_j^i equal to the change in perplexity on the validation datalsoa before and after training.

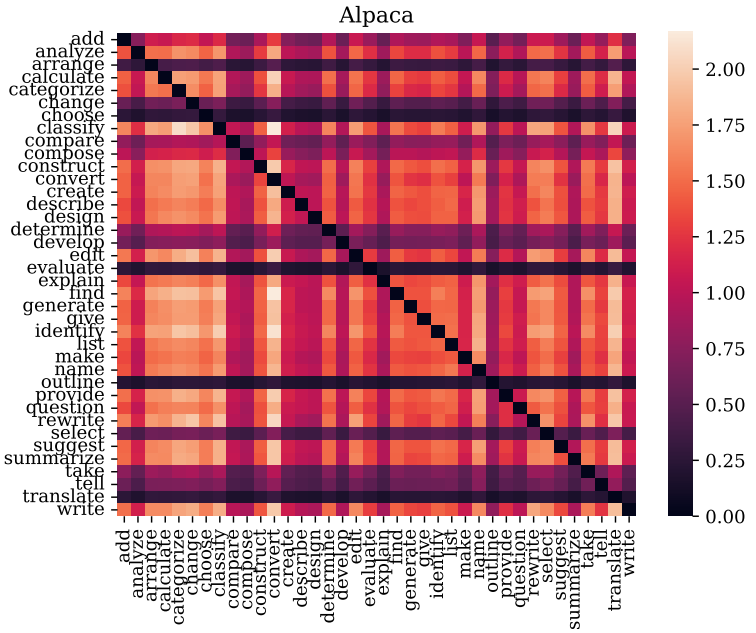


Figure 5.8: Alpaca heatmap where i, jth entry is $\max(0, \delta_j^i)$ (the change in loss on s_j after training on s_i for 150 steps). Diagonal entries are set to 0 for clearer visualization.

5.C.3 Training Details

We describe the parameters used for SKILL-IT.

SKILL-IT pre-training.

- LEGO: $\eta = 0.5, T = 6, w = 3$. We train for 6000 steps.
- Addition: $\eta = 0.1, T = 5, w = 3$. We train for 6000 steps.
- Natural Instructions (pre-training): $\eta = 0.2, T = 1$. We train for 5000 steps.

For the LEGO random baseline, when we selected points at random, we used an imbalanced training dataset with proportions 1:1:1:3:5. For the addition random baseline, we used an imbalanced dataset with randomly selected proportions: 13:14:18. For the curriculum learning baselines, the pacing function, $g(i)$, denotes the size of the subset of the highest scoring samples that we uniformly select from in the ith epoch. We define our pacing function as $g(i) = \frac{iH}{M}$, where H is the number of steps and M is 5 epochs for LEGO and NI, and 3 for addition.

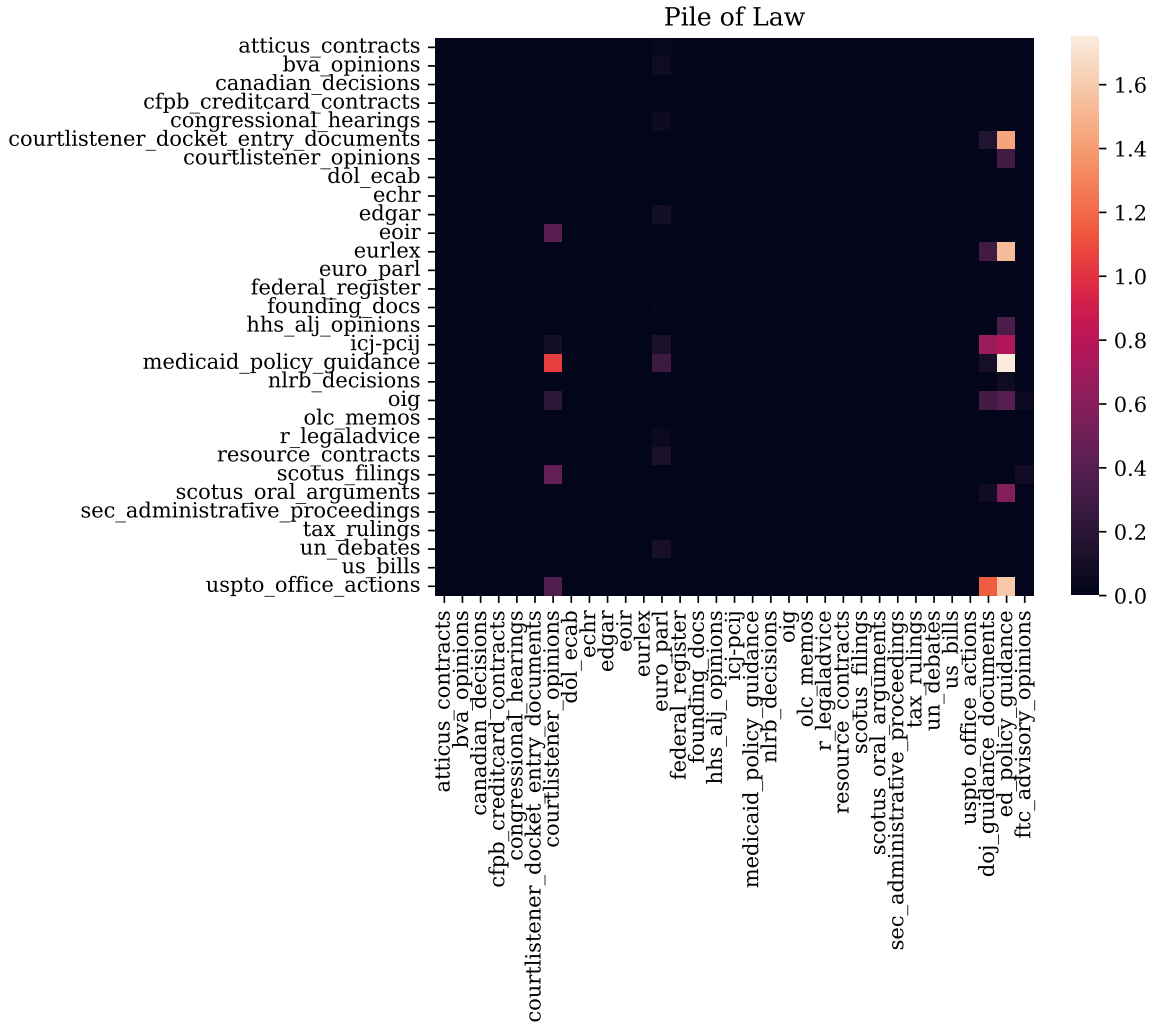


Figure 5.9: Pile of Law heatmap where i, jth entry is $\max(0, \delta_j^i)$ (the change in loss on s_j after training on s_i for 150 steps). Diagonal entries are set to 0 for clearer visualization.

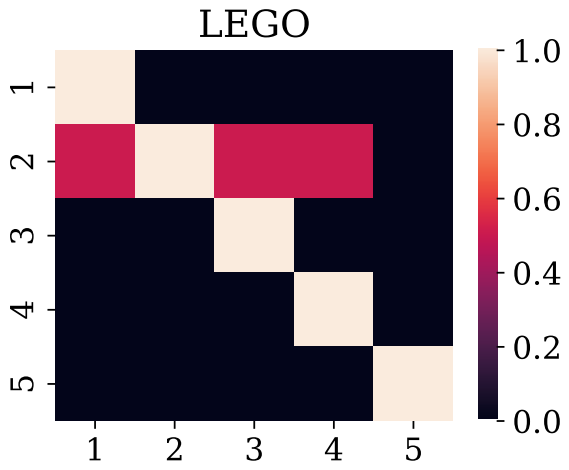


Figure 5.10: LEGO heatmap with $k = 5$ where i, jth entry is set to 0.5 if the number of steps needed to reach 0.01 loss on skill j when training on a balanced mixture of skills i and j is less than when training on skill j only.

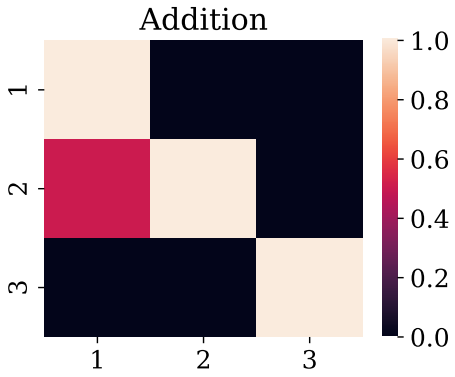


Figure 5.11: Addition heatmap with $k = 3$ where i, jth entry is set to 0.5 if the number of steps needed to reach 0.01 loss on skill j when training on a balanced mixture of skills i and j is less than when training on skill j only.

Figure 5.12: Natural Instructions heatmap where i, jth entry is $\max(0, \delta_j^i)$ (the change in loss on s_j after training on s_i for 100 steps). Diagonal entries are set to 0 for clearer visualization.

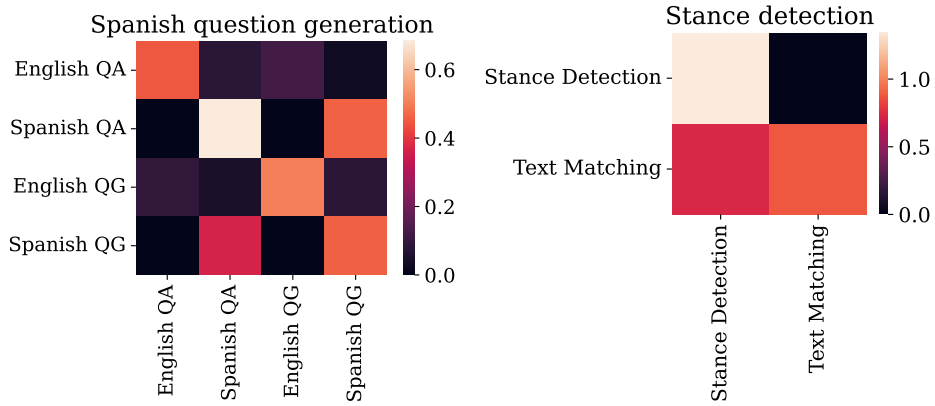


Figure 5.13: Spanish question generation and stance detection heatmaps where i, jth entry is $\max(0, \delta_j^i)$ (the change in loss on s_j after training on s_i for 100 steps).

Skill-it fine-tuning.

- LEGO: $\eta = 0.5, T = 10, w = 3$. We train for 6000 steps.
- Addition: $\eta = 0.1, T = 5, w = 3$. We train for 6000 steps.
- Natural Instructions (Spanish QG): $\eta = 0.8, T = 6, w = 3$. We train for 600 steps.
- Natural Instructions (stance detection): $\eta = 0.2, T = 6, w = 3$. We train for 600 steps.

Skill-it out-of-domain.

- Natural Instructions: $\eta = 0.2, T = 10, w = 3$. We train for 5000 steps.
- RedPajama: $\eta = 100, T = 1$. We train for 3 billion tokens.

All results are computed over 5 random seeds.

Batch sizes of 32 and 64 were used for the LEGO and addition synthetic on the 125M and 1.3B parameter model, respectively. Batch sizes of 4 and 16 were used for the Natural Instructions experiments on the 125M and 1.3B parameter model.

For the out-of-domain Natural Instructions experiment and Alpaca graph learning experiments, a learning rate of 5e-6 with linear scheduler and 50 warmup steps was used. For the Natural Instructions continual pre-training experiment on the 1.3B parameter model, a learning rate of 1e-6 was used. All other experiments used a learning rate of 5e-5. All experiments used AdamW with betas = 0.9, 0.999, eps = 1e-8, and weight decay = 0.01. A context window of 512 was used for all experiments except LEGO and addition, which used a window of 128.

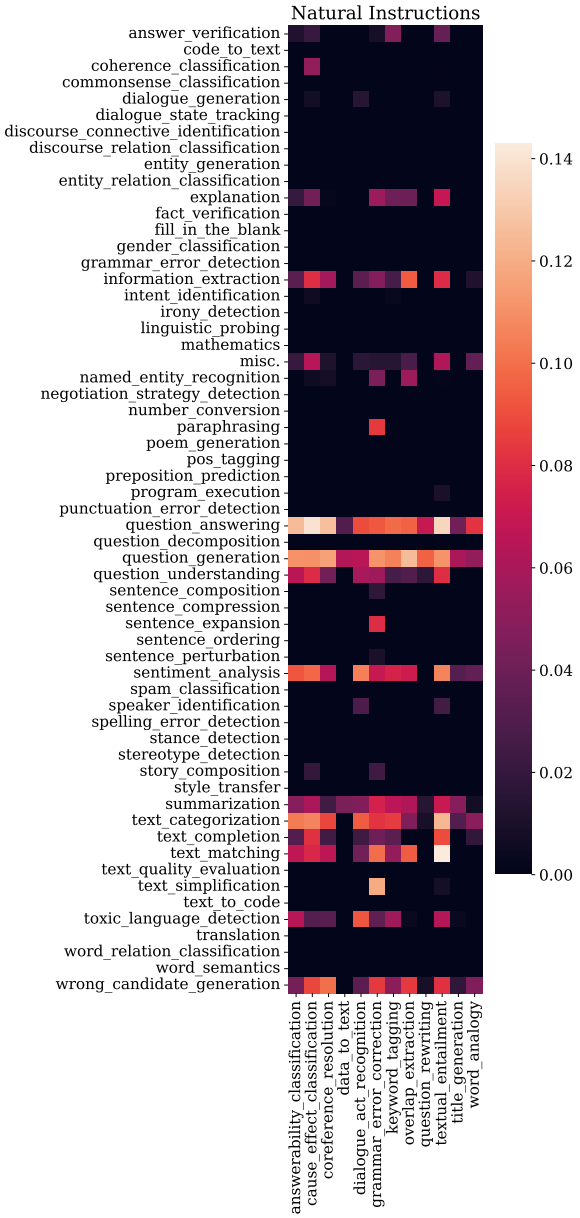


Figure 5.14: Natural Instructions heatmap for out-of-domain setting where rows are for the training skills and columns are for the evaluation skills. The i, jth entry is $\max(0, \delta_j^i)$ (the change in loss on s_j after training on s_i for 100 steps).

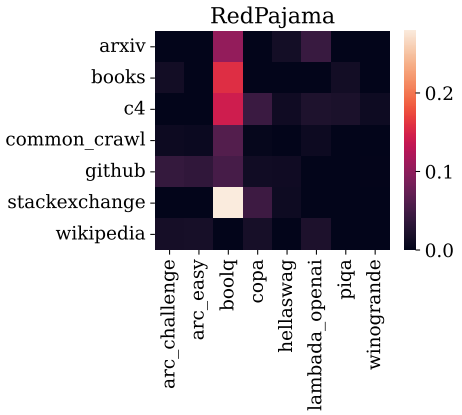


Figure 5.15: RedPajama heatmap for out-of-domain setting where rows are for the training skills and columns are for the evaluation skills. The i, jth entry is $\max(0, \delta_j^i)$ (the change in perplexity on s_j after training on s_i for 1B tokens).

Experiments with the Addition dataset were run using an Nvidia RTX A6000. Other experiments using the GPT-Neo 125M parameter model were run on an Nvidia Tesla P100. Experiments using the GPT-Neo 1.3B parameter model were run on an Nvidia Tesla A100.

5.D Additional Experimental Results

5.D.1 Additional examples of LEGO ordered skill sets

For the LEGO synthetic, it may appear obvious that the skills graph is equivalent to the reasoning chain over the variables. However, in Figure 5.16 we see that this is not the case. Training on skills 2 and 4 together results in lower loss on skill 4 than when trained on skill 4 alone. However, training on skills 3 and 4 together results in roughly the same loss on skill 4 as when training on skill 4 alone, even though skill 3 and skill 4 share an edge in the LEGO synthetic's underlying reasoning chain. This suggests that our intuition for how skills influence each other does not always match how the model learns skills.

Next, we consider a slightly more complex reasoning pattern on the LEGO synthetic. Instead of a chain, we construct a tree, where two variables in the LEGO synthetic are both defined in terms of the same parent variable. For example,

Input: c = val 1, y = not w, v = val c, w = not c. Output: y = 1.

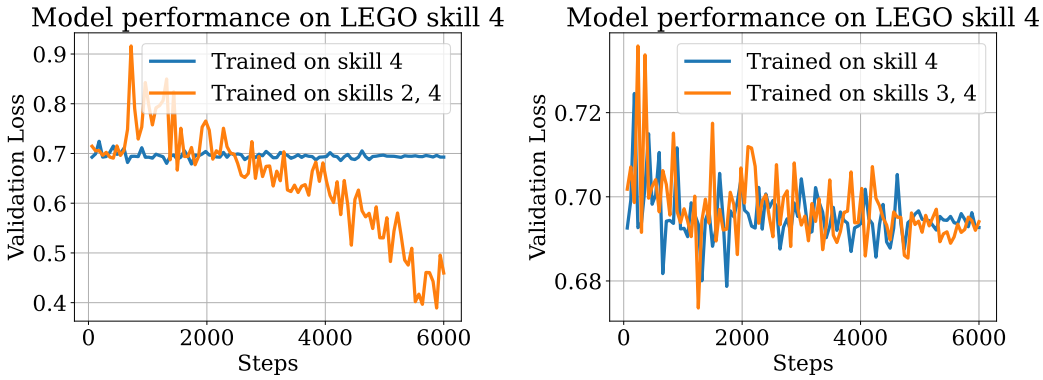


Figure 5.16: Performance on LEGO skill 4 when training on skill 4, skills 2 and 4, and skills 3 and 4. Even though skill 3 and skill 4 share an edge in the LEGO synthetic's underlying reasoning chain (i.e. a model predicting correct for the fourth variable is one extra step beyond predicting correct for the third variable), we find that training on skills 2 and 4 helps improve performance on skill 4 more.

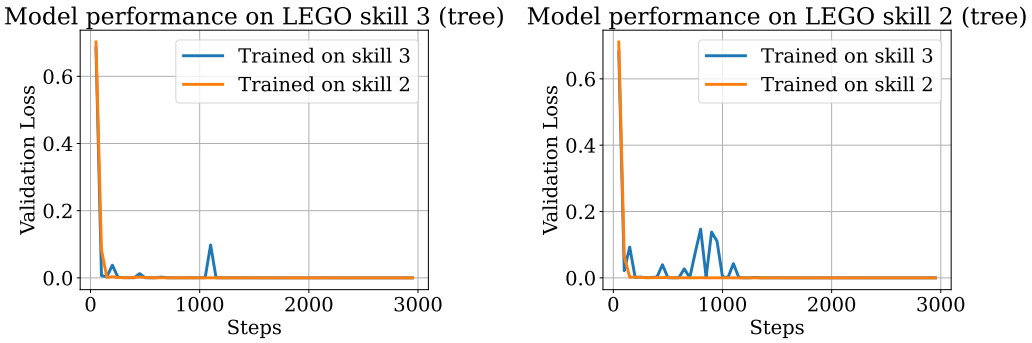


Figure 5.17: Performance on LEGO skill 2 and 3 when training on skills 2 and 3. The reasoning pattern is a tree rather than a chain over $k = 4$ variables. Skills 2 and 3 are at the same "depth" in the graph and both depend on skill 1, so there is positive influence between the skills despite there being no edge between 2 and 3 in the LEGO reasoning graph.

In this example, $k = 4$ and both ν and w are written in terms of c, and the reasoning graph has edges $1 \to 2$, $1 \to 3$, $2 \to 4$. In this case, we see that training on skill 2 or skill 3 both improve losses on skills 2 and 3 (Figure 5.17). However, unlike the previous figures, training on skills 2 and 4 or skills 3 and 4 do not significantly help reduce loss on skill 4 (Figure 5.18). Again, these measurements demonstrate that the reasoning graph does not necessarily equal the skills graph.

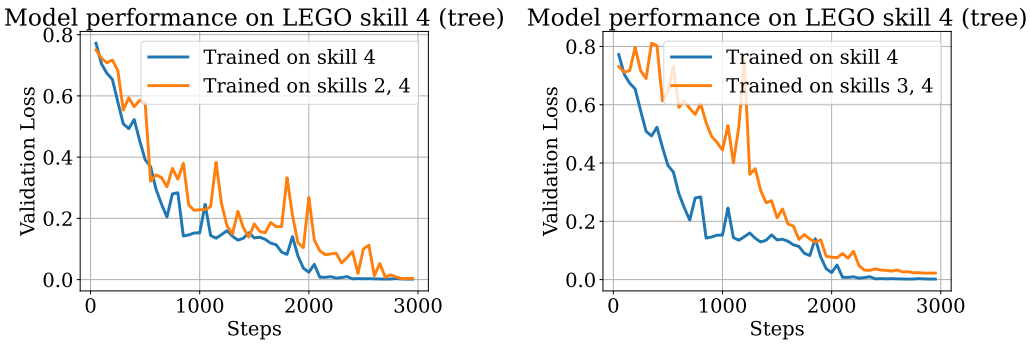


Figure 5.18: Performance on LEGO skill 4 when training on skills 2, 4 and skills 3, 4. We find that in both cases, the benefit from training on additional skills is minor. For instance, training on 2 and 4 reaches 0.01 loss in 2700 steps, while training on 4 only reaches it in 2100 steps.

5.D.2 Unsupervised skill recovery

We explore several clustering techniques for recovering the skills in the LEGO synthetic on the validation dataset. Our results are shown in Table 5.3.

We first cluster based on the pre-trained model embeddings of the last token and the average token. We also report accuracies of clustering based on the trained model embedding's last token, where we train the model using random sampling for 6000 steps, and clustering based on Sentence-BERT embeddings. Among these four methods, using the trained model embeddings has the highest accuracy of 38.4 points.

Next, we cluster points based on losses. In particular, we do 10 runs, each for 6000 steps and with a randomly sampled mixture of skills. For each run, we evaluate the model on the validation dataset at 120 checkpoints. Then, each sample in the validation dataset has 1200 losses associated with it, comprising a feature vector for that sample. We perform k-means clustering on these features, which has an accuracy of 61.0 points, significantly higher than the second best accuracy of 38.4.

5.D.3 Full results for Section 5.3

5.D.3.1 Per-skill performance

In this section, we provide tables containing the per skill break-down of our results from Section 5.3.

Table 5.3: Clustering-based skill recovery methods on the LEGO dataset. The validation dataset we cluster consists of 500 points with $k = 5$, and results are reported over 10 runs of k-means.

Cluster method	Accuracy
Pretrained embedding of last token	24.8 ± 0.5
Pretrained embedding of average token	25.2 ± 1.1
Trained model embedding of last token	38.4 ± 0.8
Sentence-BERT embedding	23.9 ± 0.7
Losses over multiple runs	$\mathbf{61.0 \pm 1.6}$

Continual Pre-training. In the continual pre-training setting, we report two additional baselines that combine curriculum learning with skills. Curriculum learning has been proposed for multitask learning [Varshney et al., 2022], in which groups of data are ranked by their average score and then trained in order of this ranking (with mixing of previously seen groups to avoid forgetting). We construct two baselines, Skill-curriculum and Skill-anticurriculum, using Algorithm 1 from [Varshney et al., 2022]. In contrast to the random baseline which has imbalanced skills, this approach has knowledge of skills and thus uses a skill-stratified training dataset to sample from. We set the fraction of the previous group to be frac $= 0.4$, as we found that setting frac $= 0.0$ resulted in forgetting.

We report loss per skill for the LEGO synthetic in Table 5.4, which corresponds to the results in Figure 5.4. We report accuracy per skill in Table 5.5 and Figure 5.19. We report the loss per skill for the Addition synthetic in Table 5.6, which also correspond to to the results in Figure 5.4. Finally, we report validation loss per task category for the Natural Instructions continual pre-training experiment in Table 5.7, where we find that Skill-it outperforms random sampling by 3.2% on average across skills.

Table 5.4: Results on validation loss per skill for LEGO pre-training experiment, averaged over 5 random seeds.

	Skill 1	Skill 2	Skill 3	Skill 4	Skill 5	Average
Random	$0_{\pm 0.000}$	$0.675_{\pm 0.041}$	$0.688_{\pm 0.008}$	$0.673_{\pm 0.049}$	$0.667_{\pm 0.056}$	$0.541_{\pm 0.031}$
Curriculum	$0_{\pm 0.000}$	$0.645_{\pm 0.052}$	$0.686_{\pm 0.018}$	$0.674_{\pm 0.042}$	$0.671_{\pm 0.0459}$	$0.535_{\pm 0.029}$
Anticurriculum	$0_{\pm 0.000}$	$0.690_{\pm 0.003}$	$0.695_{\pm 0.004}$	$0.693_{\pm 0.003}$	$0.689_{\pm 0.004}$	$0.554_{\pm 0.001}$
Skill-stratified	$0_{\pm 0.000}$	$0.045_{\pm 0.036}$	$0.056_{\pm 0.029}$	$0.079_{\pm 0.044}$	$0.050_{\pm 0.025}$	$0.046_{\pm 0.022}$
Skill-curriculum	$0_{\pm 0.000}$	$0.484_{\pm 0.200}$	$0.698_{\pm 0.027}$	$0.697_{\pm 0.010}$	$0.689_{\pm 0.007}$	$0.514_{\pm 0.040}$
Skill-anticurriculum	$0.001_{\pm 0.001}$	$0.174_{\pm 0.118}$	$0.245_{\pm 0.091}$	$0.443_{\pm 0.125}$	$0.566_{\pm 0.118}$	$0.286_{\pm 0.060}$
Skill-it	$0_{\pm 0.000}$	$0.002_{\pm 0.002}$	$0.024_{\pm 0.031}$	$0.013_{\pm 0.010}$	$0.022_{\pm 0.021}$	$0.012_{\pm 0.008}$

Table 5.5: Results on accuracy per skill (binary classification) for LEGO pre-training experiment, averaged over 5 random seeds.

	Skill 1	Skill 2	Skill 3	Skill 4	Skill 5	Average
Random	$100.0_{\pm 0.0}$	$54.2_{\pm 5.9}$	$58.0_{\pm 3.1}$	$48.0_{\pm 6.3}$	$54.4_{\pm 7.3}$	$62.9_{\pm 3.5}$
Curriculum	$100.0_{\pm 0.0}$	$60.0_{\pm 10.6}$	$55.2_{\pm 5.8}$	$51.2_{\pm 6.3}$	$51.8_{\pm 6.1}$	$63.6_{\pm 3.6}$
Anticurriculum	$100.0_{\pm 0.0}$	$53.4_{\pm 2.3}$	$49.0_{\pm 4.8}$	$48.2_{\pm 6.4}$	$56.0_{\pm 5.7}$	$61.3_{\pm 2.2}$
Skill-stratified	$100.0_{\pm 0.0}$	$98.2_{\pm 1.8}$	$98.2_{\pm 1.3}$	$97.8_{\pm 1.6}$	$98.2_{\pm 1.3}$	$98.5_{\pm 0.9}$
Skill-curriculum	$100.0_{\pm 0.0}$	$75.2_{\pm 30.1}$	$52.2_{\pm 3.7}$	$51.0_{\pm 4.6}$	$54.4_{\pm 3.1}$	$66.6_{\pm 7.7}$
Skill-anticurriculum	$100.0_{\pm 0.0}$	$90.2_{\pm 8.1}$	$88.2_{\pm 8.3}$	$73.2_{\pm 12.2}$	$62.4_{\pm 9.4}$	$82.8_{\pm 4.9}$
Skill-it	$100.0_{\pm 0.0}$	$99.2_{\pm 0.8}$	$99.0_{\pm 1.0}$	$99.4_{\pm 0.5}$	$99.6_{\pm 0.5}$	$99.4_{\pm 0.2}$

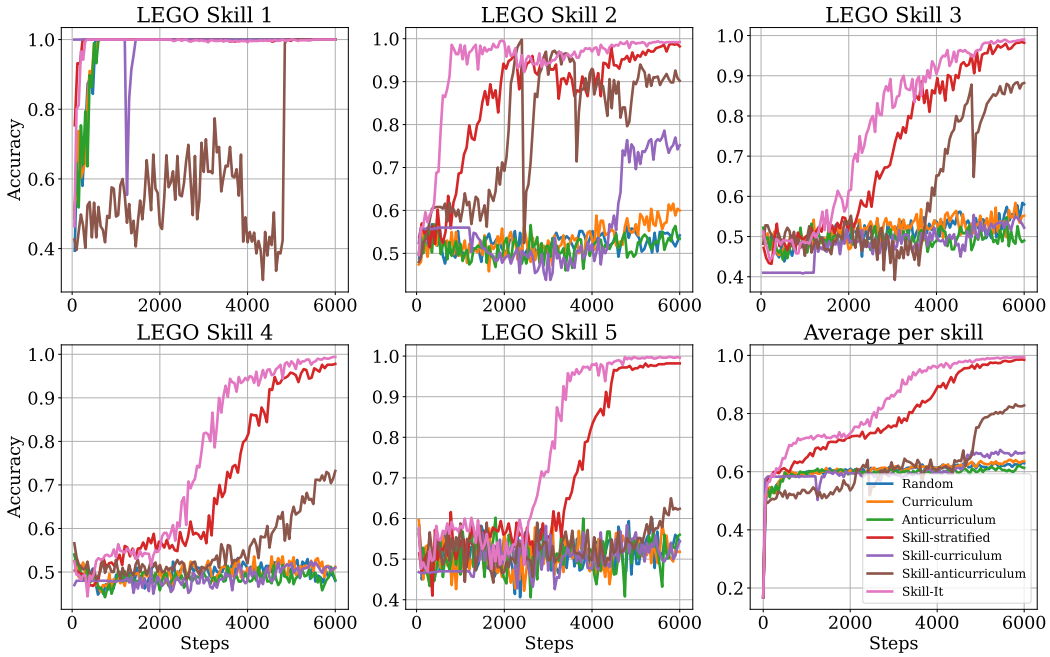


Figure 5.19: Accuracy of Skill-it on each skill (binary classification) on the LEGO synthetic in the continual pre-training setting. Skill-it attains higher accuracy more quickly than baselines that both do and do not utilize the notion of skills.

Out-of-domain. In Table 5.8, we provide a breakdown of validation loss per evaluation skill under random sampling on the training data, skill-stratified sampling over prerequisite skills (e.g., the nonzero rows in Figure 5.14), and Skill-it.

In Table 5.9 we provide a breakdown of the RedPajama experiment's accuracy per evaluation skill, corresponding to the results in Figure 5.7.

Table 5.6: Results on validation loss per skill for Addition pre-training experiment, averaged over 5 random seeds.

	Skill 1	Skill 2	Skill 3	Average
Random	$0.008_{\pm 0.007}$	$0.020_{\pm 0.019}$	$0.005_{\pm 0.005}$	$0.011_{\pm 0.014}$
Curriculum	$0.009_{\pm 0.011}$	$0.010_{\pm 0.008}$	$0.008_{\pm 0.010}$	$0.009_{\pm 0.010}$
Anticurriculum	$0.007_{\pm 0.010}$	$0.012_{\pm 0.013}$	$0.008_{\pm 0.017}$	$0.009_{\pm 0.014}$
Skill-stratified	$0.012_{\pm 0.011}$	$0.015_{\pm 0.015}$	$0.010_{\pm 0.020}$	$0.012_{\pm 0.016}$
Skill-curriculum	$0.016_{\pm 0.013}$	$0.019_{\pm 0.013}$	$0.010_{\pm 0.003}$	$0.015_{\pm 0.010}$
Skill-anticurriculum	$0.005_{\pm 0.008}$	$0.037_{\pm 0.028}$	$1.141_{\pm 1.126}$	$0.395_{\pm 0.371}$
Skill-it	$0.004_{\pm 0.003}$	$0.009_{\pm 0.007}$	$0.013_{\pm 0.017}$	$0.009_{\pm 0.011}$

5.D.3.2 Weight trajectories

We provide Skill-it's weight trajectories for each result. The weight per skill across training steps for the LEGO pre-training experiment corresponding to Figure 5.4 (left) is shown in Figure 5.20. We see that Skill-it initially allocates more weight to skill 2 and less to $1, 3, 4, 5$. Since skill 1 is learned quickly, the weight on skill 1 immediately drops to below 0.1 at 1000 steps. The weight on skills $3, 4$, and 5 increase from around 0 to 3000 steps, during which their respective validation losses are higher than those of skills 1 and 2. Near the end of training, all losses are converging to 0, and so the weight per skill is roughly uniform.

The weight per skill across training steps for the addition pre-training experiment corresponding to Figure 5.4 (right) is shown in Figure 5.21. Skill-it allocates more weight to skill 2, which has an edge to skill 1 as shown in Figure 5.11. It also allocates very little weight to skill 3, which is learned faster than the other two skills. Eventually, it puts more weight on skill 1, the hardest skill, and then converges to uniform sampling as all validation losses approach 0.

The weight per skill across training steps for the LEGO fine-tuning experiment and the Spanish question generation and stance detection experiments corresponding to Figure 5.5 is shown in Figure 5.22. Since there is only one target skill in these experiments, the mixture of weights approaches uniform as the loss on the target skill approaches 0. It is interesting to explore how to reduce edge weights and regularization so that the mixture approaches the target skill instead, although preliminary experiments where we decayed the edge weight and the strength of the Bregman divergence term did not appear better. We hypothesize that since training on a uniform mixture (as in Figure 5.3) did strictly better than training on the target skill and their loss curves did not intersect during the training run, it is better

Table 5.7: Validation loss per skill for data selection in continual pre-training setting on a subset of the Natural Instructions Dataset.

Skill	Random	Curriculum	Anticurriculum	Skill-stratified	Skill-curriculum	Skill-anticurriculum	SKILL-IT
Answer Verification	$2.297_{\pm 0.058}$	$2.368_{\pm 0.055}$	$2.391_{\pm 0.061}$	$2.180_{\pm 0.059}$	$2.249_{\pm 0.116}$	$2.325_{\pm 0.085}$	$2.158_{\pm 0.059}$
Code to Text	$0.246_{\pm 0.021}$	$0.203_{\pm 0.019}$	$1.099_{\pm 0.115}$	$0.178_{\pm 0.016}$	$0.126_{\pm 0.009}$	$1.232_{\pm 0.070}$	$0.223_{\pm 0.017}$
Discourse Connective Identification	$2.927_{\pm 0.069}$	$3.084_{\pm 0.067}$	$2.932_{\pm 0.058}$	$2.805_{\pm 0.071}$	$2.891_{\pm 0.001}$	$2.925_{\pm 0.011}$	$2.784_{\pm 0.068}$
Entity Generation	$2.033_{\pm 0.421}$	$2.012_{\pm 0.437}$	$2.363_{\pm 0.234}$	$1.803_{\pm 0.384}$	$1.853_{\pm 0.483}$	$2.068_{\pm 0.719}$	$1.863_{\pm 0.418}$
Entity Relation Classification	$1.020_{\pm 0.147}$	$1.014_{\pm 0.140}$	$1.533_{\pm 0.138}$	$0.859_{\pm 0.131}$	$0.825_{\pm 0.022}$	$0.959_{\pm 0.009}$	$0.908_{\pm 0.146}$
Information Extraction	$2.154_{\pm 0.040}$	$2.247_{\pm 0.037}$	$2.352_{\pm 0.042}$	$2.140_{\pm 0.037}$	$2.286_{\pm 0.022}$	$2.338_{\pm 0.025}$	$2.073_{\pm 0.042}$
Irony Detection	$3.024_{\pm 0.154}$	$3.798_{\pm 0.095}$	$2.942_{\pm 0.158}$	$2.680_{\pm 0.146}$	$3.889_{\pm 0.066}$	$2.099_{\pm 0.152}$	$2.797_{\pm 0.155}$
Preposition Prediction	$0.979_{\pm 0.124}$	$0.887_{\pm 0.147}$	$1.488_{\pm 0.213}$	$0.845_{\pm 0.152}$	$0.941_{\pm 0.019}$	$1.044_{\pm 0.029}$	$0.876_{\pm 0.173}$
Punctuation Error Detection	$2.950_{\pm 0.065}$	$3.120_{\pm 0.052}$	$2.961_{\pm 0.064}$	$3.264_{\pm 0.061}$	$3.019_{\pm 0.010}$	$3.360_{\pm 0.013}$	$3.216_{\pm 0.055}$
Question Answering	$2.277_{\pm 0.005}$	$2.367_{\pm 0.006}$	$2.398_{\pm 0.006}$	$2.542_{\pm 0.004}$	$2.689_{\pm 0.001}$	$2.707_{\pm 0.016}$	$2.448_{\pm 0.008}$
Question Generation	$2.617_{\pm 0.005}$	$2.777_{\pm 0.015}$	$2.695_{\pm 0.008}$	$2.783_{\pm 0.021}$	$3.062_{\pm 0.006}$	$2.876_{\pm 0.032}$	$2.666_{\pm 0.012}$
Question Understanding	$1.965_{\pm 0.051}$	$2.199_{\pm 0.059}$	$2.060_{\pm 0.033}$	$1.958_{\pm 0.051}$	$2.385_{\pm 0.022}$	$2.100_{\pm 0.054}$	$1.895_{\pm 0.043}$
Sentence Expansion	$2.501_{\pm 0.095}$	$2.598_{\pm 0.097}$	$2.583_{\pm 0.074}$	$2.225_{\pm 0.095}$	$2.311_{\pm 0.076}$	$2.408_{\pm 0.074}$	$2.236_{\pm 0.083}$
Sentiment Analysis	$3.203_{\pm 0.012}$	$3.415_{\pm 0.016}$	$3.209_{\pm 0.010}$	$3.278_{\pm 0.014}$	$3.607_{\pm 0.012}$	$3.308_{\pm 0.015}$	$3.213_{\pm 0.012}$
Stance Detection	$1.810_{\pm 0.100}$	$1.775_{\pm 0.120}$	$2.231_{\pm 0.128}$	$1.385_{\pm 0.070}$	$1.361_{\pm 0.114}$	$1.823_{\pm 0.189}$	$1.556_{\pm 0.125}$
Summarization	$2.961_{\pm 0.015}$	$3.149_{\pm 0.023}$	$3.041_{\pm 0.014}$	$2.960_{\pm 0.019}$	$3.323_{\pm 0.028}$	$3.021_{\pm 0.013}$	$2.907_{\pm 0.012}$
Text Categorization	$2.488_{\pm 0.023}$	$2.692_{\pm 0.029}$	$2.553_{\pm 0.006}$	$2.570_{\pm 0.015}$	$3.001_{\pm 0.007}$	$2.635_{\pm 0.014}$	$2.448_{\pm 0.017}$
Text Matching	$2.177_{\pm 0.059}$	$2.232_{\pm 0.055}$	$2.316_{\pm 0.048}$	$2.152_{\pm 0.061}$	$2.324_{\pm 0.004}$	$2.304_{\pm 0.035}$	$2.093_{\pm 0.054}$
Text Simplification	$2.155_{\pm 0.023}$	$2.193_{\pm 0.039}$	$2.325_{\pm 0.033}$	$1.926_{\pm 0.026}$	$2.037_{\pm 0.005}$	$2.156_{\pm 0.011}$	$1.952_{\pm 0.026}$
Text to Code	$0.560_{\pm 0.037}$	$0.495_{\pm 0.036}$	$1.215_{\pm 0.052}$	$0.490_{\pm 0.029}$	$0.433_{\pm 0.014}$	$1.455_{\pm 0.086}$	$0.553_{\pm 0.042}$
Toxic Language Detection	$3.106_{\pm 0.027}$	$3.496_{\pm 0.017}$	$3.058_{\pm 0.029}$	$3.199_{\pm 0.024}$	$3.758_{\pm 0.025}$	$3.155_{\pm 0.050}$	$3.129_{\pm 0.020}$
Word Semantics	$2.092_{\pm 0.027}$	$2.334_{\pm 0.034}$	$2.156_{\pm 0.064}$	$1.916_{\pm 0.043}$	$1.784_{\pm 0.048}$	$2.424_{\pm 0.038}$	$1.952_{\pm 0.019}$
Wrong Candidate Generation	$2.438_{\pm 0.021}$	$2.606_{\pm 0.039}$	$2.519_{\pm 0.027}$	$2.506_{\pm 0.026}$	$2.849_{\pm 0.029}$	$2.574_{\pm 0.018}$	$2.432_{\pm 0.025}$
Average	$2.173_{\pm 0.028}$	$2.307_{\pm 0.025}$	$2.366_{\pm 0.026}$	$2.115_{\pm 0.027}$	$2.304_{\pm 0.031}$	$2.317_{\pm 0.052}$	$2.103_{\pm 0.032}$

Table 5.8: Validation loss per skill for data selection in out-of-domain setting over Natural Instructions train task split and test task split.

Skill	Random	Skill-stratified	SKILL-IT
Answerability Classification	$3.048_{\pm 0.003}$	$3.076_{\pm 0.002}$	$3.043_{\pm 0.003}$
Cause Effect Classification	$2.068_{\pm 0.004}$	$2.101_{\pm 0.005}$	$2.067_{\pm 0.006}$
Coreference Resolution	$3.101_{\pm 0.003}$	$3.142_{\pm 0.004}$	$3.099_{\pm 0.004}$
Data to Text	$2.363_{\pm 0.004}$	$2.388_{\pm 0.005}$	$2.359_{\pm 0.005}$
Dialogue Act Recognition	$2.329_{\pm 0.009}$	$2.364_{\pm 0.010}$	$2.320_{\pm 0.009}$
Grammar Error Correction	$2.399_{\pm 0.008}$	$2.418_{\pm 0.009}$	$2.389_{\pm 0.007}$
Keyword Tagging	$2.744_{\pm 0.005}$	$2.760_{\pm 0.007}$	$2.733_{\pm 0.006}$
Overlap Extraction	$2.749_{\pm 0.011}$	$2.763_{\pm 0.012}$	$2.733_{\pm 0.010}$
Question Rewriting	$2.591_{\pm 0.009}$	$2.628_{\pm 0.011}$	$2.586_{\pm 0.010}$
Textual Entailment	$2.472_{\pm 0.002}$	$2.503_{\pm 0.003}$	$2.468_{\pm 0.002}$
Title Generation	$3.027_{\pm 0.002}$	$3.037_{\pm 0.002}$	$3.015_{\pm 0.002}$
Word Analogy	$1.665_{\pm 0.016}$	$1.682_{\pm 0.015}$	$1.668_{\pm 0.016}$
Average	$2.546_{\pm 0.003}$	$2.572_{\pm 0.003}$	$2.540_{\pm 0.003}$

Table 5.9: Performance of model trained on RedPajama with uniform sampling and SKILL-IT on LM evaluation harness. Unless otherwise noted, accuracy is reported for each task.

	1 Billion Tokens		2 Billion Tokens		3 Billion Tokens	
	Uniform	SKILL-IT	Uniform	SKILL-IT	Uniform	SKILL-IT
ARC Challenge (acc norm)	35.4	34.6	35.3	34.9	34.6	34.8
ARC Easy (acc norm)	62.2	61.2	62.4	61.7	62.5	62.0
BoolQ	68.9	68.2	67.7	68.6	67.2	68.7
COPA	81.0	82.0	80.0	81.0	81.0	81.0
HellaSwag (acc norm)	63.9	63.7	63.8	63.9	64.0	63.9
LAMBADA OpenAI	64.4	67.0	65.9	66.7	66.8	66.0
PIQA (acc norm)	74.8	75.0	75.5	75.2	75.0	75.7
Winogrande	62.8	63.9	63.9	63.2	63.4	63.1
Average accuracy	64.2	64.4	64.3	64.4	64.3	64.4

to allocate non-negligible weight on all skills throughout the training run.

The weight per skill across training steps for the Natural Instructions out-of-domain experiment corresponding to Figure 5.6 is shown in Figure 5.23, where the legend is provided for the top 10 task categories with the largest weights. While the initial weights based on

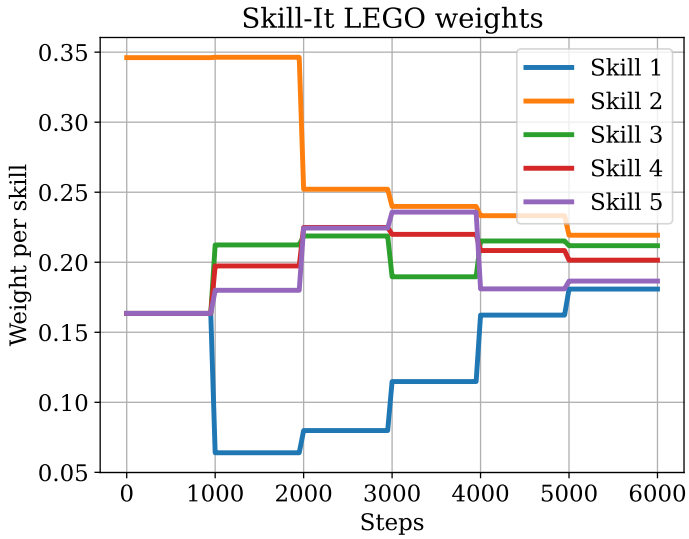


Figure 5.20: Weight per skill for LEGO pre-training experiment. Skill-it initially allocates more weight to skill 2, but eventually puts more weight on harder skills $(3, 4, 5)$ before converging to uniform sampling when all losses converge roughly to 0.

the skills graph roughly establishes the order of weight magnitude, the differences among the losses on the evaluation skills increases the range of weights as training continues. As validation losses saturate, the weights also converge to fixed values.

5.D.4 Experiments on 1.3B parameter model

We demonstrate that the skills graph learned on the 125M parameter model can be used for data selection with the GPT-Neo-1.3B model. We present results in the continual pre-training setting on the LEGO synthetic and Natural Instructions.

All results are reported over 3 random seeds. For the LEGO experiment, we train for 1500 steps with $\eta = 0.5, \mathsf{T} = 30, w = 3$. For the NI experiment, we train for 5000 steps with $\eta = 0.2$, and $\mathsf{T} = 1$. The skill graphs were learned using the 125M parameter model as described in section 5.C.2.

In Figure 5.24, we train the 1.3B model using Skill-it for the LEGO synthetic and find that it still outperforms random and skill-stratified sampling on average. In particular, while performance across sampling methods is similar for early skills, the discrepancy is larger for skill 5, for which Skill-it allocates more weight to dynamically. In Figure 5.25, we provide the weight trajectories of Skill-it. We observe that the weight trajectories are similar to

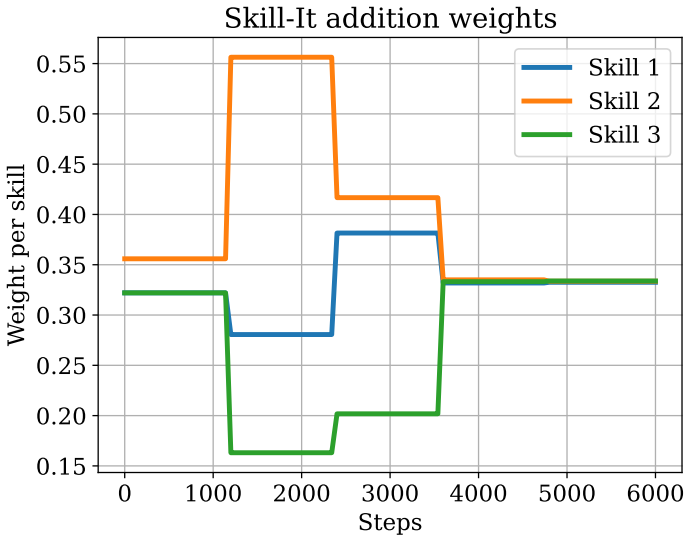


Figure 5.21: Weight per skill for addition pre-training experiment. SKILL-IT initially allocates more weight to skill 2, which has an edge to skill 1, while allocating little weight to skill 3 which is learned quickly. Eventually, SKILL-IT puts more weight on the harder skill 1 before converging to uniform sampling when all losses roughly approach 0.

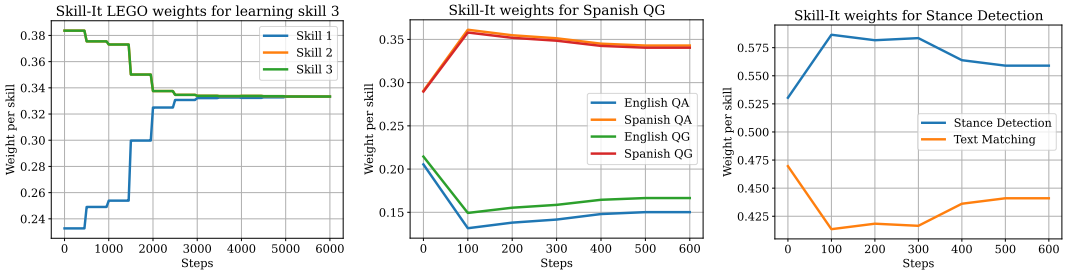


Figure 5.22: Weight per skill for fine-tuning experiments. Left: LEGO; Center: Spanish question generation; Right: stance detection.

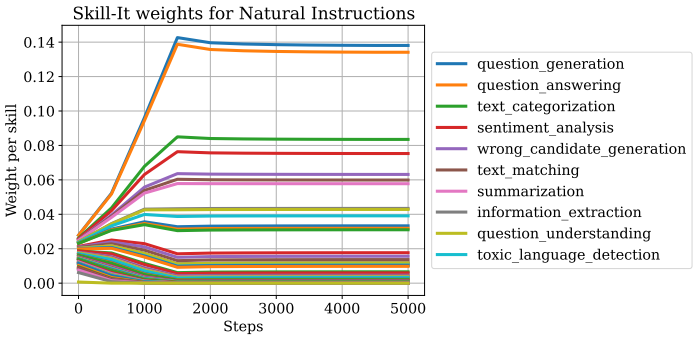


Figure 5.23: Weight per skill for Natural Instructions out-of-domain experiment. The legend shows the top 10 skills with the largest weight. While the relative order of weight magnitude does not change significantly across training, the incorporation of loss dramatically increases the range of the weights, showing the importance of an online algorithm.

that on the 125M parameter model, where initial weight is allocated towards skill 2. Later on, more weight is allocated towards skills 4 and 5, whose losses are higher, and eventually the weight mixture converges to uniform as all losses converge to near 0.

In Table 5.10, we report performance of Skill-it with the 1.3B model on the Natural Instructions pre-training experiment and find that the trends from the smaller model hold—Skill-it outperforms random and skill-stratified sampling on average.

5.D.5 Ablations

We report ablations on the skills graph and the online component of Skill-it. Instead of using A in Algorithm 1, we study the performance when the identity matrix is used instead; intuitively, this corresponds to a misspecified skills graph where no skill influences another skill. We refer this approach as "No graph". Note that the opposite case of a complete graph recovers skill-stratified sampling, which we already have as a baseline.

Second, instead of sampling over multiple rounds and weighting according to the loss of each skill, we study the effect of setting $T = 1$, which only uses a softmax on A to yield static weights on the skills. We refer to this approach as "Static". We omit results on Natural Instructions continual pre-training, since Skill-it uses $T = 1$ and using no graph with $T = 1$ recovers skill-stratified sampling. Intuitively, we expect the static version of Skill-it to perform somewhat well unless there is significant discrepancy among the losses (e.g. in synthetics where the loss on one skill can be close to 0 while the other is not, versus in

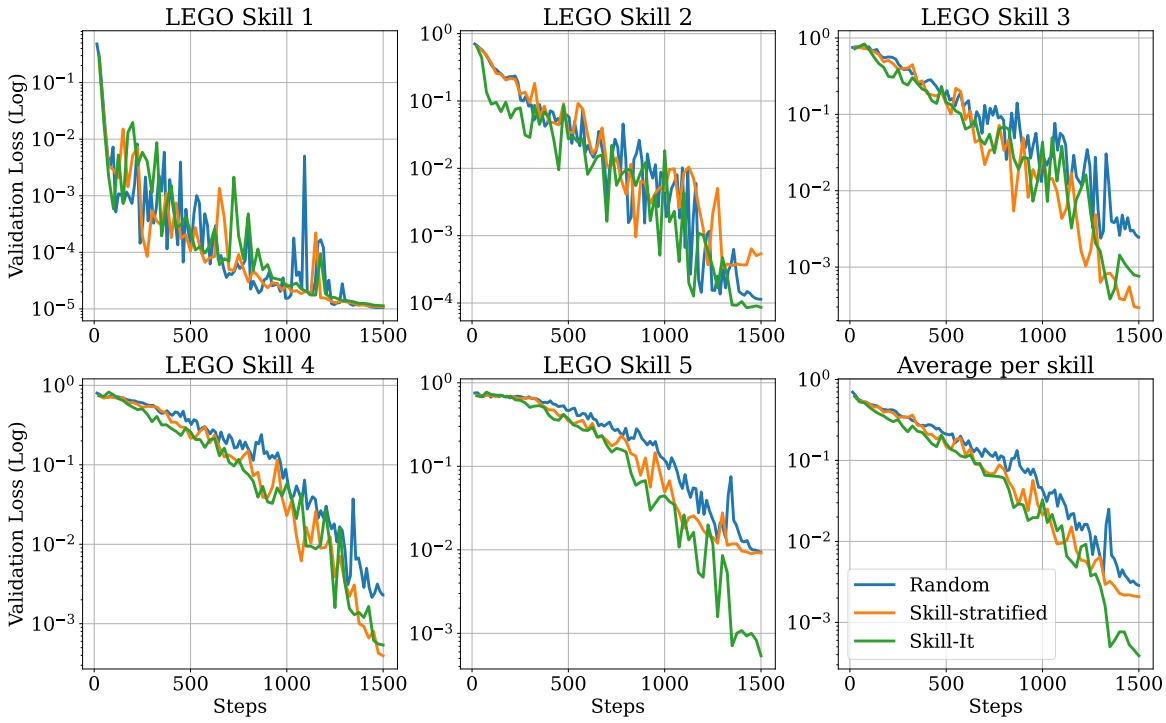


Figure 5.24: Performance of Skill-it for LEGO pre-training setting when skills graph is learned on a 125M parameter model and used for data selection with a 1.3B model. Skill-it on average still outperforms random and skill-stratified sampling, suggesting that findings on ordered skill sets can transfer from small models to large models.

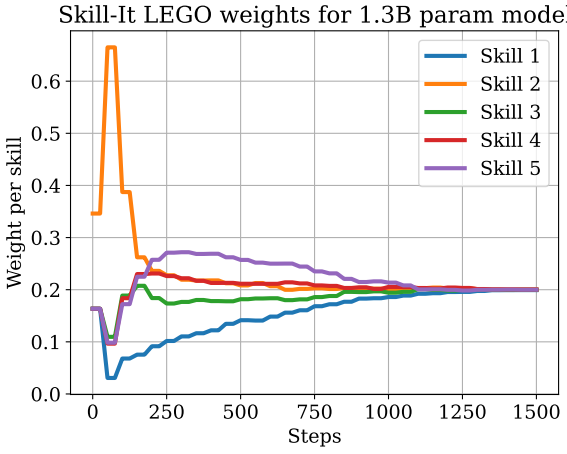


Figure 5.25: Weight per skill for LEGO pre-training experiment on 1.3B parameter model. The trajectories are similar to those of the 125M parameter model in Figure 5.20. Skill-it initially allocates more weight to skill 2, but eventually puts more weight on skills 4 and 5 before converging to uniform sampling when all losses converge to near 0.

Table 5.10: Results when skills graph for Natural Instructions learned on 125M parameter model is used for data selection with a 1.3B model. We see that SKILL-IT on average still outperforms random and skill-stratified sampling, even though the edges used by SKILL-IT are not derived from the larger model.

Skill	Random	Skill-stratified	SKILL-IT
Answer Verification	$2.005_{\pm 0.059}$	$1.903_{\pm 0.069}$	$1.890_{\pm 0.072}$
Code to Text	$0.302_{\pm 0.032}$	$0.204_{\pm 0.022}$	$0.269_{\pm 0.032}$
Discourse Connective Identification	$2.529_{\pm 0.046}$	$2.372_{\pm 0.054}$	$2.393_{\pm 0.056}$
Entity Generation	$2.108_{\pm 0.328}$	$1.788_{\pm 0.429}$	$1.885_{\pm 0.461}$
Entity Relation Classification	$1.130_{\pm 0.048}$	$0.836_{\pm 0.006}$	$0.841_{\pm 0.010}$
Information Extraction	$2.032_{\pm 0.013}$	$1.992_{\pm 0.006}$	$1.933_{\pm 0.013}$
Irony Detection	$2.802_{\pm 0.125}$	$2.528_{\pm 0.146}$	$2.585_{\pm 0.149}$
Preposition Prediction	$1.095_{\pm 0.040}$	$0.686_{\pm 0.041}$	$0.774_{\pm 0.029}$
Punctuation Error Detection	$2.633_{\pm 0.027}$	$3.188_{\pm 0.055}$	$2.726_{\pm 0.025}$
Question Answering	$1.947_{\pm 0.003}$	$2.119_{\pm 0.003}$	$2.073_{\pm 0.001}$
Question Generation	$2.214_{\pm 0.007}$	$2.345_{\pm 0.008}$	$2.263_{\pm 0.010}$
Question Understanding	$1.928_{\pm 0.020}$	$1.837_{\pm 0.031}$	$1.700_{\pm 0.042}$
Sentence Expansion	$2.054_{\pm 0.018}$	$1.828_{\pm 0.060}$	$1.853_{\pm 0.058}$
Sentiment Analysis	$2.771_{\pm 0.009}$	$2.818_{\pm 0.006}$	$2.774_{\pm 0.007}$
Stance Detection	$1.814_{\pm 0.151}$	$1.500_{\pm 0.117}$	$1.628_{\pm 0.149}$
Summarization	$2.531_{\pm 0.009}$	$2.472_{\pm 0.012}$	$2.440_{\pm 0.013}$
Text Categorization	$2.289_{\pm 0.016}$	$2.341_{\pm 0.021}$	$2.231_{\pm 0.022}$
Text Matching	$1.967_{\pm 0.008}$	$1.913_{\pm 0.005}$	$1.872_{\pm 0.005}$
Text Simplification	$1.861_{\pm 0.003}$	$1.692_{\pm 0.023}$	$1.698_{\pm 0.022}$
Text to Code	$0.614_{\pm 0.030}$	$0.518_{\pm 0.030}$	$0.585_{\pm 0.022}$
Toxic Language Detection	$2.853_{\pm 0.020}$	$2.911_{\pm 0.019}$	$2.862_{\pm 0.018}$
Word Semantics	$1.999_{\pm 0.023}$	$1.870_{\pm 0.039}$	$1.902_{\pm 0.024}$
Wrong Candidate Generation	$2.187_{\pm 0.028}$	$2.192_{\pm 0.023}$	$2.140_{\pm 0.020}$
Average	$1.985_{\pm 0.022}$	$1.907_{\pm 0.027}$	$1.883_{\pm 0.032}$

Natural Instructions where all losses decrease consistently). For both ablations, we sweep over values of $\eta = [0.1, 0.2, 0.5, 0.8]$.

Figure 5.26 shows the comparison between SKILL-IT and no graph on the continual pre-training LEGO experiment, and Figure 5.27 shows the comparison between SKILL-IT and a static approach. We see that both the graph and the online dynamics of SKILL-IT are important for its performance. In particular, using no graph results in allocating significant weight to harder skills early on, even though many of them have easier prerequisite skills (such as skill 3 having edges to skills 1 and 2). Using a static graph results in consistent allocation of significant weight to prerequisite skills even after their validation losses converge to near 0, and thus the harder skills that have higher loss are not learned quickly

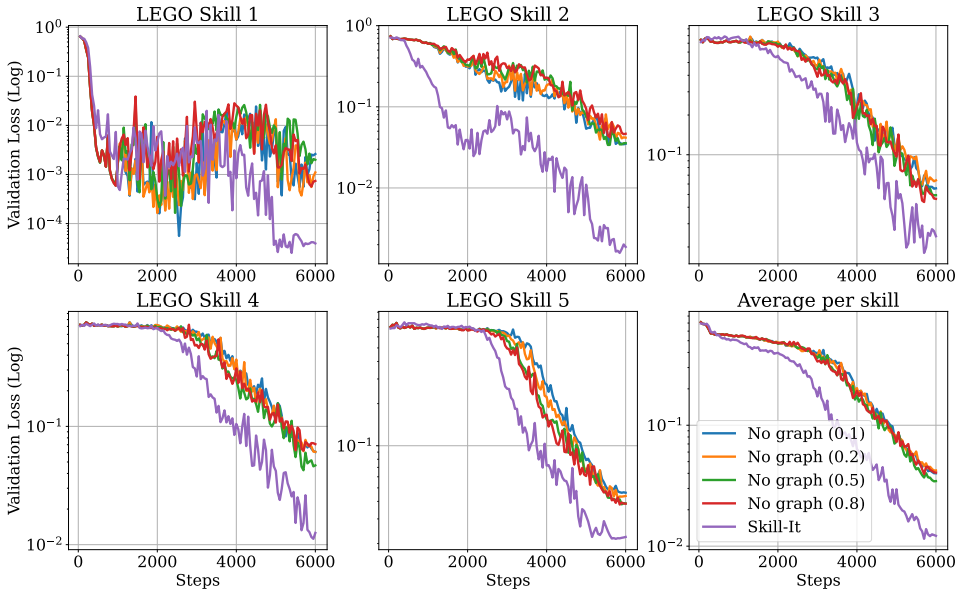


Figure 5.26: Comparison of Skill-it versus using the identity adjacency matrix (no skills graph) with $\eta = 0.1, 0.2, 0.5, 0.8$ on the LEGO continual pre-training experiment. The latter does not capture the relationship between skills, and we find that Skill-it attains lower loss on all skills.

afterwards.

We perform the same ablation on the Addition dataset—the results for this are shown in Figures 5.28 and Figure 5.29. We find that these simple baselines, including using a static graph and no graph perform similarly to Skill-it on average across all skills—while Skill-it performs the best on skill 2 compared to vanilla multiplicative weights, and Skill-it performs the best on skill 1 compared to a static graph. This suggests that Addition is somewhat easier than the other datasets that we consider, as Skill-it still outperforms other baselines, as shown in Figure 5.4.

Figure 5.30 compares Skill-it, no graph, and static data selection for the LEGO fine-tuning experiment. No graph can be interpreted as allocating equal weight to all training skills not equal to the target skill, and varying this weight versus the weight on the target skill. While Skill-it and setting $\mathsf{T} = 1$ behave similarly, we see that Skill-it is slightly better than using no graph. For instance, Skill-it obtains a validation loss of 0.05 in 2000 steps, compared to 2050-2200 steps when using no graph.

Figure 5.31 and 5.32 compare Skill-it, no graph, and static data selection for the Natural Instructions fine-tuning experiments. For both Spanish QG and stance detection, Skill-it

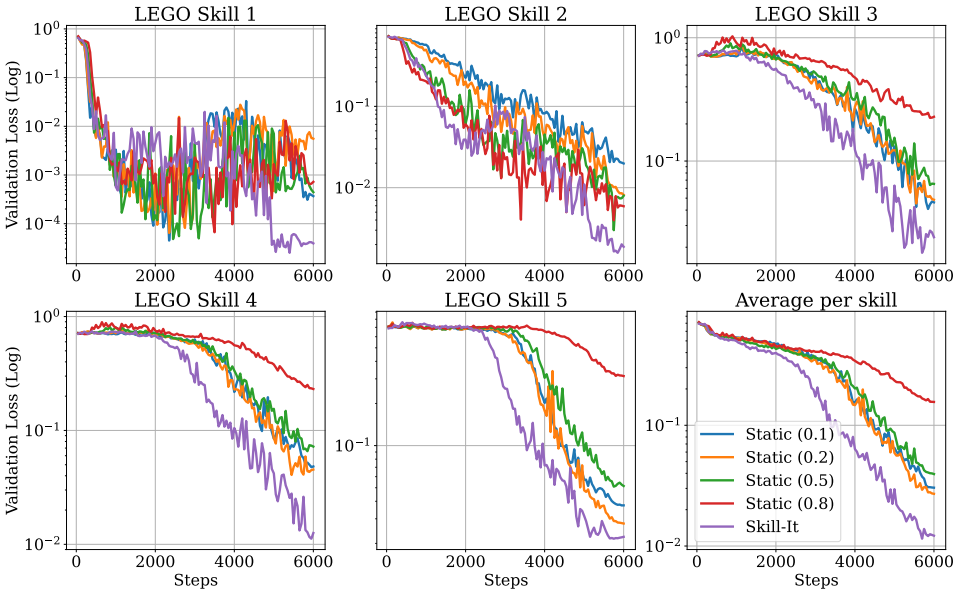


Figure 5.27: Comparison of Skill-it versus using static data selection ($T = 1$) with $\eta = 0.1, 0.2, 0.5, 0.8$ on the LEGO continual pre-training experiment. While Skill-it eventually allocates more weights to skills 3, 4, 5, which have higher loss, the static approach is not able to do this. We find that Skill-it attains lower loss on all skills.

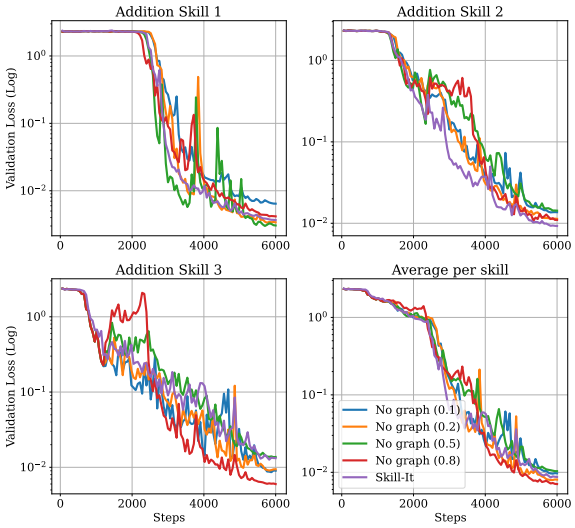


Figure 5.28: Comparison of Skill-it versus using the identity adjacency matrix (no skills graph) with $\eta = 0.1, 0.2, 0.5, 0.8$ on the Addition continual pre-training experiment. The latter does not capture the relationship between skills, and we find that Skill-it attains lower loss on skill 2, but attains similar performance to methods that do not use the skills graph.

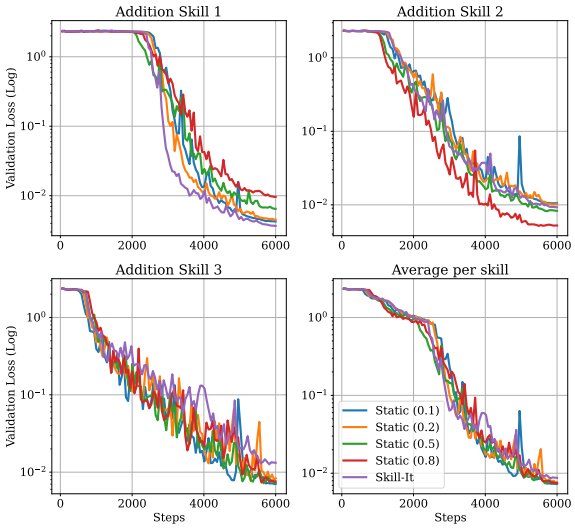


Figure 5.29: Comparison of Skill-it versus using static data selection ($T = 1$) with $\eta = 0.1, 0.2, 0.5, 0.8$ on the Addition continual pre-training experiment. We find that Skill-it attains lower loss on skill 1, but attains similar performance to the static methods.

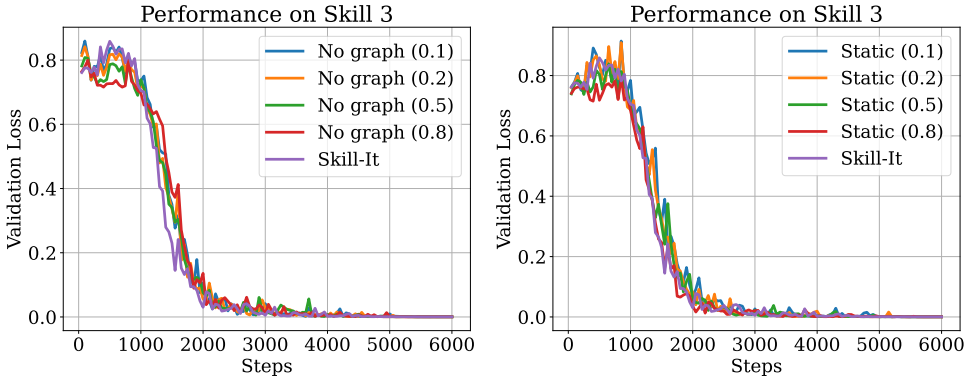


Figure 5.30: Comparison of Skill-it versus using no graph (left) and static data selection (right) with $\eta = 0.1, 0.2, 0.5, 0.8$ on the LEGO fine-tuning experiment. All approaches have roughly the same loss trajectories, but Skill-it is slightly lower than using no graph.

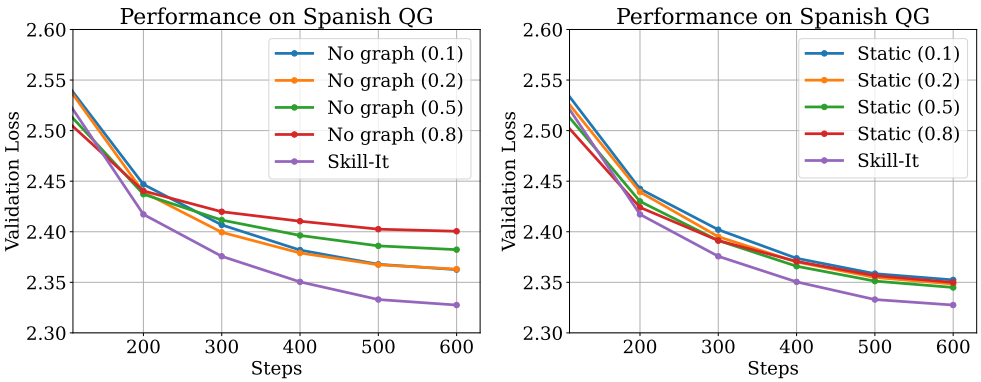


Figure 5.31: Comparison of SKILL-IT versus using no graph (left) and static data selection (right) with $\eta = 0.1, 0.2, 0.5, 0.8$ on the Natural Instructions Spanish QG fine-tuning experiment. SKILL-IT attains lower validation loss than both no graph and static data selection.

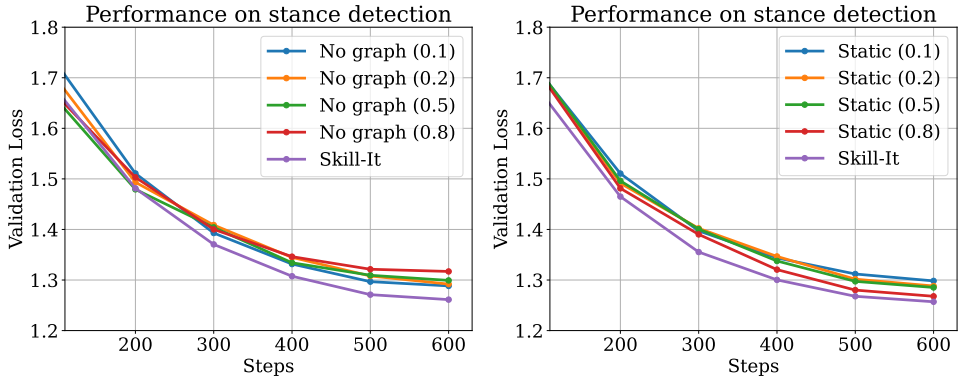


Figure 5.32: Comparison of SKILL-IT versus using no graph (left) and static data selection (right) with $\eta = 0.1, 0.2, 0.5, 0.8$ on the Natural Instructions stance detection fine-tuning experiment. SKILL-IT attains lower validation loss than both no graph and static data selection.

attains lower loss than using no graph or using $T = 1$ round.

Figure 5.33 compares SKILL-IT and static data selection for the Natural Instructions out-of-domain experiment. SKILL-IT attains the lowest validation loss on 7 out of 12 evaluation skills. It has an average loss of 2.540 compared to a range of 2.541-2.551 for static data selection.

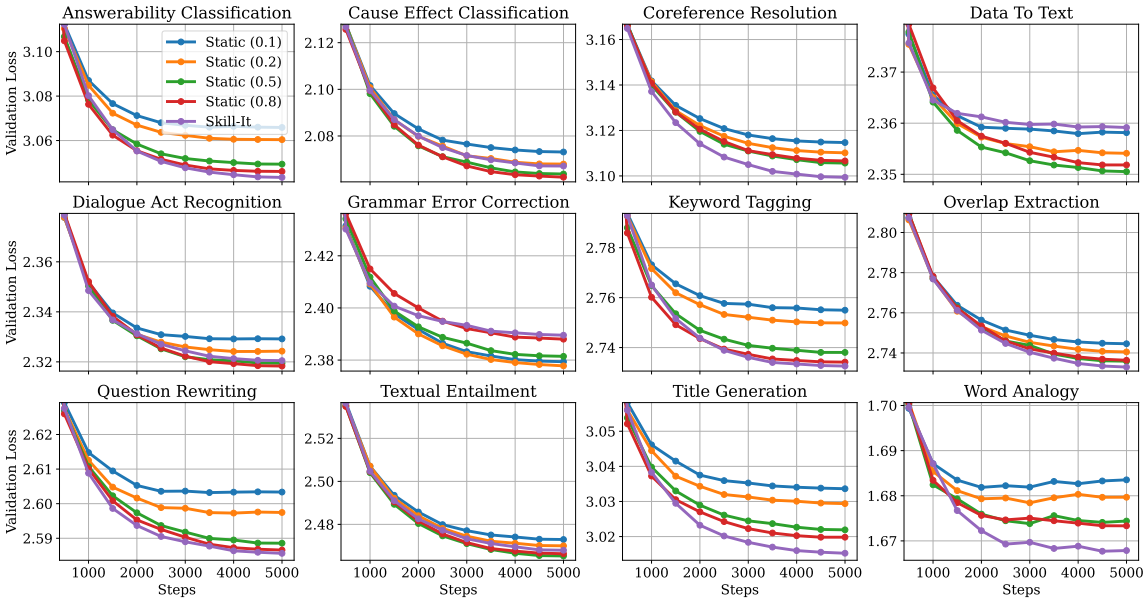


Figure 5.33: Comparison of SKILL-IT versus using static data selection with $\eta = 0.1, 0.2, 0.5, 0.8$ on the Natural Instructions out-of-domain experiment. SKILL-IT attains the lowest validation loss on 7 out of 12 evaluation skills, and an average loss of 2.540 compared to a range of 2.541-2.551 for static data selection.

Chapter 6

Geometry-Aware Adaptation for Pretrained Models

The use of pretrained models is perhaps the most significant recent shift in machine learning. Such models can be used out-of-the-box, either to predict classes observed during pretraining (as in ImageNet-trained ResNets [He et al., 2016b]) or to perform zero-shot classification on any set of classes [Radford et al., 2021]. While this is exciting, label spaces are often so huge that models are unlikely to have seen *even a single point* with a particular label. Without additional modification, pretrained models will simply fail to reliably predict such classes. Instead, users turn to fine-tuning—which requires additional labeled data and training cycles and so sacrifices much of the promise of zero-shot usage.

How can we adapt pretrained models to predict new classes, without fine-tuning or retraining? At first glance, this is challenging: predictive signal comes from labeled training data. However, *relational* information between the classes can be exploited to enable predicting a class even when there are no examples with this label in the training set. Such relational data is commonly available, or can be constructed with the help of knowledge bases, ontologies, or powerful foundation models [Stewart and Ermon, 2017].

How to best exploit relational structure remains unclear, with a number of key challenges: We might wish to know what particular subset of classes is rich enough to enable predicting many (or all) remaining labels. This is crucial in determining whether a training set is usable or, even with the aid of structure, insufficient. It is also unclear how approaches that use relational information interact with the statistical properties of learning, such as

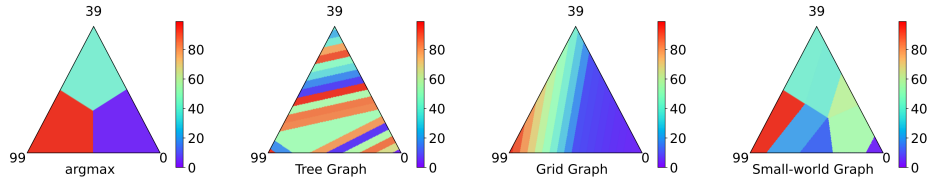


Figure 6.1: Classification regions in the probability simplex of 3-class classifiers faced with a 100-class problem. The probability simplex using arg max prediction can only output one of three classes. Loki uses the entire probability vector to navigate the class metric space, leading to more prediction regions. (Left) regions from arg max prediction. (Centers, Right) classification regions from Loki.

training sample complexity. Finally, performing adaptation requires an efficient and scalable algorithm.

This work addresses these challenges. It proposes a simple and practical approach to learning in structured label spaces, with theoretical guarantees. First, we offer a simple way to translate the soft outputs (i.e., probability vectors) produced by *any* supervised learning model into a more general model that can exploit geometric information for label structure. In other words, our approach, called Loki,[1] is a simple *adaptor* for pretrained models. Loki can be applied via a fixed linear transformation of the model outputs. Loki's simplicity makes it applicable to a broad range of settings while enabling very high-cardinality predictions subject to a potentially small model output budget—we provide a visualization of this key idea in Figure 6.1.

Theoretically, we provide a rich set of results for the metric-based adaptation setting. First, we introduce a learning-theoretic result in terms of the sample complexity of the pretrained model. It captures a key tradeoff between the number of classes and metric structure, the problem dimensionality, and the number of samples used to train the model prior to adaptation. Next we exhaustively study the properties of training sets, determining for a wide class of relational structures the minimum number (and structure) of subsets that enable reliable prediction. Finally we show how to exploit this result in an active learning-like approach to selecting points that will improve deficient training datasets.

Experimentally, we show that using structural information improves prediction in high-cardinality settings. We demonstrate the strength of the active learning-based approach to

[1]Refers to the 'locus' (plural: *loci*) of the Fréchet mean.

dataset expansion over random baselines. Finally, and most excitingly, we show that even in zero-shot models like CLIP, where it is possible to produce probability vectors over any possible class, the use of our adaptor leads to a **19.53%** relative improvement.

6.1 Background

First, we introduce the problem setting, notation, and mathematical tools that we will use. Afterward, we discuss how Loki relates to prior work.

Problem Setting. As in conventional supervised learning, we have a dataset $(x_1, y_1), \ldots, (x_n, y_n)$ drawn from a distribution on $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are the input and label spaces. In our setting, $N := |\mathcal{Y}|$ is finite but large; often $N \gg n$—so that many labels will simply not be found in the training dataset. We let $\Lambda \subseteq \mathcal{Y}$ with $|\Lambda| = K$ be the set of observed labels. For convenience of notation, we also define $\boldsymbol{\lambda} := [\lambda_i]_{i=1}^K$, $\boldsymbol{y} := [y_i]_{j=1}^N$ to be the vectors of elements in Λ and $\mathcal{Y}$.

In addition to our dataset, we have access to a relational structure on $\mathcal{Y}$. We assume that $\mathcal{Y}$ is a metric space with metric (distance) $d : \mathcal{Y}^2 \to \mathbb{R}$; d encodes the relational structure of the label space. Specifically, we model this metric space using a graph, $G = (\mathcal{Y}, \mathcal{E})$ where $\mathcal{E} \subseteq \mathcal{Y}^2 \times \mathbb{R}_+$ is a set of edges relating the labels and the standard shortest-path distance $d : \mathcal{Y}^2 \to \mathbb{R}_{\geqslant 0}$. In addition to its use in prediction, the metric d can be used for evaluating a model by measuring, for example, $\frac{1}{n}\sum_{i=1}^n d^2(f(x_i), y)$—the analogue of the empirical square loss.

Fréchet Mean Estimator. Drawing on ideas from structured prediction [Ciliberto et al., 2016, Rudi et al., 2018b], we use a simple predictor that exploits the label metric. It relies on computing the *Fréchet mean* [Fréchet, 1948], given by

$$m_{\boldsymbol{y}}(\boldsymbol{w}) := \operatorname*{arg\,min}_{y \in \mathcal{Y}} \sum_{i=1}^{K} \boldsymbol{w}_i d^2(y, \boldsymbol{y}_i), \tag{6.1}$$

where $\boldsymbol{w} \in \mathbb{R}^K_{\geqslant 0}$ is a set of weights. The Fréchet mean generalizes the barycenter to metric spaces and is often used in geometric machine learning [Lou et al., 2020].

Locus of the Fréchet mean. The *locus of the Fréchet mean* is the set of all Fréchet means under different weights [Nye et al., 2017]. We write it as $\Pi(\boldsymbol{y}) := \cup_{\boldsymbol{w}\in\Delta^{K-1}} \mathfrak{m}_{\boldsymbol{y}}(\boldsymbol{w})$.

$\Pi(\boldsymbol{y})$ can be thought of the set of all labels in $\mathcal{Y}$ that are reachable by the Fréchet mean given $\{\boldsymbol{y}_i\}_{i=1}^K \subseteq \mathcal{Y}$ and different choices of its parameter $\boldsymbol{w}$. Intuitively, we can think of the locus for a given dataset as describing how usable it is for predicting beyond the observed labels. Trivially, if $\{\boldsymbol{y}_i\}_{i=1}^K = \mathcal{Y}$, then $\Pi(\boldsymbol{y}) = \mathcal{Y}$. We are primarily interested in the case in which $\{\boldsymbol{y}_i\}_{i=1}^K \subset \mathcal{Y}$ yet we still have $\Pi(\boldsymbol{y}) = \mathcal{Y}$, or that $|\Pi(\boldsymbol{y})|$ is at least sufficiently large.

6.1.1 Relation to Prior work

Loki is primarily related to two areas: zero-shot learning and structured prediction.

Zero-Shot Learning. Like zero-shot learning (ZSL), Loki is capable of predicting unobserved classes. Our framework is most closely related to *generalized* ZSL, which uses side information to predict both observed and unobserved classes. Many types of external knowledge are used in ZSL, including text [Norouzi et al., 2014, Socher et al., 2013, Frome et al., 2013], attributes [Lampert et al., 2009, 2014, Farhadi et al., 2009], knowledge graphs [Wang et al., 2018, Roy et al., 2022, Gao et al., 2019], ontologies [Li et al., 2020], and logical rules [Rocktäschel et al., 2015]. Our work is most closely related to ZSL approaches that rely on knowledge graph information. Often, ZSL methods that use knowledge graph information rely on the use of graph neural network architectures [Wang et al., 2018, Kampffmeyer et al., 2018, Kipf and Welling, 2017b]. *However, we note that these architectures can be heavyweight and can be challenging to scale up to extremely large graphs, whereas Loki does not have architectural requirements and scales linearly in the size of the graph when* K *is small.*

Structured Prediction. Structured prediction (SP) operates in label spaces that are endowed with algebraic or geometric structure [Bakir et al., 2007, Kuleshov and Liang, 2015]. This includes problems such as predicting permutations [Korba et al., 2018], non-Euclidean and manifold regression [Petersen and Muller, 2016, Rudi et al., 2018a], and learning on graphs [Graber and Schwing, 2019]. Loki is the most immediately related to the latter, however, any finite metric space can be represented as a graph, which further lends to the flexibility of our approach. Even in discrete structured prediction settings, the cardinality

of the label space may be combinatorially large. *As such, Loki can be viewed as a simple method for adapting classifiers to structured prediction.*

The Fréchet mean has been used in structured prediction—but in approaches requiring training. In [Ciliberto et al., 2016], $\hat{f}(x) = \arg\min_{y \in \mathcal{Y}} \frac{1}{n} \sum_{i=1}^{n} \alpha_i(x) d^2(y, \boldsymbol{y}_i)$, where $\alpha(x) = (K + n\nu I)^{-1} K_x$. K is the kernel matrix for a kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, so that $K_{i,j} = k(x_i, x_j)$, $(K_x)_i = k(x, x_i)$. ν is a regularization parameter. In other words, the weight w_i corresponding to $\boldsymbol{y}_i$ is the average produced by solving a kernel regression problem at all points x_k in the dataset where $y_k = \boldsymbol{y}_i$. It has also used in weak supervision (WS) for metric-equipped label spaces [Vishwakarma and Sala, 2022, Shin et al., 2022], where the goal is to produce labeled data for training structured prediction models.

6.2 Framework

We introduce our framework—Loki. We show how Loki can be used to adapt any supervised classifier over a set of K classes to a much richer set of possible class predictions. It does so by weighting the Fréchet mean by the classifier's per-class prediction probabilities or logits, allowing it to predict any class in the locus of the Fréchet mean—potentially far more classes than the initial K. Next, we show how Loki can be expressed as a *fixed* linear transformation of a model's outputs. Finally, we show that Loki relates to standard classification.

6.2.1 Loki: Adapting Pretrained Models

We describe our approach to adapting pretrained classifiers–trained on a set of classes Λ—to the metric geometry of the label space $\mathcal{Y}$, enabling the prediction of unobserved classes.

We model unobserved classes $\mathcal{Y} \setminus \Lambda$ using the Fréchet mean among observed classes weighted by their *prediction probabilities* $P(y = \lambda_i | x \text{ and } y \in \Lambda)$. We denote the vector of model outputs as $\mathbf{P}_{y|x} := [\mathbf{P}_{\lambda_i|x}]_{i=1}^{K} = [P(y = \lambda_i | x \text{ and } y \in \Lambda)]_{i=1}^{K}$. Then predictions using Loki are given by

$$\hat{y} \in m_{\boldsymbol{\lambda}}(\mathbf{P}_{y|x}) = \arg\min_{y \in \mathcal{Y}} \sum_{i=1}^{K} \mathbf{P}_{\lambda_i|x} d^2(y, \lambda_i).$$

6.2.2 Loki as a linear transformation of model outputs

Most standard classifiers output a vector of prediction probabilities, $\mathbf{P}_{y|x}$, whose entries correspond to the confidence of predicting a specific class. Predictions are typically given by $\hat{y} \in \arg\max_{i \in [K]} (\mathbf{P}_{y|x})_i$. Loki generalizes this prediction rule when viewed as a linear transformation of $\mathbf{P}_{y|x}$. Consider the Loki prediction rule

$$\hat{y} \in m_{\boldsymbol{\lambda}}(\mathbf{P}_{y|x}) = \arg\min_{y \in \mathcal{Y}} \sum_{i=1}^{K} \mathbf{P}_{\lambda_i|x} d^2(y, \lambda_i) = \arg\max_{j \in [N]} (\mathbf{D}\mathbf{P}_{y|x})_j,$$

where $\mathbf{D}_{j,i} = \left[-d^2(y_j, \lambda_i)\right]$; $\mathbf{D} \in \mathbb{R}^{N \times K}$ is the matrix of negative squared distances between the observed classes and the rest of the label space. Thus Loki can be used within standard classification pipelines when the model output $\mathbf{P}_{y|x}$ is multiplied by the fixed matrix $\mathbf{D}$.

6.2.3 Generalizing Standard Classification

We provide a simple intuition for our approach. The fact that Loki reduces to standard classification among the observed classes has several implications. This includes the idea that under our framework, forms of few-shot, zero-shot, hierarchical, and partial label learning all reduce to standard classification when additional metric information is introduced.

Generalizing the arg max prediction rule. In the absence of this metric information—a situation that we model using the complete graph and setting $\Lambda = \mathcal{Y}$—our framework also recovers standard classification. Indeed, both in terms of error modeling and in terms of inter-class similarity, the intuition of standard classification and of the 0-1 loss are captured well by the unweighted complete graph—simply treat all differences equally. This graph is given as $K_N := (\mathcal{Y}, \mathcal{Y}^2 \times \{1\})$—i.e., every label is equidistant from every other label. Plugging this into Loki, we obtain the following:

$$\hat{y} \in m_{\boldsymbol{\lambda}}(\mathbf{P}_{y|x}) = \arg\min_{y \in \mathcal{Y}} \sum_{i=1}^{K} \mathbf{P}_{\lambda_i|x} d^2(y, \lambda_i) = \arg\min_{y \in \mathcal{Y}} \sum_{i=1}^{K} \mathbf{P}_{\lambda_i|x} \mathbf{1}\{y \neq \lambda_i\} = \arg\max_{i \in [K]} \mathbf{P}_{\lambda_i|x},$$

which is exactly the standard classification prediction rule.

Generalizing the 0-1 loss via the expected squared distance. *The expected squared distance* $\mathbb{E}[d^2(y, \hat{y})]$ *is the standard loss function for many structured prediction problems.* Note that accuracy fails in such settings—since it cannot distinguish between small and large errors. This is most clearly seen in the extreme case of regression, where test accuracy will virtually always be zero no matter how good a trained model is. At the other extreme, the complete graph, this loss function becomes the standard 0-1 loss: $\mathbb{E}[d^2(y, \hat{y})] = \mathbb{E}[\mathbf{1}\{y \neq \hat{y}\}]$. As an adaptor for structured label spaces, we use the empirical version of this loss to evaluate Loki. Note that the expected squared distance subsumes other metrics as well. For example, when $\mathcal{Y} = \mathbb{R}$, we can derive the standard MSE by setting $d(y, \hat{y}) = |y - \hat{y}|$, which is just the standard L1 distance metric. Other scores such as recall at top-k can be similarly obtained at the cost of $d(\cdot, \cdot)$ being a true metric. In other words, $\mathbb{E}[d^2(y, \hat{y})]$ is an very general metric that supports any metric space, and we use it throughout this work.

6.3 Theoretical Results

Challenges and Opportunities. The arg max of per-class model probabilities is a ubiquitous component of classification pipelines in machine learning. In order to predict unobserved classes using metric space information, Loki replaces this standard component. As a simple but significant change to standard pipelines, Loki opens up a new area for fundamental questions. There are three main flavors of theoretical questions that arise in this setting:

1. How does the performance of Loki change as a function of the number of samples?
2. What minimal sets of observed classes are required to predict any class in the metric space?
3. How can we acquire new classes that maximize the total number of classes we can predict?

Excitingly, we provide a comprehensive answer to these questions for the most common metric spaces used in practice. First, we provide a general error bound in terms of the number of samples, observed classes, problem dimension, and the diameter of the metric space, that holds for any finite metric space. Second, we characterize the sets of observed classes that are required to enable prediction of any class, and show how this set differs for

various types of metric spaces of interest. Finally, we provide an active learning algorithm for selecting additional classes to observe so as to maximize the number of classes that can be predicted and we characterize the types of metric spaces for which the locus can be computed efficiently. These results provide a strong theoretical grounding for LOKI.

6.3.1 Sample Complexity

What is the interplay between predicting unobserved classes based on metric space information and standard learning-theoretic notions like the sample complexity needed to train a model? Our first result illustrates this tradeoff, and relates it to the squared distance loss. Suppose that we only observe $K \leqslant N$ of the classes at training time, and that we fit a K-class Gaussian mixture model. We use LOKI to adapt our pretrained classifier to enable classification of any of the N classes. We have that (a formal statement and interpretation are in the Appendix),

Theorem 6.1 (Informal LOKI sample complexity**).** *Let $\mathcal{Y}$ be a set of classes represented by d dimensional vectors under the Euclidean distance, and let $\Lambda \subseteq \mathcal{Y}$ be the set of K observed classes. Assume that n training examples are generated by an identity covariance Gaussian mixture model over classes Λ, and that test examples are generated over all classes $\mathcal{Y}$. Assume that we estimate a Gaussian mixture model on the training set and obtain probability estimates $\hat{\mathbb{P}}(y_i|x)$ for $i \in [K]$ for a sample (x, y_*) from the test distribution. Then with high probability, under the following model,*

$$\hat{y}_* \in m_\Lambda([\hat{\mathbb{P}}(y_i|x)]_{i\in[K]}) = \arg\min_{y\in\mathcal{Y}} \sum_{i\in[K]} \hat{\mathbb{P}}(y_i|x)\, d^2(y, y_i)$$

the sample complexity of estimating target y_* from the test distribution $\mathcal{D}_{test}$ with prediction $\hat{y}_*$ is:

$$\mathbb{E}_{(x,y_*)\sim\mathcal{D}_{test}}[d^2(y_*, \hat{y}_*)] \leqslant O\left(\frac{d}{\alpha}\sqrt{\frac{\log K/\delta}{n}}\left(\frac{1}{\left(R^{1-\frac{2}{d}} - \frac{\log R}{R}\right)} + \sqrt{d}\right)\right)$$

where α is related to the sensitivity of the Fréchet variance to different node choices.

6.3.2 Optimal Label Subspaces

Next we characterize the subset of distinct labels required to predict any label using our label model with respect to various types of metric spaces.

We first consider label spaces whose metric is a tree graph—such metrics are, for example, related to performing hierarchical classification and weak supervision (where only partial labels are available). We consider a special type of tree called a phylogenetic tree, in which only the leaves can be designated as labels—phylogenetic trees are commonly used to relate the labels of image classification datasets. Afterwards we perform a similar analysis for grid graphs, which are important for label spaces that encode spatial information. Finally, we discuss the case in which no useful metric information is available, i.e., the complete graph.

Our goal in this section is to characterize the properties and size of $\{\lambda_i\}_{i=1}^K$ in each of these metric spaces such that we still have $\Pi(\boldsymbol{\Lambda}) = \mathcal{Y}$. We characterize 'optimal' subsets of classes in each of the spaces under a certain notions of optimality. We provide several relevant definitions pertaining to this concept, starting with a notion of being able to predict any possible class using observed classes.

Definition 6.2 (Locus cover)**.** *Given a set $\Lambda \subseteq \mathcal{Y}$ for which we construct a tuple of its elements $\boldsymbol{\Lambda}$, if it holds that $\Pi(\boldsymbol{\Lambda}) = \mathcal{Y}$, then Λ is a locus cover.*

Definition 6.2 captures the main idea of Loki—using some set of observed classes for which we can train classifiers, we would like to be able to predict additional unobserved classes using the geometry that relates the observed and unobserved classes. Namely, elements of $\Pi(\boldsymbol{\Lambda})$ are 'reachable' using Loki. We refine this Definition to describe the trivial case that defaults to standard classification and the nontrivial case for which Loki moves beyond standard classification.

Definition 6.3 (Trivial locus cover)**.** *If $\Lambda = \mathcal{Y}$, then Λ is the trivial locus cover.*

This Definition captures the notion of observing all of the classes in the label space. Here, all of the elements of $\mathcal{Y}$ are trivially reachable using Loki.

Definition 6.4 (Nontrivial locus cover)**.** *A locus cover Λ is nontrivial if $\Lambda \neq \mathcal{Y}$.*

Loki is more useful and interesting when faced with a nontrivial locus cover—under Definition 6.4, we can use some subset of classes Λ to predict any label in $\mathcal{Y}$.

Definition 6.5 (Minimum locus cover)**.** *Given a set $\Lambda \subseteq \mathcal{Y}$, if Λ is the smallest set that is still a locus cover, then it is a minimum locus cover.*

In cases involving an extremely large number of classes, it is desirable to use Loki on the smallest possible set of observed classes Λ such that all labels in $\mathcal{Y}$ can still be predicted. Definition 6.5 characterizes these situations—later, we obtain the minimum locus covers for all trees and grid graphs. It is worth noting that the minimum locus cover need not be unique for a fixed graph.

Definition 6.6 (Identifying locus cover)**.** *Given a set $\Lambda \subseteq \mathcal{Y}$, if Λ is a locus cover where $\forall y \in \mathcal{Y}, \exists \mathbf{w} \in \Delta^{|\Lambda|-1}$ such that $\mathfrak{m}_{\boldsymbol{\Lambda}}(\mathbf{w}) = \{y\}$, then Λ is an identifying locus cover.*

The Fréchet mean need not be unique—as an arg min, it returns a set of minimizers. In certain metric spaces, the minimum locus cover can yield large sets of minimizers—this is undesirable, as it makes predicting a single class challenging. Definition 6.6 appeals to the idea of finding some set of classes for which the Fréchet mean *always* returns a unique minimizer—this is desirable in practice, and in some cases, moreso than Definition 6.5.

Definition 6.7 (Pairwise decomposable)**.** *Given $\Lambda \subseteq \mathcal{Y}$, $\Pi(\Lambda)$ is called pairwise decomposable when it holds that $\Pi(\Lambda) = \cup_{\lambda_1,\lambda_2 \in \Lambda} \Pi(\{\lambda_1, \lambda_2\})$.*

In many cases, the locus can be written in a more convenient form—the union of the locus of pairs of nodes. We refer to this definition as pairwise decomposability. Later, we shall see that pairwise decomposability is useful in computing the locus in polynomial time.

Trees. Many label spaces are endowed with a tree metric in practice: hierarchical classification, in which the label space includes both classes and superclasses, partial labeling problems in which internal nodes can represent the prediction of a set of classes, and the approximation of complex or intractable metrics using a minimum spanning tree. We show that for our purposes, trees have certain desirable properties that make them easy to use with Loki—namely that we can easily identify a locus cover that satisfies both Definition 6.5 and Definition 6.6. Conveniently, we also show that any locus in any tree satisfies Definition 6.7.

We first note that the leaves of any tree yield the minimum locus cover. This is a convenient property—any label from any label space endowed with a tree metric can be predicted using Loki using only classifiers trained using labels corresponding to the leaves

of the metric space. This can be especially useful if the tree has long branches and few leaves. Additionally, for tree metric spaces, the minimum locus cover (Definition 6.5) is also an identifying locus cover (Definition 6.6). This follows from the construction of the weights in the proof of Theorem 6.11 (shown in the Appendix) and the property that all paths in trees are unique. Finally, we note that any locus in any tree is pairwise decomposable—the proof of this is given in the Appendix (Lemma 6.12). We will see later that this property yields an efficient algorithm for computing the locus.

Phylogenetic Trees. Image classification datasets often have a hierarchical tree structure, where only the leaves are actual classes, and internal nodes are designated as superclasses—examples include the ImageNet [Deng et al., 2009] and CIFAR-100 datasets [Krizhevsky et al., 2009]. Tree graphs in which only the leaf nodes are labeled are referred to as phylogenetic trees [Billera et al., 2001]. Often, these graphs are weighted, but unless otherwise mentioned, we assume that the graph is unweighted.

For any arbitrary tree $T = (\mathcal{V}, \mathcal{E})$, the set of labels induced by phylogenetic tree graph is $\mathcal{Y} = \text{Leaves}(T)$. We provide a heuristic algorithm for obtaining locus covers for arbitrary phylogenetic trees in Algorithm 4 (see Appendix). We prioritize adding endpoints of long paths to Λ, and continue adding nodes in this way until $\Pi(\Lambda) = \mathcal{Y}$. Similarly to tree metric spaces, any phylogenetic tree metric space is pairwise decomposable. We prove the correctness of Algorithm 4 and pairwise decomposability of phylogenetic trees in the Appendix (Theorem 6.13 and Lemma 6.14). Later, we give algorithms for computing the set of nodes in an arbitrary locus in arbitrary graphs—if the locus is pairwise decomposable, the algorithm for doing so is efficient, and if not, it has time complexity exponential in K. Due to the pairwise decomposability of phylogenetic trees, this polynomial-time algorithm to compute $\Pi(\Lambda)$ applies.

Grid Graphs. Classes often have a spatial relationship. For example, classification on maps or the discretization of a manifold both have spatial relationships—grid graphs are well suited to these types of spatial relationships. We obtain minimum locus covers for grid graphs satisfying Definition 6.5, but we find that these are not generally identifying locus covers. On the other hand, we give an example of a simple identifying locus cover satisfying Definition 6.6. Again, we find that grid graphs are in general pairwise decomposable and hence follow Definition 6.7.

We find that the pair of vertices on furthest opposite corners yields the minimum locus cover. While the set of vertices given by Theorem 6.15 (found in the Appendix) satisfies Definition 6.5, this set does not in general satisfy Definition 6.6. This is because the path between any two vertices is not unique, so each minimum path of the same length between the pair of vertices can have an equivalent minimizer. On contrast, the following example set of vertices satisfies Definition 6.6 but it clearly does not satisfy Definition 6.5. *Example*: Given a grid graph, the set of all corners is an identifying locus cover. On the other hand, the vertices given by Theorem 6.15 can be useful for other purposes. Lemma 6.16 (provided in the Appendix) shows that subspaces of grid graphs can be formed by the loci of pairs of vertices in Λ. This in turn helps to show that loci in grid graphs are pairwise decomposable in general (see Lemma 6.17 in the Appendix).

The Complete Graph. The standard classification setting does not use relational information between classes. As before, we model this setting using the complete graph, and we show the expected result that in the absence of useful relational information, Loki cannot help, and the problem once again becomes standard multiclass classification among observed classes. To do so, we show that there is no nontrivial locus cover for the complete graph (Theorem 6.18 in the Appendix).

6.3.3 Label Subspaces in Practice

While it is desirable for the set of observed classes to form a minimum or identifying locus cover, it is often not possible to choose the initial set of observed classes a priori—these are often random. In this section, we describe the more realistic cases in which a random set of classes are observed and an active learning-based strategy to choose the next observed class. The aim of our active learning approach is, instead of randomly selecting the next observed class, to actively select the next class so as to maximize the total size of the locus—i.e., the number of possible classes that can be output using Loki. Before maximizing the locus via active learning, we must first address a much more basic question: can we even efficiently compute the locus?

Computing the Locus. We provide algorithms for obtaining the set of all classes in the locus, given a set of classes Λ. We show that when the locus is pairwise decomposable

(Definition 6.7), we can compute the locus efficiently using a polynomial time algorithm. When the locus is not pairwise decomposable, we provide a general algorithm that has time complexity exponential in $|\Lambda|$—we are not aware of a more efficient algorithm. We note that any locus for every type of graph that we consider in Section 6.3.2 is pairwise decomposable, so our polynomial time algorithm applies. Algorithms 5 and 6 along with their time complexity analyses can be found in the Appendix.

Large Locus via Active Next-Class Selection. We now turn to actively selecting the next class to observe in order to maximize the size of the locus. For this analysis, we focus on the active learning setting when the class structure is a tree graph, as tree graphs are generic enough to apply to a wide variety of cases—including approximating other graphs using the minimum spanning tree. Assume the initial set of K observed classes are sampled at random from some distribution. We would like to actively select the $K+1$st class such that $|\Pi(\Lambda)|$ with $\Lambda = \{\lambda\}_{i=1}^{K+1}$ is as large as possible.

Theorem 6.8. *Let $T = (\mathcal{Y}, \mathcal{E})$ be a tree graph and let $\Lambda \subseteq \mathcal{Y}$ with $K = |\Lambda|$. Let T' be the subgraph of the locus $\Pi(\Lambda)$. The vertex $v \in \mathcal{Y} \setminus \Lambda$ that maximizes $|\Pi(\Lambda \cup \{v\})|$ is the solution to the following optimization problem:* $\arg\max_{y \in \mathcal{Y} \setminus \Pi(\Lambda)} d(y, b)$ *s.t.* $b \in \partial_{in} T'$ *and* $\Gamma(y, b) \setminus \{b\} \subseteq \mathcal{Y} \setminus \Pi(\Lambda)$. *where $\partial_{in} T'$ is the inner boundary of T' (all vertices in T' that share an edge with vertices not in T').*

This procedure can be computed in polynomial time—solving the optimization problem in Theorem 6.8 simply requires searching over pairs of vertices. Hence we have provided an efficient active learning-based strategy to maximize the size of the locus for trees.

6.4 Experimental Results

In this section, we provide experimental results to validate the following claims:

1. Loki improves performance of zero-shot foundation models even with no external metric.
2. Loki adapts to label spaces with a large number of unobserved classes.
3. The active approach given in Theorem 6.8 yields a larger locus than the passive baseline.

Table 6.1: CIFAR-100. Improving CLIP predictions using Loki. Results are reported as $\mathbb{E}[d^2(y, \hat{y})]$ in the respective metric space. CLIP-like zero-shot models can be improved using Loki even without access to an external metric, and internal class embedding distances are used. When an external metric is available, Loki outperforms CLIP using the default CIFAR-100 hierarchy and WordNet.

Model	Metric Space	arg max	Loki	Relative Improvement
CLIP-RN50 [Radford et al., 2021]	Internal	0.2922	**0.2613**	**10.57%**
CLIP-ViT-L-14 [Radford et al., 2021]	Internal	0.1588	**0.1562**	**1.63%**
ALIGN [Jia et al., 2021]	Internal	0.1475	**0.1430**	**3.02%**
CLIP-RN50 [Radford et al., 2021]	K_{100}	0.5941*	0.5941*	0.0%*
CLIP-RN50 [Radford et al., 2021]	Default tree	7.3528	**7.1888**	**2.23%**
CLIP-RN50 [Radford et al., 2021]	WordNet tree	24.3017	**19.5549**	**19.53%**

* methods equivalent under the complete graph as Loki reduces to arg max prediction.

4. The same active approach yields better performance on ImageNet.

5. With Loki, calibration can improve *accuracy*, even with no external metric.

6.4.1 Loki Improves Zero-Shot Models

We evaluate the capability of Loki to improve upon zero-shot models where all classes are observed.

Setup. Our experiment compares the zero-shot prediction performance of CLIP [Radford et al., 2021] on CIFAR-100 [Krizhevsky et al., 2009] to CLIP logits used with Loki. First, we consider the setting in which no external metric relating the labels is available, and instead derive internal metric information from Euclidean distances between text embeddings from the models using their respective text encoders. Second, we consider three external metric spaces for use with Loki: the complete graph K_{100} and two phylogenetic trees: the default CIFAR-100 superclasses [Krizhevsky et al., 2009] and WordNet [Barz and Denzler, 2019].

Results. The results of this experiment are given in Table 6.1. When no external metric information is available, Loki still outperforms CLIP-like models that use the standard prediction rule—in other words, Loki seems to unconditionally improve CLIP. As expected,

under the complete graph, our method becomes equivalent to the standard prediction mechanism used by CLIP. On the other hand, Loki outperforms CLIP when using the default CIFAR-100 tree hierarchy and even more so when using the WordNet geometry, with a **19.53%** relative improvement in mean squared distance over the CLIP baseline. We postulate that the strong performance using WordNet is due to the richer geometric structure compared to that of the default CIFAR-100 hierarchy.

6.4.2 Loki on Partially-Observed Label Spaces

To validate our approach on partially observed label spaces, we evaluate the performance of adapting a logistic classifier trained on SimCLR embeddings of ImageNet [Deng et al., 2009], 5-NN models trained on a 9,419 class subset of the PubMed dataset,[2] and the 325,056-class LSHTC dataset [Partalas et al., 2015].

Setup. For ImageNet, we use the WordNet phylogenetic tree as the metric space [Barz and Denzler, 2019]. In this setting, we sample random subsets of size K of the 1000 ImageNet classes and compare a baseline one-vs-rest classifier to the same classifier but using Loki to adapt predictions to classes beyond the original K. We conduct two sets of experiments. In the first, we sample K classes uniformly, while in the second, we adopt a more realistic sampler—we sample from a Gibbs distribution: $P(\lambda|\theta) = \frac{1}{Z}\exp(-\theta d(\lambda, \lambda^c))$, where $\lambda^c = m_y(\mathbf{1}_N)$ is the centroid of the metric space, θ is the concentration parameter around the centroid, and Z is the normalizer. While the Gibbs distribution sampler is more realistic, it is also the more challenging setting—classes which have low probability according to this distribution are less likely to appear in the locus. For PubMed, we derive our metric from Euclidean distances between SimCSE class embeddings [Gao et al., 2021b]. Finally for LSHTC, we summarize the default graph by randomly selecting nodes and merging them with their neighbors until we obtain a graph with 10,000 supernodes representing sets of classes.

Results. Figure 6.2 shows the mean squared distances compared to the baseline one-vs-rest classifiers, across various settings of K. We find that Loki always significantly

[2] https://www.kaggle.com/datasets/owaiskhan9654/pubmed-multilabel-text-classification

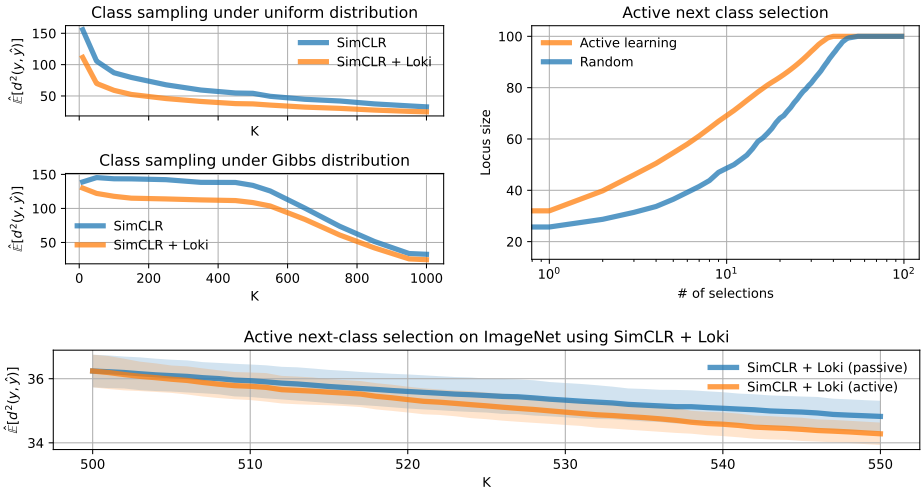


Figure 6.2: (Top left) ImageNet. Mean squared distances under uniform class sampling and under the Gibbs distribution—Loki improves upon the baseline SimCLR one-vs-rest classifier. (Top right) Synthetic. Active class selection consistently leads to larger loci compared to uniform sampling. (Bottom) Active selection on ImageNet. Active class selection improves performance on ImageNet.

outperforms the baseline, even in the more challenging setting of sampling according to a Gibbs distribution. Tables 6.2 and 6.3 show our improvements when using Loki over the baseline 5-NN models. While Loki consistently yields an improvement on PubMed and LSHTC, the improvement is more dramatic on LSHTC.

6.4.3 Large Loci via Active Learning

We evaluate our active next class selection approach on increasing the locus size. We expect that compared to passive (random) selection, our active approach will lead to larger loci, and in practice, a larger set of possible classes that can be predicted using Loki while observing fewer classes during training.

Setup. We compare with a baseline that randomly selects the next class. We first generate a synthetic random tree with size N and fix an initial K. In active selection, we use the approach described in Theorem 6.8 to select the next node. As a passive baseline, we randomly sample (without replacement) the remaining nodes that are not in Λ.

Results. The results are shown in Figure 6.2. We set $N = 100$ and the initial $K = 3$, then we average the result over 10 independent trials. The active approach consistently outperforms the random baseline as it attains a larger locus with fewer nodes selected.

6.4.4 Improving Performance via Active Learning

Next, we evaluate the our active next class selection approach on improving error on ImageNet. While we found that the the active approach indeed leads to an increased locus size compared to the passive baseline, we expect that this increased locus size will lead to improved performance.

Setup. We randomly sample 500 ImageNet classes, and using the WordNet metric space, we use our active approach to iteratively sample 50 more classes. We compare this to a passive baseline in which the 50 classes are sampled randomly. We repeat this experiment over 10 independent trials.

Results. Figure 6.2 shows that our active approach yields improved error over the passive baseline. The gap between the active approach and the passive baseline widens with more selection rounds.

6.4.5 Improving Accuracy via Calibration

Finally, we evaluate the effect of calibrating the Softmax outputs on the performance of Loki.

Table 6.2: PubMed. Loki improves baseline for all settings of K. The metric space is Euclidean distances applied to SimCSE embeddings.

K	5-NN	5-NN + Loki
100	1.68666	**1.42591**
250	1.52374	**1.47801**
500	1.64074	**1.45921**

Table 6.3: LSHTC. Loki improves baseline for all settings of K. We summarize the metric space graph by generating 10,000 supernodes.

K	5-NN	5-NN + Loki
50	1.2519	**0.2896**
100	1.4476	**0.3467**
150	1.7225	**0.3969**
200	1.7092	**0.3171**
250	1.6465	**0.3995**
300	1.6465	**0.4004**

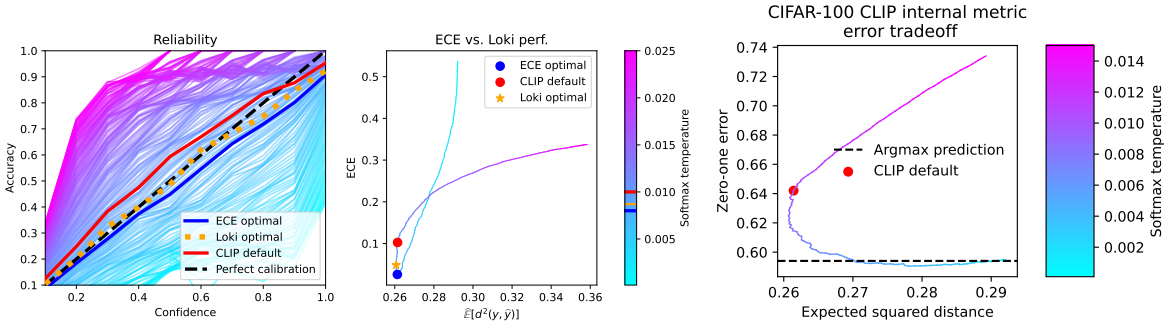


Figure 6.3: CLIP on CIFAR-100 with no external metric. (Left) Reliability diagrams across a range of Softmax temperatures, highlighting the CLIP default temperature, the optimal temperature for Loki, and the minimizer of the Expected Calibration Error (ECE). All three are well-calibrated. (Center) Tradeoff between optimizing for ECE and the expected squared distance. As with the reliability diagrams, the CLIP default temperature, the Loki-optimal temperature, and the ECE-optimal temperature are similar. (Right) Tradeoff between optimizing for zero-one error and the expected squared distance. Temperature can be tuned to improve *accuracy* when using Loki.

Setup. We calibrate via Softmax temperature scaling [Guo et al., 2017] using CLIP on CIFAR-100. We do not use an external metric space, and instead use Euclidean distance applied to the CLIP text encoder.

Results. The reliability diagram in Figure 6.3 shows that the optimal Softmax temperature for Loki is both close to the default temperature used by CLIP and to the optimally-calibrated temperature. In Figure 6.3 (right), we find that appropriate tuning of the temperature parameter *can lead to improved accuracy with CLIP*, even when no external metric space is available.

6.5 Conclusion

In this work, we proposed Loki—a simple adaptor for pretrained models to enable the prediction of additional classes that are unobserved during training by using metric space information. We comprehensively answered the space of new questions that arise under Loki in terms of learning, optimal metric space settings, and a practical active selection strategy. Experimentally, we showed that Loki can be used to improve CLIP even without

external metric space information, can be used to predict a large number of unobserved classes, and a validation of our active selection strategy.

The appendix is organized as follows: in Appendix 6.A, we provide proofs of the claims made in the main text. Next, in Appendix 6.B, we provide algorithms and time complexity analyses for the algorithms referenced in the main text. Appendix 6.C contains additional experimental results. In Appendix 6.D, we provide additional details about our experimental setup. Then in Appendix 6.E, we discuss the broader impacts and limitations of Loki. Finally, in Appendix 6.F, we provide more details and examples of the locus visualizations shown in Figure 6.1.

6.A Deferred Proofs

6.A.1 Proof of Theorem 6.1 (Loki sample complexity)

In this section, we provide a formal statement and proof of our sample complexity result for Loki, including additional required definitions and assumptions.

First, we define the required tools to prove the sample complexity bound for Loki. For the purposes of this proof, we define the Fréchet variance as the function over which the Fréchet mean returns the minimizer.

Definition 6.9 (Fréchet variance)**.** *The Fréchet variance is defined as*

$$\Psi_{\mathbf{w}}(V) := \sum_{i \in [K]} \mathbf{w}_i d^2(V, V_i).$$

Additionally, we will require a technical assumption related to the sensitivity of the Fréchet variance to different node choices.

Assumption 6.1 (Fréchet variance is $\frac{1}{\alpha}$-bi-Lipschitz)**.** *For a metric space defined by a graph $G = (\mathcal{V}, \mathcal{E})$, the Fréchet variance is K-bi-Lipschitz if there exists a $K \geqslant 1$ such that*

$$\frac{1}{K} d^2(V, \tilde{V}) \leqslant |\Psi_{\mathbf{w}}(V) - \Psi_{\mathbf{w}}(\tilde{V})| \leqslant K d^2(V, \tilde{V})$$

for all $V, \tilde{V} \in \mathcal{V}$, and a fixed $\mathbf{w} \in \Delta^{K-1}$. For our purposes, such a K always exists: consider setting $K = diam(G)^2 \max_{V_1, V_2 \in \mathcal{V}} |\Psi_{\mathbf{w}}(V_1) - \Psi_{\mathbf{w}}(V_2)|$. However, this is a very conservative

bound that holds for all graphs that we consider. Instead, we assume access to the largest $\alpha = \frac{1}{K} \leqslant 1$ *such that*

$$\alpha d^2(V, \tilde{V}) \leqslant |\Psi_{\mathbf{w}}(V) - \Psi_{\mathbf{w}}(\tilde{V})|,$$

which may be problem dependent.

Theorem 6.10 (LOKI sample complexity)**.** *Let* $\mathcal{Y}$ *be a set of points on the* d *dimensional 2-norm ball of radius* R*, and let* $\Lambda \subseteq \mathcal{Y}$ *be the set of* K *observed classes. Assume that* Λ *forms a* $2R/(\sqrt{d}K - 1)$*-net under the Euclidean distance. Assume that training examples are generated by drawing* n *samples from the following process: draw* $x \sim \mathcal{N}(y, I)$ *where* $y \sim$ *Unif*(Λ)*, and at test time, draw* $x \sim \mathcal{N}(y, I)$ *where* $y \sim$ *Unif*$(\mathcal{Y})$*. Assume that we estimate a Gaussian mixture model with* K *components (each having identity covariance) on the training set and obtain probability estimates* $\hat{\mathbb{P}}(y_i|x)$ *for* $i \in [K]$ *for a sample* (x, y_*) *from the test distribution. Then with high probability, under the following model,*

$$\hat{y}_* \in m_\Lambda([\hat{\mathbb{P}}(y_i|x)]_{i\in[K]}) = \arg\min_{y\in\mathcal{Y}} \sum_{i\in[K]} \hat{\mathbb{P}}(y_i|x) d^2(y, y_i)$$

the sample complexity of estimating target y_* *from the test distribution* $\mathcal{D}_{test}$ *with prediction* $\hat{y}_*$ *is:*

$$\mathbb{E}_{(x,y_*)\sim\mathcal{D}_{test}}[d^2(y_*, \hat{y}_*)] \leqslant O\left(\frac{d}{\alpha}\sqrt{\frac{\log K/\delta}{n}}\left(\frac{1}{\left(R^{1-\frac{2}{d}} - \frac{\log R}{R}\right)} + \sqrt{d}\right)\right)$$

where d *is the dimensionality of the input,* M *is the number of classes that are near* x*, and* α *is our parameter under Assumption 6.1.*

Proof. We begin by detailing the data-generating process.

At training time, our underlying data-generating process, $\mathcal{D}_{\text{train}}$, is as follows:

- We begin with a $\frac{2R}{\sqrt{d}K-1}$-net, $\mathcal{Y}$, on the d dimensional 2-norm ball of radius R under the Euclidean distance, and let $\Lambda \subseteq \mathcal{Y}$
- Draw $y \sim \text{Unif}(\mathcal{Y})$.
- Discard draws if $y \notin \Lambda$.
- Draw $x \sim \mathcal{N}(y, I)$.

At test time, we do not discard draws and allow for classes not in Λ and use the following data generating process, $\mathcal{D}_{\text{test}}$:

- We begin with a $\frac{2R}{\sqrt{d}K-1}$-net, $\mathcal{Y}$, on the d dimensional 2-norm ball of radius R under the Euclidean distance, and let $\Lambda \subseteq \mathcal{Y}$
- Draw $y \sim \text{Unif}(\mathcal{Y})$.
- Draw $x \sim \mathcal{N}(y, I)$.

Given a labeled training dataset $D = \{(x_i, y_i)\}_{i=1}^n$ containing n points drawn from $\mathcal{D}_{\text{train}}$ with $|\Lambda| = K$ distinct classes, we would like to fit a K-component Gaussian mixture model with identity covariance. We first perform mean estimation of each of the classes separately using the median-of-means estimator [Hsu and Sabato, 2016, Minsker, 2015, Lerasle and Oliveira, 2011]. Using thie estimator yields the following parameter estimation bound [Minsker, 2015, Cherapanamjeri et al., 2019]:

$$\|y_i - \hat{y}_i\|_2 \leqslant O\left(\sqrt{\frac{d \log K/\delta}{n}}\right)$$

with probability $1-\delta$.

Next, we consider the relationships between four quantities: $\Psi_{\mathbb{P}}(y_*)$, $\Psi_{\mathbb{P}}(\hat{y}_*)$, $\Psi_{\widehat{\mathbb{P}}}(y_*)$, $\Psi_{\widehat{\mathbb{P}}}(\hat{y}_*)$, where $\mathbb{P}$ is the vector of probabilities from the true Gaussian mixture, $\widehat{\mathbb{P}}$ is the vector of probabilities from the estimated model, $y_* \in m_\Lambda(\mathbb{P})$ is the target class, and $\hat{y}_* \in m_\Lambda(\widehat{\mathbb{P}})$ is the predicted class. While it is problem-dependent as to whether $\Psi_{\mathbb{P}}(y_*) \leqslant \Psi_{\widehat{\mathbb{P}}}(\hat{y}_*)$ or $\Psi_{\mathbb{P}}(y_*) > \Psi_{\widehat{\mathbb{P}}}(\hat{y}_*)$, a similar argument holds for both cases. So without loss of generality, we assume that $\Psi_{\mathbb{P}}(y_*) \leqslant \Psi_{\widehat{\mathbb{P}}}(\hat{y}_*)$. Then by the definition of the Fréchet mean, the following inequalities hold:

$$\Psi_{\mathbb{P}}(y_*) \leqslant \Psi_{\widehat{\mathbb{P}}}(\hat{y}_*) \leqslant \Psi_{\widehat{\mathbb{P}}}(y_*),$$

and consequently,

$$\Psi_{\widehat{\mathbb{P}}}(\hat{y}_*) - \Psi_{\mathbb{P}}(y_*) \leqslant \Psi_{\widehat{\mathbb{P}}}(y_*) - \Psi_{\mathbb{P}}(y_*). \tag{6.2}$$

We proceed by obtaining upper and lower bounds of the existing bounds in Equation 6.2.

First, we will obtain an upper bound on $\Psi_{\widehat{\mathbb{P}}}(y_*) - \Psi_{\mathbb{P}}(y_*)$.

$$
\begin{aligned}
|\Psi_{\widehat{\mathbb{P}}}(y_*) - \Psi_{\mathbb{P}}(y_*)| &= \left| \sum_{i \in [K]} (\widehat{\mathbb{P}} - \mathbb{P}) d^2(y_*, V_i) \right| \\
&= \left| \sum_{i \in [K]} \left(\widehat{\mathbb{P}}(y_i|x) - \mathbb{P}(y_i|x) \right) ||y_* - y_i||_2^2 \right| \\
&= \left| \sum_{i \in [K]} \left(\frac{\widehat{\mathbb{P}}(x|y_i)\mathbb{P}(y_i)}{\sum_{j \in [K]} \widehat{\mathbb{P}}(x|y_j)\mathbb{P}(y_j)} - \frac{\mathbb{P}(x|y_i)\mathbb{P}(y_i)}{\sum_{j \in [K]} \mathbb{P}(x|y_j)\mathbb{P}(y_j)} \right) ||y_* - y_i||_2^2 \right|
\end{aligned} \tag{6.3}
$$

$$
\begin{aligned}
&= \left| \sum_{i \in [K]} \left(\frac{\exp\{-\frac{1}{2}||x - \hat{y}_i||_2^2\}\frac{1}{K}}{\sum_{j \in [K]} \exp\{-\frac{1}{2}||x - \hat{y}_j||_2^2\}\frac{1}{K}} \right.\right. \\
&\qquad \left.\left. - \frac{\exp\{-\frac{1}{2}||x - y_i||_2^2\}\frac{1}{K}}{\sum_{j \in [K]} \exp\{-\frac{1}{2}||x - y_j||_2^2\}\frac{1}{K}} \right) ||y_* - y_i||_2^2 \right|. \\
&= \left| \sum_{i \in [K]} \left(\frac{\exp\{-\frac{1}{2}||x - \hat{y}_i||_2^2\}}{\sum_{j \in [K]} \exp\{-\frac{1}{2}||x - \hat{y}_j||_2^2\}} \right.\right. \\
&\qquad \left.\left. - \frac{\exp\{-\frac{1}{2}||x - y_i||_2^2\}}{\sum_{j \in [K]} \exp\{-\frac{1}{2}||x - y_j||_2^2\}} \right) ||y_* - y_i||_2^2 \right|.
\end{aligned} \tag{6.4}
$$

For notational convenience, we define the following:
$a_i := \exp\{-\frac{1}{2}||x - y_i||_2^2\}$,
$\hat{a}_i := \exp\{-\frac{1}{2}||x - \hat{y}_i||_2^2\}$,
$b := \sum_{j \in [K]} \exp\{-\frac{1}{2}||x - y_j||_2^2\}$,
$\hat{b} := \sum_{j \in [K]} \exp\{-\frac{1}{2}||x - \hat{y}_j||_2^2\}$, and
$c_i := ||\hat{y}_* - y_i||_2^2$.

Then (6.4) becomes:

$$\begin{aligned}
\left|\sum_{i\in[K]}\left(\frac{\hat{a}_i}{\hat{b}}-\frac{a_i}{b}\right)c_i\right| &= \left|\sum_{i\in[K]}\left(\frac{\hat{a}_i}{\hat{b}}-\frac{\hat{a}_i}{b}+\frac{\hat{a}_i}{b}-\frac{a_i}{b}\right)c_i\right| \\
&= \left|\sum_{i\in[K]}\left(\frac{\hat{a}_i-a_i}{b}+\hat{a}_i\left(\frac{1}{\hat{b}}-\frac{1}{b}\right)\right)c_i\right| \\
&\leqslant \left|\sum_{i\in[K]}\left(\frac{\hat{a}_i-a_i}{b}\right)c_i\right|+\left|\sum_{i\in[K]}\left(\hat{a}_i\left(\frac{1}{\hat{b}}-\frac{1}{b}\right)\right)c_i\right| \\
&= \left|\sum_{i\in[K]}\frac{a_i}{b}\left(\frac{\hat{a}_i}{a_i}-1\right)c_i\right|+\left|\frac{b-\hat{b}}{b}\sum_{i\in[K]}\frac{\hat{a}_i}{\hat{b}}c_i\right| \\
&= \left|\sum_{i\in[K]}\frac{a_i}{b}\left(\frac{\hat{a}_i}{a_i}-1\right)c_i\right|+\left|\left(\sum_{i\in[K]}\frac{a_i}{b}\left(\frac{\hat{a}_i}{a_i}-1\right)\right)\left(\sum_{i\in[K]}\frac{\hat{a}_i}{\hat{b}}c_i\right)\right|.
\end{aligned} \tag{6.5}$$

Now we define the following: $L := \|x-y_z\|_2^2$ with $z \in \arg\min_j \|x-y_j\|_2^2$ is the smallest distance to a class mean, and $L+E_i := \|x-y_i\|_2^2$ with $E_i > 0$. Similarly, define $\hat{L} := \|x-\hat{y}_z\|_2^2$ with $z \in \arg\min_j \|x-\hat{y}_j\|_2^2$ is the smallest distance to an estimated class mean, and $\hat{L}+\hat{E}_i := \|x-\hat{y}_i\|_2^2$ with $\hat{E}_i > 0$. Finally, we define $T := \sqrt{\hat{L}}+\sqrt{\hat{L}+\hat{E}_i}+\sqrt{L}+\sqrt{L+E_i}$.

Then we can bound the parts separately:

For $i \neq z$, we have

$$\begin{aligned}
\frac{a_i}{b} &= \left(\frac{\exp\{-\frac{1}{2}\|x-y_i\|_2^2\}}{\sum_{j\in[K]}\exp\{-\frac{1}{2}\|x-y_j\|_2^2\}}\right) \\
&\leqslant \left(\frac{\exp\{-\frac{1}{2}(L+E_i)\}}{\exp\{-\frac{1}{2}L\}+\exp\{-\frac{1}{2}(L+E_i)\}}\right) \\
&= \frac{1}{\exp\{\frac{1}{2}E_i\}}
\end{aligned} \tag{6.6}$$

and in the case of $i = z$, we bound $\frac{a_z}{b} \leqslant 1$. So overall, we have $\frac{a_i}{b} \leqslant \frac{1}{\exp\{\mathbf{1}_{i\neq z}\frac{1}{2}E_i\}}$ for all $i \in [K]$.

$$\begin{aligned} c_i &= \|y_* - y_i\|_2^2 \\ &\leqslant 2\|x - y_i\|_2^2 + 2\|x - y_*\|_2^2 \\ &= 2(L + E_i + \|x - y_*\|_2^2). \end{aligned} \tag{6.7}$$

$$\begin{aligned} \frac{\hat{a}_i}{a_i} - 1 &= \exp\left\{\frac{1}{2}\|x - y_i\|_2^2 - \frac{1}{2}\|x - \hat{y}_i\|_2^2\right\} - 1 \\ &\approx 1 + \frac{1}{2}\|x - y_i\|_2^2 - \frac{1}{2}\|x - \hat{y}_i\|_2^2 - 1 \\ &= \frac{1}{2}\langle \hat{y}_i - y_i, 2x - y_i - \hat{y}_i\rangle \end{aligned} \tag{6.8}$$

$$\begin{aligned} &\leqslant \frac{1}{2}\|\hat{y}_i - y_i\| \cdot \|2x - y_i - \hat{y}_i\| \\ &\leqslant \|\hat{y}_i - y_i\| \left(\|x - y_i\| + \|x - \hat{y}_i\|\right) \\ &= \|\hat{y}_i - y_i\| \left(\|x - y_i\| + \|x - y_i + y_i - \hat{y}_i\|\right) \\ &\leqslant \|\hat{y}_i - y_i\| \left(2\|x - y_i\| + \|y_i - \hat{y}_i\|\right) \\ &= \|\hat{y}_i - y_i\| \left(2\sqrt{L + E_i} + \|y_i - \hat{y}_i\|\right). \end{aligned} \tag{6.9}$$

Next, we must control $E_i - \hat{E}_i$ in order to obtain the bound for $\frac{\hat{a}_i}{\hat{b}}$.

$$\begin{aligned}
E_i - \hat{E}_i &= (\|x - y_i\|^2 - L) - (\|x - \hat{y}_i\|^2 - \hat{L}) \\
&= \|x - y_i\|^2 - \|x - \hat{y}_i\|^2 + \|x - \hat{y}_z\|^2 - \|x - y_z\|^2 \\
&= \langle \hat{y}_i - y_i, 2x - y_i - \hat{y}_i \rangle + \langle y_z - \hat{y}_z, 2x - \hat{y}_z - y_z \rangle \\
&\leqslant \|\hat{y}_i - y_i\| \cdot \|2x - y_i - \hat{y}_i\| + \|y_z - \hat{y}_z\| \cdot \|2x - \hat{y}_z - y_z\| \\
&\leqslant \|y_i - \hat{y}_i\| \left(\|x - \hat{y}_i\| + \|x - y_i\|\right) + \|\hat{y}_z - y_z\| \left(\|x - y_z\| + \|x - \hat{y}_z\|\right) \\
&\leqslant c\sqrt{\frac{d \log K/\delta}{n}} \left(\|x - \hat{y}_i\| + \|x - y_i\| + \|x - y_z\| + \|x - \hat{y}_z\|\right) \\
&= c\sqrt{\frac{d \log K/\delta}{n}} \left(\sqrt{\hat{L}} + \sqrt{\hat{L} + \hat{E}_i} + \sqrt{L} + \sqrt{L + E_i}\right) \\
\hat{E}_i &\geqslant \max\left\{0, E_i - cT\sqrt{\frac{d \log K/\delta}{n}}\right\}.
\end{aligned}$$

Using (6.6), we obtain

$$\begin{aligned}
\frac{\hat{a}_i}{\hat{b}} &\leqslant \frac{1}{\exp\{\mathbf{1}_{i \neq z} \frac{1}{2} \hat{E}_i\}} \\
&\leqslant \frac{1}{\exp\left\{\mathbf{1}_{i \neq z} \max\left\{0, \frac{1}{2} E_i - \frac{cT}{2}\sqrt{\frac{d \log K/\delta}{n}}\right\}\right\}}
\end{aligned} \tag{6.10}$$

Plugging (6.6), (6.7), (6.9), and (6.10) into (6.5), we obtain

$$\left|\sum_{i \in [K]} \frac{a_i}{b}\left(\frac{\hat{a}_i}{a_i} - 1\right) c_i\right| + \left|\left(\sum_{i \in [K]} \frac{a_i}{b}\left(\frac{\hat{a}_i}{a_i} - 1\right)\right)\left(\sum_{i \in [K]} \frac{\hat{a}_i}{\hat{b}} c_i\right)\right|$$

$$
\begin{aligned}
&\leqslant \sum_{i\in[K]} \frac{\|\hat{y}_i - y_i\| \left(2\sqrt{L+E_i} + \|y_i - \hat{y}_i\|\right)}{\exp\{\mathbf{1}_{i\neq z}\frac{1}{2}E_i\}} 2(L + E_i + \|x - y_*\|_2^2) \\
&\quad + \left(\sum_{i\in[K]} \frac{\|\hat{y}_i - y_i\| \left(2\sqrt{L+E_i} + \|y_i - \hat{y}_i\|\right)}{\exp\{\mathbf{1}_{i\neq z}\frac{1}{2}E_i\}}\right) \\
&\quad \cdot \left(\sum_{i\in[K]} \frac{2(L + E_i + \|x - y_*\|_2^2)}{\exp\left\{\mathbf{1}_{i\neq z}\max\left\{0, \frac{1}{2}E_i - \frac{cT}{2}\sqrt{\frac{d\log K/\delta}{n}}\right\}\right\}}\right) \\
&\leqslant O\left(\sqrt{\frac{d\log K/\delta}{n}} \sum_{i\in[K]} \frac{L + E_i + \|x - y_*\|_2^2}{\exp\left\{\mathbf{1}_{i\neq z}\frac{1}{2}E_i\right\}}\right)
\end{aligned}
$$

Next, recalling the fact that our class means form an ε-net on the radius-R ball, we use Lemma 5.2 from [Vershynin, 2010] to bound L as $L \leqslant \frac{2R}{\sqrt{d}K-1}$.

$$
\leqslant O\left(\sqrt{\frac{d\log K/\delta}{n}} \sum_{i\in[K]} \frac{\frac{R}{\sqrt{d}K} + E_i + \|x - y_*\|_2^2}{\exp\left\{\mathbf{1}_{i\neq z}\frac{1}{2}E_i\right\}}\right). \tag{6.11}
$$

Next, we will consider cases on E_i. We will consider the cases in which $E_i < \log R^2$ and $E_i \geqslant \log R^2$.

We will begin with the case in which $E_i < \log R^2$, then Equation 6.11 becomes:

$$
\leqslant O\left(\sqrt{\frac{d\log K/\delta}{n}} \left(\sum_{i\in[K]} \frac{R}{\sqrt{d}K} + \|x - y_*\|_2^2\right)\right).
$$

Next, the case in which $E_i \geqslant \log R^2$, then Equation 6.11 becomes:

$$\begin{aligned}
&\leqslant O\left(\sqrt{\frac{d\log K/\delta}{n}} \sum_{i\in[K]} \frac{\frac{R}{\sqrt{d}K} + 2\log R + \|x - y_*\|_2^2}{R}\right) \\
&\leqslant O\left(\sqrt{\frac{d\log K/\delta}{n}} \left(\sum_{i\in[K]} \frac{1}{\sqrt{d}K} + \|x - y_*\|_2^2\right)\right).
\end{aligned}$$

Setting $M \leqslant K$ to be the number of terms for which $E_i < \log R^2$ holds and by combining the two bounds, we obtain the following:

$$\leqslant O\left(\sqrt{\frac{d\log K/\delta}{n}} \left(\left(\frac{MR}{\sqrt{d}K} + \frac{K-M}{\sqrt{d}K}\right) + K\|x - y_*\|_2^2\right)\right).$$

Ultimately, our bound is

$$|\Psi_{\widehat{\mathbb{P}}}(y_*) - \Psi_{\mathbb{P}}(y_*)| \leqslant O\left(\sqrt{\frac{d\log K/\delta}{n}} \left(\left(\frac{MR}{\sqrt{d}K} + \frac{K-M}{\sqrt{d}K}\right) + K\|x - y_*\|_2^2\right)\right). \tag{6.12}$$

Next, we will obtain a lower bound on $\Psi_{\widehat{\mathbb{P}}}(\hat{y}_*) - \Psi_{\mathbb{P}}(y_*)$.

$$\begin{aligned}
|\Psi_{\widehat{\mathbb{P}}}(\hat{y}_*) - \Psi_{\mathbb{P}}(y_*)| &\geqslant |\Psi_{\mathbb{P}}(y_*) - \Psi_{\mathbb{P}}(\hat{y}_*)| - |\Psi_{\hat{\mathbb{P}}}(\hat{y}_*) - \Psi_{\mathbb{P}}(\hat{y}_*)| && \text{triangle inequality} \\
&\geqslant \alpha d^2(y_*, \hat{y}_*) - |\Psi_{\widehat{\mathbb{P}}}(y_*) - \Psi_{\mathbb{P}}(y_*)| && \text{Assumption 6.1.}
\end{aligned}$$

Combining both of these bounds with Equation 6.2, we obtain

$$\begin{aligned}
\alpha d^2(y_*, \hat{y}_*) - |\Psi_{\widehat{\mathbb{P}}}(y_*) - \Psi_{\mathbb{P}}(y_*)| \leqslant \Psi_{\widehat{\mathbb{P}}}(\hat{y}_*) - \Psi_{\mathbb{P}}(y_*) &\leqslant \Psi_{\widehat{\mathbb{P}}}(y_*) - \Psi_{\mathbb{P}}(y_*) \\
&\leqslant |\Psi_{\widehat{\mathbb{P}}}(y_*) - \Psi_{\mathbb{P}}(y_*)|
\end{aligned}$$

$$
\begin{aligned}
&\Rightarrow \alpha d^2(y_*, \hat{y}_*) \leqslant 2|\Psi_{\hat{\mathbb{P}}}(y_*) - \Psi_{\mathbb{P}}(y_*)| \\
&\Rightarrow d^2(y_*, \hat{y}_*) \leqslant O\left(\frac{1}{\alpha}\sqrt{\frac{d\log K/\delta}{n}}\left(\frac{MR}{\sqrt{d}K} + \frac{K-M}{\sqrt{d}K} + K\|x - y_*\|_2^2\right)\right) \\
&\Rightarrow \mathbb{E}_{(x,y_*)\sim\mathcal{D}_{\text{test}}}[d^2(y_*, \hat{y}_*)] \leqslant O\left(\frac{1}{\alpha}\sqrt{\frac{d\log K/\delta}{n}}\left(\frac{MR}{\sqrt{d}K} + \frac{K-M}{\sqrt{d}K} + dK\right)\right),
\end{aligned}
$$

Next, we obtain a bound on M. Since $M \leqslant K$ is the number of terms for which $E_i < \log R^2$, we have that $M = VK$, where $V \in [0,1]$ is the ratio of the volumes of the ball for which $E_i < \log R^2$, and the ball containing the entire set of classes $\mathcal{Y}$. We have

$$
\begin{aligned}
E_i = \|x - y_i\|_2^2 - \|x - y_z\|_2^2 &< \log R^2 \\
\|x - y_i\|_2^2 &< \log R^2 + L \\
&\leqslant \log R^2 + \frac{2R}{\sqrt{d}K - 1} \leqslant R^2 \\
\Rightarrow \sqrt{\log R^2 + \frac{2R}{\sqrt{d}K - 1}} &\leqslant R,
\end{aligned}
$$

which is an upper bound of the radius of the ball for which $E_i < \log R^2$ is true. Now we construct V:

$$
V = \left(\frac{\sqrt{\log R^2 + \frac{2R}{\sqrt{d}K-1}}}{R}\right)^d.
$$

Combining this with our error bound, we have

$$
\begin{aligned}
&\mathbb{E}_{(x,y_*)\sim\mathcal{D}_{\text{test}}}[d^2(y_*, \hat{y}_*)] \leqslant \\
&O\left(\frac{1}{\alpha}\sqrt{\frac{d\log K/\delta}{n}}\left(\frac{\left(\frac{\sqrt{\log R^2 + \frac{2R}{\sqrt{d}K-1}}}{R}\right)^d R}{\sqrt{d}} + \frac{1 - \left(\frac{\sqrt{\log R^2 + \frac{2R}{\sqrt{d}K-1}}}{R}\right)^d}{\sqrt{d}} + dK\right)\right).
\end{aligned}
$$

However, we can choose K to weaken our dependence on R.

$$V = \left(\frac{\sqrt{\log R^2 + \frac{2R}{\sqrt{d}K-1}}}{R} \right)^d \leqslant \frac{1}{R}$$
$$\Rightarrow K \geqslant \frac{2}{\sqrt{d}\left(R^{1-\frac{2}{d}} - \frac{\log R}{R}\right)} + 1.$$

Finally, we have

$$\begin{aligned}
\mathbb{E}_{(x,y_*)\sim\mathcal{D}_{\text{test}}}[d^2(y_*, \hat{y}_*)] &\leqslant O\left(\frac{1}{\alpha}\sqrt{\frac{d\log K/\delta}{n}}\left(\frac{1}{\sqrt{d}} + \frac{1-\frac{1}{R}}{\sqrt{d}} + \frac{\sqrt{d}}{\left(R^{1-\frac{2}{d}} - \frac{\log R}{R}\right)} + d\right)\right) \\
&\leqslant O\left(\frac{1}{\alpha}\sqrt{\frac{d\log K/\delta}{n}}\left(\frac{1}{\sqrt{d}} + \frac{\sqrt{d}}{\left(R^{1-\frac{2}{d}} - \frac{\log R}{R}\right)} + d\right)\right) \\
&\leqslant O\left(\frac{d}{\alpha}\sqrt{\frac{\log K/\delta}{n}}\left(\frac{1}{\left(R^{1-\frac{2}{d}} - \frac{\log R}{R}\right)} + \sqrt{d}\right)\right)
\end{aligned}$$

which completes the proof. □

Discussion of Theorem 6.10. The above bound contains several standard quantities, including the dimensionality of the inputs d, the radius R of the ball from which labels are drawn, the number of classes and samples used to estimate the model K and n respectively, and δ, which is used to control the probability that the bound holds. Naturally, the bound scales up with higher d and K, and improves with an increased number of samples n. The dependence on R is also related to its dependence on d — so long as we have that $K \geqslant 2(\sqrt{d}(R^{1-\frac{2}{d}} - \log R/R))^{-1} + 1$ and for $d \geqslant 2$, R increases, which improves the bound. The bound includes a metric space dependent quantity α, which is related to the amount that the Fréchet variance can change subject to changes in its argument, i.e., which node is considered.

6.A.2 Proof of Theorem 6.11 (minimum locus cover for trees)

Theorem 6.11 (Minimum locus cover for trees)**.** *The minimum locus cover for a tree graph* T *is* $\Lambda = \mathit{Leaves}(\mathsf{T})$.

Proof. We would like to show to show two things:

1. any node can be resolved by an appropriate construction of $\boldsymbol{w}$ using only the leaves and
2. removing any leaf makes that leaf node unreachable, no matter what the other nodes are in $\boldsymbol{y}$–i.e., we must use at least all of the leaf nodes.

If both of these conditions hold (i.e., that the leaves form a locus cover and that they are the smallest such set), then the the leaves of any tree form a minimum locus cover. To set up the problem, we begin with some undirected tree, T, whose nodes are our set of labels: $\mathsf{T} = (\mathcal{Y}, \mathsf{E})$ where $\Lambda = \text{Leaves}(\mathsf{T}) \subseteq \mathcal{Y}$ is the set of leaf nodes. Moreover, let $\boldsymbol{\Lambda}$ be a tuple of the leaf nodes. We will start by proving that Λ is a locus cover. We will show this by cases on $v \in \mathcal{Y}$, the node that we would like to resolve:

1. If $v \in \Lambda$, i.e. v is a leaf node, then setting $\boldsymbol{w}_i = \mathbf{1}\{\boldsymbol{\Lambda}_i = v\}$ yields

$$\begin{aligned} m_{\boldsymbol{\Lambda}}(\boldsymbol{w}) &= \arg\min_{y \in \mathcal{Y}} \sum_{i=1}^{K} \mathbf{1}\{\boldsymbol{\Lambda}_i = v\} d^2(y, \boldsymbol{\Lambda}_i) \\ &= \arg\min_{y \in \mathcal{Y}} d^2(y, v) = \{v\}. \end{aligned}$$

2. If $v \notin \Lambda$, we have a bit more work to do. Since v is an internal node, consider any pair of leaves, $l_1 \neq l_2$ such that v is along the (unique) path between l_1 and l_2: $v \in \Gamma(l_1, l_2)$.

$$\text{Then set } \boldsymbol{w}_i = \begin{cases} \frac{d(v,l_2)}{d(l_1,l_2)} \text{ if } \boldsymbol{\Lambda_i} = l_1, \\ \frac{d(v,l_1)}{d(l_1,l_2)} \text{ if } \boldsymbol{\Lambda_i} = l_2, \\ 0 \text{ otherwise} \end{cases} \quad \text{yields}$$

$$\begin{aligned} m_{\boldsymbol{\Lambda}}(\boldsymbol{w}) &= \arg\min_{y \in \mathcal{Y}} \frac{d(v,l_2)d^2(y,l_1) + d(v,l_1)d^2(y,l_2)}{d(l_1,l_2)} \\ &= \arg\min_{y \in \mathcal{Y}} d(v,l_2)d^2(y,l_1) + d(v,l_1)d^2(y,l_2) \\ &= \{v\}. \end{aligned}$$

Thus Λ is a locus cover. Next, we will show that it is the smallest such set. Let $l \in \Lambda$ be any leaf node, and define $\Lambda' = \Lambda \setminus \{l\}$. We must show that Λ' is not a locus cover. Assume for contradiction that Λ' is a locus cover. This means that given some tuple $\boldsymbol{y}'$ whose entries are the elements of Λ', we have $\Pi(\boldsymbol{\Lambda}') = \mathcal{Y}$. This implies that the path between any two nodes in Λ is also contained in $\Pi(\boldsymbol{\Lambda}')$. Since the leaves form a locus cover and any $y \in \mathcal{Y}$ can be constructed by the appropriate choice of $\boldsymbol{w}$ along with a path between two leaf nodes, l must either be one of the entries of $\boldsymbol{\Lambda}'$ (one of the endpoints of the path) or it must be an internal node. Both cases are contradictions–$\boldsymbol{\Lambda}'$ cannot include l by assumption, and l is assumed to be a leaf node. It follows that Λ is a minimum locus cover. □

6.A.3 Proof of Lemma 6.12 (tree pairwise decomposability)

Lemma 6.12 (Tree pairwise decomposability)**.** *Let $T = (\mathcal{Y}, \mathcal{E})$ be a tree and $\Lambda \subseteq \mathcal{Y}$. Then $\Pi(\Lambda)$ is pairwise decomposable.*

Proof. Assume for contradiction that $\exists y^* \in \mathcal{Y}$ where $y^* \notin \Pi(\{\lambda_i, \lambda_j\})$, $\forall \lambda_i, \lambda_j \in \Lambda$, but $y^* \in \Pi(\Lambda)$. Then, $y^* \in m_\Lambda(\mathbf{w})$ for some $\mathbf{w}$. Note that $y^* \notin \Pi(\{\lambda_i, \lambda_j\})$ implies $y^* \notin \Gamma(\lambda_i, \lambda_j)$, $\forall \lambda_i, \lambda_j \in \Lambda$, because $\Pi(\{\lambda_i, \lambda_j\}) = \Gamma(\lambda_i, \lambda_j)$ due to the uniqueness of paths in trees. As such,

$\cap_{\lambda\in\Lambda}\Gamma(y^*,\lambda)$ must contain an immediate relative y' of y^* where

$$\sum_{i=1}^{|\Lambda|} w_i d^2(y^*,\lambda_i) = \sum_{i=1}^{|\Lambda|} w_i (d(y',\lambda_i)+1)^2 > \sum_{i=1}^{|\Lambda|} w_i d^2(y',\lambda_i).$$

Therefore, $y^* \notin \Pi(\Lambda)$ because y^* is not a minimizer of $\sum_{i=1}^{|\Lambda|} w_i d^2(y,\lambda_i)$. So, it must be that $y^* \in \Pi(\{\lambda_i,\lambda_j\})$ for some $\lambda_i,\lambda_j \in \Lambda$ when $y \in \Pi(\Lambda)$. □

6.A.4 Proof of Theorem 6.13 (Algorithm 4 correctness)

Theorem 6.13 (Algorithm 4 correctness)**.** *Algorithm 4 returns a locus cover for phylogenetic trees.*

Proof. We first prove that Algorithm 4 will halt, then according to the stopping criterion, it is guaranteed that Algorithm 4 returns a locus cover. Suppose the algorithm keeps running until it reaches the case in which all of the leaf nodes are included in Λ (i.e. $\Lambda = \mathcal{Y}$), this results in the trivial locus cover of the phylogenetic tree and the algorithm halts. □

6.A.5 Proof of Lemma 6.14 (phylogenetic tree pairwise decomposability)

Lemma 6.14 (Phylogenetic tree pairwise decomposability)**.** *Let $T = (\mathcal{V},\mathcal{E})$ be a tree with $\mathcal{Y} = Leaves(T)$ and $\Lambda \subseteq \mathcal{Y}$. Then $\Pi(\Lambda)$ is pairwise decomposable.*

Proof. Suppose to reach a contradiction that $\exists y^* \in \mathcal{Y}$ where $y^* \in \Pi(\Lambda)$, but $y^* \notin \Pi(\{\lambda_i,\lambda_j\})$ $\forall \lambda_i,\lambda_j \in \Lambda$. In other words, y^* is a Fréchet mean for some weighting on Λ, but y^* is not a Fréchet mean for any weighting for any two vertices within Λ.

For arbitrary $\lambda_i,\lambda_j \in \Lambda$, define the closest vertex to y^* on the path between λ_i and λ_j as

$$v'(\lambda_i,\lambda_j) = \operatorname*{arg\,min}_{v\in\Gamma(\lambda_i,\lambda_j)} d(y^*,v).$$

Also, let

$$\{\lambda_1, \lambda_2\} = \arg\min_{\lambda_i, \lambda_j \in \Lambda} d(y^*, v'(\lambda_i, \lambda_j)),$$

the pair of vertices in Λ whose path $\Gamma(\lambda_1, \lambda_2)$ passes *closest* to y^*. To avoid notational clutter, define $v' = v'(\lambda_1, \lambda_2)$. Thus, v' represents the closest vertex to y^* lying on the path between any two $\lambda_i, \lambda_j \in \Lambda$.

Because v' lies on the path $\Gamma(\lambda_1, \lambda_2)$, there exists some weighting $\mathbf{w}$ for which

$$v' = \arg\min_{v \in \Gamma(\lambda_1, \lambda_2)} (\mathbf{w}_1 d^2(v, \lambda_1) + \mathbf{w}_2 d^2(v, \lambda_2)).$$

For this $\mathbf{w}$, we have that $m_{\{\lambda_1, \lambda_2\}}(\mathbf{w}) = \arg\min_{y \in \mathcal{Y}} d(y, v')$, the set of leaf nodes of smallest distance from v'. By the initial assumption that $y^* \notin \Pi(\{\lambda_1, \lambda_2\})$, we have that $y^* \notin m_{\{\lambda_1, \lambda_2\}}(\mathbf{w})$. Let y' be a vertex of $m_{\{\lambda_i, \lambda_j\}}(\mathbf{w})$ that is closest to y^*. Clearly, $d(y', v')^2 < d(y^*, v')^2$.

For any $v \in \mathcal{V}$, respectively define the sets of vertices (1) shared in common on the two paths $\Gamma(v, y^*)$ and $\Gamma(v, y')$, (2) on the path $\Gamma(v, y^*)$ that are not on $\Gamma(v, y')$, and (3) on the path $\Gamma(v, y')$ that are not on $\Gamma(v, y^*)$ as

$$\begin{aligned} d_C(v) &= \Gamma(v, y^*) \cap \Gamma(v, y'), \\ d_{y^*}(v) &= \Gamma(v, y^*) \setminus \Gamma(v, y'), \text{ and} \\ d_{y'}(v) &= \Gamma(v, y') \setminus \Gamma(v, y^*). \end{aligned}$$

We have that $|d_{y^*}(v)| + |d_{y^*}(v)| + 1 = |\Gamma(y^*, y')|$ (the difference by 1 represents the final vertex shared in common by both paths) and also that $|d_{y'}(v')| < |d_{y^*}(v')|$ since y' is closer to v' than y^* is.

Suppose $\exists \lambda_3, \lambda_4 \in \Lambda$ yielding

$$u' = v'(\lambda_3, \lambda_4) \in \Gamma(y^*, y') \text{ such that } d^2(y^*, u') \leqslant d^2(y', u').$$

This is to say that u' is defined analogously to v', with the additional restrictions that u' lie on the path $\Gamma(y^*, y')$ and u' be at least as close to y^* as to y'. Then, $|d_{y^*}(u')| < |d_{y^*}(v')|$ implying $d^2(y^*, u') < d^2(y^*, v')$, which contradicts the definition of v'.

Suppose now that $\exists \lambda_3, \lambda_4 \in \Lambda$ yielding

$$u' = v'(\lambda_3, \lambda_4) \notin \Gamma(y^*, y') \text{ such that } d^2(y^*, u') \leqslant d^2(y', u').$$

This is to say that u' is again defined analogously to v', but u' is not on $\Gamma(y^*, y')$ and is at least as close to y^* as to y'. Then, either $\cap_{\lambda \in \Lambda} d_C(\lambda) \neq \emptyset$ (i.e. all lambdas lie in the same branch off of $\Gamma(y', y^*)$), in which case the path between any two lambdas will have the same closest node v' to y^*, or it is possible to choose a $\lambda_5 \in \Lambda$ such that $d_C(\lambda_3) \cap d_C(\lambda_5) = \emptyset$. In this situation, it must be that $\Gamma(\lambda_3, \lambda_5) \cap \Gamma(y', y^*) \neq \emptyset$, which implies there exists a vertex $t \in \Gamma(\lambda_3, \lambda_5)$ such that $d^2(y^*, t) < d^2(y^*, u')$ which violates the definition of u'.

Altogether, for all $v \in \Gamma(\lambda_i, \lambda_j)$ with arbitrary $\lambda_i, \lambda_j \in \Lambda$, we have that $d^2(y', v) < d^2(y^*, v)$. This implies that for all $\lambda \in \Lambda$, there exists a $y' \in \mathcal{Y}$ such that $d^2(y', \lambda) < d^2(y^*, \lambda)$ and therefore that $y^* \notin \Pi(\Lambda)$ which contradicts that $y^* \in \Pi(\Lambda)$. It must be that $\forall y^* \in \mathcal{Y}$ where $y^* \in \Pi(\Lambda)$, we have $y^* \in \Pi(\{\lambda_i, \lambda_j\})$ $\forall \lambda_i, \lambda_j \in \Lambda$. □

6.A.6 Proof of Theorem 6.15 (minimum locus cover for grid graphs)

Theorem 6.15 (Minimum locus cover for the grid graphs)**.** *The minimum locus cover for a grid graph is the pair of vertices in the furthest opposite corners.*

Proof. We would like to show two things:

1. any vertex can be resolved by construction of $\boldsymbol{w}$ using the furthest pair of two vertices (i.e. opposite corners) and

2. there does not exist a locus cover of size one for grid graphs with more than one vertex.

Let $G = (\mathcal{V}, \mathcal{E})$ be a grid graph and let $\Lambda = \{\lambda_1, \lambda_2\}$ be a set of two vertices who are on opposite corners of G (i.e., perhipheral vertices achieving the diameter of G). For notational convenience, set $\boldsymbol{\Lambda} = (\lambda_1, \lambda_2)$ We can reach any interior vertex by an appropriate setting the weight vector $\boldsymbol{w} = [\boldsymbol{w}_1, \boldsymbol{w}_2]$. Then set $\boldsymbol{w} = \left(\frac{d(v, \lambda_2)}{d(\lambda_1, \lambda_2)}, \frac{d(v, \lambda_1)}{d(\lambda_1, \lambda_2)} \right)$. Finally, the Fréchet mean

is given by

$$\begin{aligned} \mathfrak{m}_{\boldsymbol{\Lambda}}(\boldsymbol{w}) &= \arg\min_{y \in \mathcal{Y}} \frac{d(v,\lambda_2)d^2(y,\lambda_1) + d(v,\lambda_1)d^2(y,\lambda_2)}{d(\lambda_1,\lambda_2)} \\ &= \arg\min_{y \in \mathcal{Y}} d(v,\lambda_2)d^2(y,\lambda_1) + d(v,\lambda_1)d^2(y,\lambda_2) \\ &\ni v. \end{aligned}$$

Hence Λ is a locus cover. We can clearly see that Λ is a minimum locus cover—the only way to obtain a smaller set Λ' is for it to include only a single vertex. However, if $\Lambda' = \{v'\}$ contains only a single vertex, it cannot be a locus cover so long as G contains more than one vertex—v' is always guaranteed to be a unique minimizer of the Fréchet mean under Λ' which misses all other vertices in G. Thus Λ is a minimum locus cover. □

6.A.7 Proof of Lemma 6.16 (loci of grid subspaces)

Lemma 6.16 (Locus of grid subspaces)**.** *Given any pair of vertices in Λ, we can find a subset G' of the original grid graph $G = (\mathcal{Y}, \mathcal{E})$ which takes the given pair as two corner. $\Pi(\boldsymbol{\Lambda})$ equals to all the points inside G'.*

Proof. The result follows from application of Theorem 6.15 to the choice of metric subspace. □

6.A.8 Proof of Lemma 6.17 (grid pairwise decomposability)

Lemma 6.17 (Grid pairwise decomposability)**.** *Let $G = (\mathcal{Y}, \mathcal{E})$ be a grid graph and $\Lambda \subseteq \mathcal{Y}$. Then $\Pi(\Lambda)$ is pairwise decomposable.*

Proof. Suppose we have a grid graph $G = (\mathcal{Y}, \mathcal{E})$ and a vertex $\hat{y} \in \Pi(\Lambda)$ with $\Lambda \subseteq \mathcal{Y}$. Due to the fact that $\hat{y} \in \Pi(\Lambda)$ and the fact that G is a grid graph, we have that $\hat{y} \in \Gamma(\lambda_\alpha, \lambda_\beta)$ for some $\lambda_\alpha, \lambda_\beta \in \Lambda$. Then by Lemma 6.17,

$$\hat{y} \in \Pi(\{\lambda_\alpha, \lambda_\beta\}) \subseteq \cup_{\lambda_i, \lambda_j \in \Lambda} \Pi(\{\lambda_i, \lambda_j\}).$$

Therefore loci on grid graphs are pairwise decomposable. □

6.A.9 Proof of Theorem 6.18 (no nontrivial locus covers for complete graphs)

Theorem 6.18 (Trivial locus cover for the complete graph)**.** *There is no non-trivial locus cover for the complete graph.*

Proof. We show that there is no nontrivial locus cover for complete graphs by showing that removing any vertex from the trivial locus cover renders that vertex unreachable. We proceed by strong induction on the number of vertices, n.
Base case We first set $n = 3$. Let $\boldsymbol{K_3} = (\mathcal{V}, \mathcal{V}^2)$ be the complete graph with three vertices: $\mathcal{V} = \{v_1, v_2, v_3\}$, and without loss of generality, let $\Lambda = \{v_2, v_3\}$ be our set of observed classes. There are two cases on the weight vector $\boldsymbol{w} = [w_2, w_3]$:
Case 1: Suppose $\boldsymbol{w} \notin \text{int}\Delta^2$. This means that either $w_2 = 1$ or $w_3 = 1$—which leads to the Fréchet mean being either v_2 or v_3, respectively. Neither of these instances correspond to v_1 being a minimizer.
Case 2: Suppose $\boldsymbol{w} \in \text{int}\Delta^2$. Then the Fréchet mean is given by

$$m_{\boldsymbol{\lambda}}(\boldsymbol{w}) = \underset{y \in \mathcal{Y}}{\arg\min}\, w_2 d^2(y, v_2) + w_3 d^2(y, v_3)$$

Assume for contradiction that $v_1 \in \Pi(\boldsymbol{\Lambda})$:

$$\begin{aligned} w_2 d^2(v_1, v_2) + w_3 d^2(v_1, v_3) &= w_2 + w_3 \\ &> w_3 && \text{because } \boldsymbol{w} \in \text{int}\Delta^2 \\ &= w_2 d^2(v_2, v_2) + w_3 d^2(v_2, v_3). \end{aligned}$$

Therefore, $v_1 \notin \Pi(\boldsymbol{\Lambda})$. This is a contradiction. Thus there is no nontrivial locus cover for $\boldsymbol{K_3}$.

Inductive step: Let $\boldsymbol{K_{n-1}} = (\mathcal{V}, \mathcal{V}^2)$ be the complete graph with $n-1$ vertices. Assume that there is no nontrivial locus cover for $\boldsymbol{K_{n-1}}$. We will show that there is no nontrivial locus cover for $\boldsymbol{K_n}$. Let the weight vector be $\boldsymbol{w} = [w_1, ...w_{n-1}]$ corresponding to vertices $v_1, ..., v_{n-1} \in \mathcal{V}$ with $\Lambda = \{v_1, ..., v_{n-1}\}$. We want to show that $v_n \notin \Pi(\boldsymbol{\Lambda})$. We proceed by cases on $\boldsymbol{w}$.
Case 1: Suppose $\boldsymbol{w} \notin \text{int}\Delta^{n-2}$ where m entries of $\boldsymbol{w}$ are zero, then by strong induction we know that $v_n \notin \Pi(\boldsymbol{\Lambda}) = \{v_i\}_{i=1}^{n-m}$, i.e., there is no nontrivial locus cover for $\boldsymbol{K_{n-m}}$.

Case 2: Suppose $\boldsymbol{w} \in \text{int}\Delta^{n-2}$.
Assume for contradiction that $v_n \in \Pi(\boldsymbol{\Lambda})$. Using this assumption, we obtain the following

$$\begin{aligned}\sum_{i=1}^{n-1} w_i d^2(v_n, v_i) &= \sum_{i=1}^{n-1} w_i \\ &> \sum_{i=1}^{n-2} w_i && \text{because } \boldsymbol{w} \in \text{int}\Delta^{n-2} \\ &= \sum_{i=1}^{n-1} w_i d^2(v_{n-1}, v_i).\end{aligned}$$

Therefore, v_n is not a minimizer, so $v_n \notin \Pi(\Lambda)$. This is a contradiction, hence $\boldsymbol{K_n}$ has no nontrivial locus cover. □

6.A.10 Proof of Theorem 6.8 (active next-class selection for trees)

Proof. We will prove the result by showing that the two optimization problems are equivalent. Let v be a solution to the following optimization problem:

$$\begin{aligned}\underset{y \in \mathcal{Y} \setminus \Pi(\Lambda)}{\arg\max} \quad & d(y, b) \\ \text{s.t.} \quad & b \in \partial_{\text{in}} T', \\ & \Gamma(b, y) \setminus \{b\} \subseteq \mathcal{Y} \setminus \Pi(\Lambda),\end{aligned}$$

where $\partial_{\text{in}} T'$ is the inner boundary of T', the subgraph of T whose vertices are $\Pi(\Lambda)$. This optimization problem can be equivalently rewritten as

$$\begin{aligned}\underset{y \in \mathcal{Y} \setminus \Pi(\Lambda)}{\arg\max} \quad & |\Gamma(y, b)| \\ \text{s.t.} \quad & b \in \partial_{\text{in}} T', \\ & \Gamma(b, y) \setminus \{b\} \subseteq \mathcal{Y} \setminus \Pi(\Lambda),\end{aligned}$$

and we can furthermore introduce additional terms that do not change the maximizer

$$\begin{aligned} \underset{y \in \mathcal{Y} \setminus \Pi(\Lambda)}{\arg\max} \quad & |\cup_{\lambda_i,\lambda_j \in \Lambda} \Gamma(\lambda_i,\lambda_j) \cup \Gamma(b,y)| \\ \text{s.t.} \quad & b \in \partial_{\text{in}} T', \\ & \Gamma(b,y) \setminus \{b\} \subseteq \mathcal{Y} \setminus \Pi(\Lambda). \end{aligned}$$

Equivalently, we can also connect b to one of the elements of Λ

$$\begin{aligned} \underset{y \in \mathcal{Y} \setminus \Pi(\Lambda)}{\arg\max} \quad & |\cup_{\lambda_i,\lambda_j \in \Lambda} \Gamma(\lambda_i,\lambda_j) \cup \Gamma(\lambda_i,b) \cup \Gamma(b,y)| \\ \text{s.t.} \quad & b \in \partial_{\text{in}} T', \\ & \Gamma(b,y) \setminus \{b\} \subseteq \mathcal{Y} \setminus \Pi(\Lambda). \end{aligned}$$

Due to the uniqueness of paths in trees, this optimization problem also has the following equivalent form without any dependence on b:

$$\begin{aligned} v \in \underset{y \in \mathcal{Y} \setminus \Lambda}{\arg\max} |\cup_{\lambda_i,\lambda_j \in \Lambda} \Gamma(\lambda_i,\lambda_j) \cup \Gamma(\lambda_i,y)| &= \underset{y \in \mathcal{Y} \setminus \Lambda}{\arg\max} |\cup_{\lambda_i,\lambda_j \in \Lambda} \Pi(\{\lambda_i,\lambda_j\}) \cup \Pi(\{\lambda_i,y\})| \\ &= \underset{y \in \mathcal{Y} \setminus \Lambda}{\arg\max} |\cup_{\lambda_i,\lambda_j \in \Lambda \cup \{y\}} \Pi(\{\lambda_i,\lambda_j\})| \\ &= \underset{y \in \mathcal{Y} \setminus \Lambda}{\arg\max} |\Pi(\Lambda \cup \{y\})| \\ &\quad \text{using Lemma 6.12.} \end{aligned}$$

Therefore v is a maximizer of $|\Pi(\Lambda \cup \{v\})|$, as required. □

6.B Algorithms and Time Complexity Analyses

We provide time complexity analyses for Algorithms 4, 5, and 6.

6.B.1 Analysis of Algorithm 4 (locus cover for phylogenetic trees)

We provide Algorithm 4 with comments corresponding to the time complexity of each step.

Algorithm 4 Locus cover for phylogenetic trees

Require: phylogenetic tree $T = (\mathcal{V}, \mathcal{E})$, $\mathcal{Y} = \text{Leaves}(T)$
$N \leftarrow |\mathcal{Y}|$
$P \leftarrow \text{sortbylength}([\Gamma(y_i, y_j)]_{i,j\in[N]})$ ▷ $N|\mathcal{E}| + N^2 \log N$
$P \leftarrow \text{reverse}(P)$ ▷ $O(N^2)$
$\Lambda \leftarrow \emptyset$
for $\Gamma(y_i, y_j)$ in P **do**
▷ $O(N^2)$ *iterations*
  if $\Pi(\Lambda) = \mathcal{Y}$ **then**
▷ $O(K^2 D \max\{N|\mathcal{E}|, N^2 \log N\})$
    return Λ
  else
    $\Lambda \leftarrow \Lambda \cup \{y_i, y_j\}$
  end if
end for

Combining these, we obtain the following time complexity:

$$O(N|\mathcal{E}| + N^2 \log N + N^2 + N^2 K^2 D \max\{N|\mathcal{E}|, N^2 \log N\}) = O(N^2 K^2 D \max\{N|\mathcal{E}|, N^2 \log N\}).$$

6.B.2 Analysis of Algorithm 5 (computing a pairwise decomposable locus)

We first provide Algorithm 5 here, with comments corresponding to the time complexity of each step.

We first compute the diameter of the graph $G = (\mathcal{Y}, \mathcal{E})$ with $N = |\mathcal{Y}|$, which is done in $O(N|\mathcal{E}| + N^2 \log N)$ time using Dijkstra's algorithm to compute the shortest paths between all pairs of vertices. We then iterate over all pairs of elements in Λ with $K = |\Lambda|$, which amounts to $O(K^2)$ iterations. Within this, we perform $O(D)$ computations of the Fréchet mean, for which each iteration requires $O(N|\mathcal{E}| + N^2 \log N)$ arithmetic operations or comparisons. Combining these, the final time complexity is

$$O(N|\mathcal{E}| + N^2 \log N + K^2 D(N|\mathcal{E}| + N^2 \log N)) = O(K^2 D \max\{N|\mathcal{E}|, N^2 \log N\}).$$

Algorithm 5 Computing a pairwise decomposable locus

Require: Λ, $\mathcal{Y}$, $G = (\mathcal{V}, \mathcal{E})$
$\Pi \leftarrow \emptyset$
$D \leftarrow \text{diam}(G)$ ▷ $O(N|\mathcal{E}| + N^2 \log N)$
for $\lambda_i, \lambda_j \in \Lambda$ **do**
  for w_1 in $\left\{\frac{0}{D}, \frac{1}{D}, ..., \frac{D}{D}\right\}$ **do**
    ▷ $O(D)$ *iterations*
    $\mathbf{w} \leftarrow [w_1, 1 - w_1]$
    $\Pi \leftarrow \Pi \cup m_{\{\lambda_i, \lambda_j\}}(w)$ ▷ $O(N|\mathcal{E}| + N^2 \log N)$
  end for
end for
return Π

6.B.3 Analysis of Algorithm 6 (computing a generic locus)

We provide Algorithm 6 with comments corresponding to the time complexity of each step.

Algorithm 6 Computing a generic locus

Require: Λ, $\mathcal{Y}$, $G = (\mathcal{V}, \mathcal{E})$
$\Pi \leftarrow \emptyset$
$D \leftarrow \text{diam}(G)$ ▷ $O(N|\mathcal{E}| + N^2 \log N)$
for $\mathbf{w}$ in $\left\{\frac{0}{D}, \frac{1}{D}, ..., \frac{D}{D}\right\}^{|\Lambda|}$ **do**
  ▷ $O(D^K)$ *iterations*
  $\Pi \leftarrow \Pi \cup m_{\boldsymbol{\Lambda}}(w)$ ▷ $O(N|\mathcal{E}| + N^2 \log N)$
end for
return Π

Following a similar argument from the analysis of Algorithm 5, the time complexity is

$$O(N|\mathcal{E}| + N^2 \log N + D^K(N|\mathcal{E}| + N^2 \log N)) = O(D^K),$$

where K is the number of observed classes.

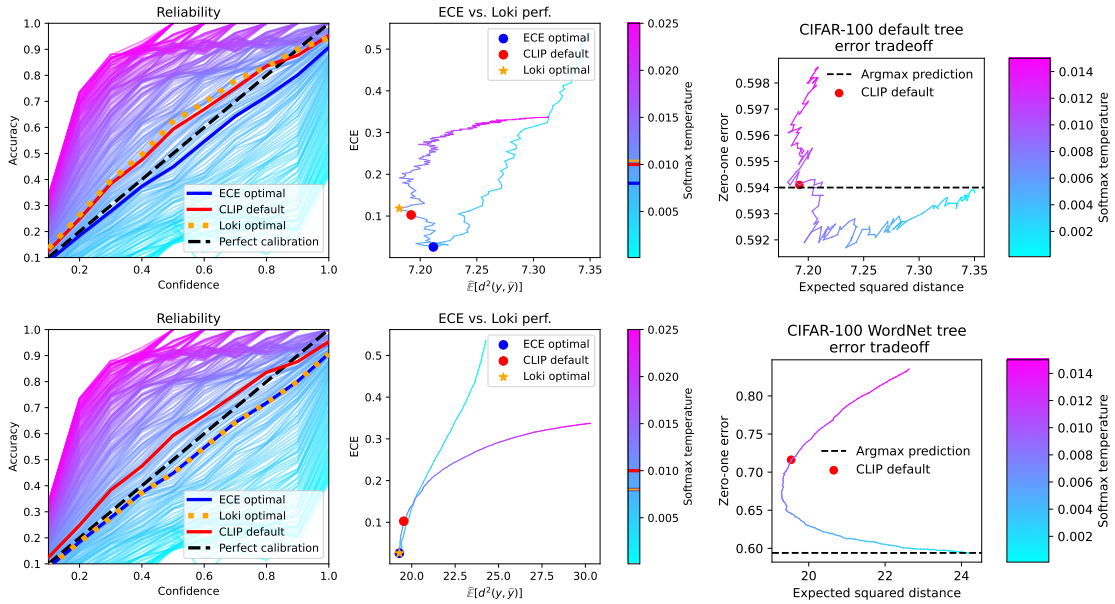


Figure 6.4: CLIP on CIFAR-100 with the WordNet hierarchy. (Left) Reliability diagrams across a range of Softmax temperatures, highlighting the CLIP default temperature, the optimal temperature for Loki, and the minimizer of the Expected Calibration Error (ECE). All three are well-calibrated. (Center) Tradeoff between optimizing for ECE and the expected squared distance. As with the reliability diagrams, the CLIP default temperature, the Loki-optimal temperature, and the ECE-optimal temperature are similar. (Right) Tradeoff between optimizing for zero-one error and the expected squared distance. Depending on the metric space, temperature scaling can improve *accuracy*.

6.C Additional Experiments

6.C.1 Additional calibration experiments

In the main text, we evaluated the effect of calibrating the Softmax outputs on the performance of Loki using a metric space based on the internal features of the CLIP model. Here, we perform the same analysis using two external metric spaces.

Setup. We perform our calibration analysis using the default and WordNet tree.

Results. Our results are shown in Figure 6.4. Like our experiment using an internally-derived metric, we find that the optimal Softmax temperature is close to the CLIP default

and the optimal temperature for calibration. For the default tree, we again found that temperature scaling can be used to improve accuracy using Loki.

Setup. We calibrate via Softmax temperature scaling [Guo et al., 2017] using CLIP on CIFAR-100. We do not use an external metric space, and instead use Euclidean distance applied to the CLIP text encoder.

Results. The reliability diagram in Figure 6.3 shows that the optimal Softmax temperature for Loki is both close to the default temperature used by CLIP and to the optimally-calibrated temperature. In Figure 6.3 (right), we find that appropriate tuning of the temperature parameter *can lead to improved accuracy with CLIP*, even when no external metric space is available.

6.C.2 Ablation of Loki formulation

Loki is based on the Fréchet mean, which is defined as $\arg\min_{y \in \mathcal{Y}} \sum_{i=1}^{K} \mathbf{P}_{\lambda_i|x} d^2(y, \lambda_i)$. However, this is not the only approach that can be considered. For example, the Fréchet *median*, often used in robust statistics, is defined as $\arg\min_{y \in \mathcal{Y}} \sum_{i=1}^{K} \mathbf{P}_{\lambda_i|x} d(y, \lambda_i)$, without squaring the distances. More generally, we define $\arg\min_{y \in \mathcal{Y}} \sum_{i=1}^{K} \mathbf{P}_{\lambda_i|x} d^\beta(y, \lambda_i)$ and evaluate different choices of β.

Setup. We conduct this experiment on ImageNet using SimCLR as our pretrained classifier with $K = 250$, 500, and 750 randomly selected classes.

Results. From this analysis, we conclude that using the Fréchet mean is the optimal formulation for Loki, as it achieved the lowest mean squared distance for all settings of K.

6.C.3 Comparison across metric spaces

Expected squared distances cannot be directly compared across metric spaces, as they may be on different scales. Our solution is to use a form of normalization: we divide the expected squared distance by the square of the graph diameter. This brings all of the values to the 0-1 range, and since $\mathbb{E}[d^2(y, \hat{y})]/\text{diam}(G)^2$ indeed also generalizes the 0-1 error, this enables comparison between 0-1 errors and those from different metric spaces. We provide these

Table 6.4: Expected squared distances of SimCLR+Loki alternatives on ImageNet. We ablate over the choice of distance exponent in Loki (where $\beta = 2$, corresponding to the Fréchet mean), including the Fréchet median ($\beta = 1$). That is, we tune $\beta : \hat{y} \in \arg\min_{y \in \mathcal{Y}} \sum_{i=1}^{K} \mathbf{P}_{\lambda_i|x} d^{\beta}(y, \lambda_i)$ and find that the optimal setting is $\beta = 2$, corresponding to Loki.

$\beta =$	$K = 250$	$K = 500$	$K = 750$
0.5	61.28	46.57	36.31
1	52.98	41.06	33.14
2 (Loki)	**46.78**	**37.76**	**29.88**
4	47.99	39.58	34.65
8	65.29	58.42	54.90

results in Table 1, again for our CLIP experiments on CIFAR-100. This evaluation metric enables us to determine which metric spaces have geometry best 'fit' to our pretrained models.

Setup. Using $\mathbb{E}[d^2(y, \hat{y})]/\text{diam}(G)^2$, we re-evaluate our CLIP results on CIFAR-100 shown in Table 6.1 for the ResNet-50 architecture.

Results. For CIFAR-100, we observe that the WordNet metric space resulted in the lowest error and therefore has the best geometry.

Table 6.5: Comparison across metric spaces for CLIP on CIFAR-100 by normalizing by the squared diameter of the metric space: $\mathbb{E}[d^2(y, \hat{y})]/\text{diam}(G)^2$.

Metric space G	$\text{diam}(G)^2$	$\mathbb{E}[d^2(y, \hat{y})]/\text{diam}(G)^2$
Complete graph	1	0.5941 (0-1 error)
Default tree	16	0.4493
WordNet tree	169	**0.1157**
CLIP features	0.9726	0.2686

6.D Experimental Details

In this section, we provide additional details about our experimental setup.[3]

6.D.1 CLIP experiments

All experiments are carried out using CLIP frozen weights. There are no training and hyper-parameters involved in experiments involving CLIP, except for the Softmax temperature in the calibration analysis. We evaluted using the default CIFAR-100 test set. The label prompt we use is "a photo of a [class_name]."

6.D.2 ImageNet experiments

To construct our datasets, we randomly sample 50 images for each class from ImageNet as our training dataset then use the validation dataset in ImageNet to evaluate Loki's performance. We extract the image embeddings by using SimCLRv1[4] and train a baseline one-vs-rest classifier. We use WordNet phylogenetic tree as the metric space. The structure of phylogenetic tree can be found at here[5]. We test different numbers of observed classes, K, from 1000 classes. Observed classes are sampled in two ways, uniformly and Gibbs distribution with the concentration parameter 0.5.

6.D.3 LSHTC experiments

We generate a summary graph of the LSHTC class graph (resulting in supernodes representing many classes) by iteratively:

1. randomly selecting a node or supernode
2. merging its neighbors into the node to create a supernode

until the graph contains at most 10,000 supernodes. In the LSHTC dataset, each datapoint is assigned to multiple classes. We push each class to its supernode, then apply majority

[3] Code implementing all of our experiments is available here: `https://github.com/SprocketLab/loki`.

[4] Embeddings are extracted from the checkpoints stored at: `https://github.com/tonylins/simclr-converter`

[5] `https://github.com/cvjena/semantic-embeddings/tree/master/ILSVRC`

vote to determine the final class. We test different numbers of observed classes, K, from 10,000 classes. We collect datapoints which are in the observed classes. Then split half of dataset as training dataset and make the rest as testing dataset, including those datapoints which are not in the observed classes. We train a baseline classifier using a 5-NN model and compare its performance with Loki.

6.E Broader Impacts and Limitations

As a simple adaptor of existing classifiers, it is possible for our method to inherit biases and failure modes that might be present in existing pretrained models. On the other hand, if the possibility of harmful mis-classifications are known a priori, information to mitigate these harms can be baked into the metric space used by Loki. Finally, while we have found that Loki often works well in practice, it is possible for the per-class probabilities that are output by a pretrained model to be sufficiently mis-specified relative to the metric over the label space, or for the metric itself to be mis-specified. Nonetheless, we have found that off-the-shelf or self-derived metric spaces to work well in practice.

6.F Random Locus Visualizations

In Figures 6.1, 6.5, 6.6, 6.7, 6.8, and 6.9, we provide visualizations of classification regions of the probability simplex when using Loki with only three observed classes out of 100 total classes, and different types of random metric spaces. The first example in Figure 6.1 shows the prediction regions on the probability simplex when using standard arg max prediction—the three regions correspond to predicting one of the three classes (0, 39, and 99) and no regions corresponding to any of the other classes $\{1, ..., 38, 40, ..., 98\}$. We compute these regions using a version of Algorithm 6, and while it does have exponential time complexity, the exponent is only three in this case since we consider only three observed classes.

On the other hand, the other three examples in Figure 6.1 and Figures 6.5, 6.6, 6.7, 6.8, and 6.9 all show the prediction regions on the probability simplex when using Loki. Figure 6.5 shows this for random tree graphs. Here, the prediction regions are striped or contain a single triangle-shaped region in the center—these correspond, respectively, to intermediate classes along branches of the tree leading up from the observed class and the prediction

region formed by the common-most parent node. Figure 6.6 shows similar regions, although these are more complex and are thus more difficult to interpret, as phylogenetic trees are metric subspaces equipped with the induced metric from trees. Furthermore, in order to generate phylogenetic trees with 100 leaves, we needed to create much larger trees than the ones used for Figure 6.5, which led to narrower prediction regions due to the higher graph diameter. Finally, Figures 6.7, 6.8, and 6.9 each show the prediction regions using Loki when the metric space is a random graph produced using Watts-Strogatz, Erdℋos-Rényi, and Barabási–Albert, models respectively. These prediction regions are more complex and represent complex relationships between the classes.

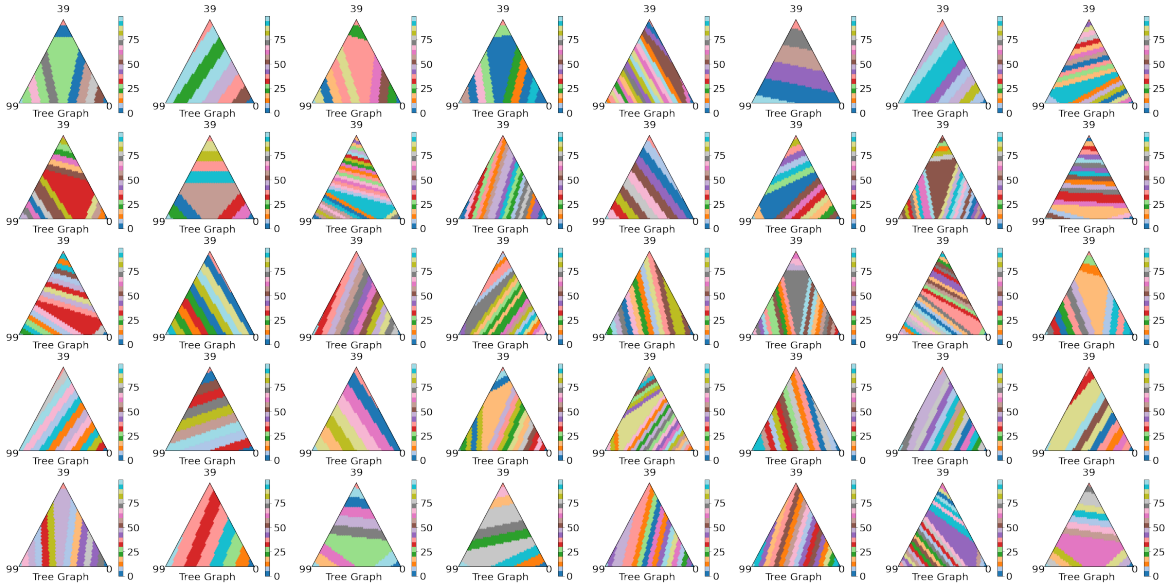


Figure 6.5: Classification regions in the probability simplex of 3-class classifiers faced with a 100-class problem where the classes are related by random trees graphs.

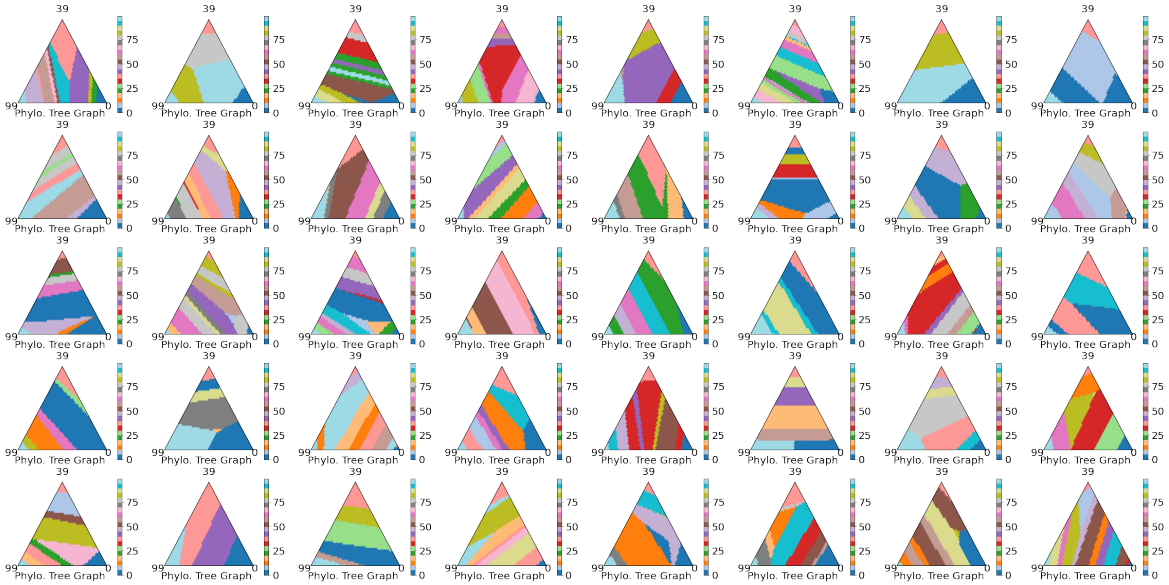


Figure 6.6: Classification regions in the probability simplex of 3-class classifiers faced with a 100-class problem where the classes are related by random phylogenetic trees graphs.

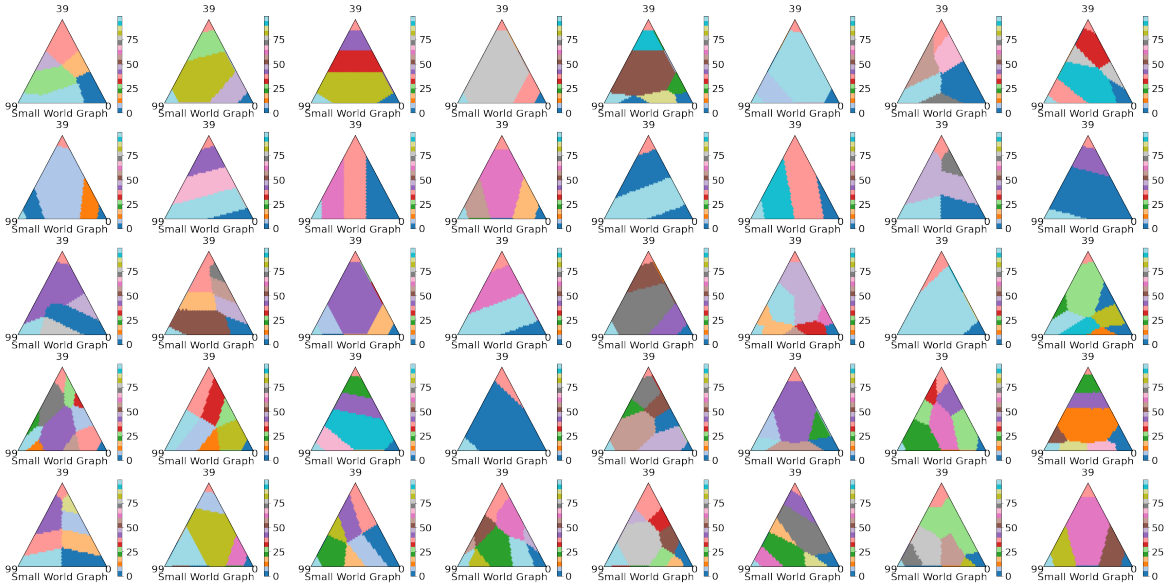


Figure 6.7: Classification regions in the probability simplex of 3-class classifiers faced with a 100-class problem where the classes are related by random small-world graphs generated by a Watts–Strogatz model.

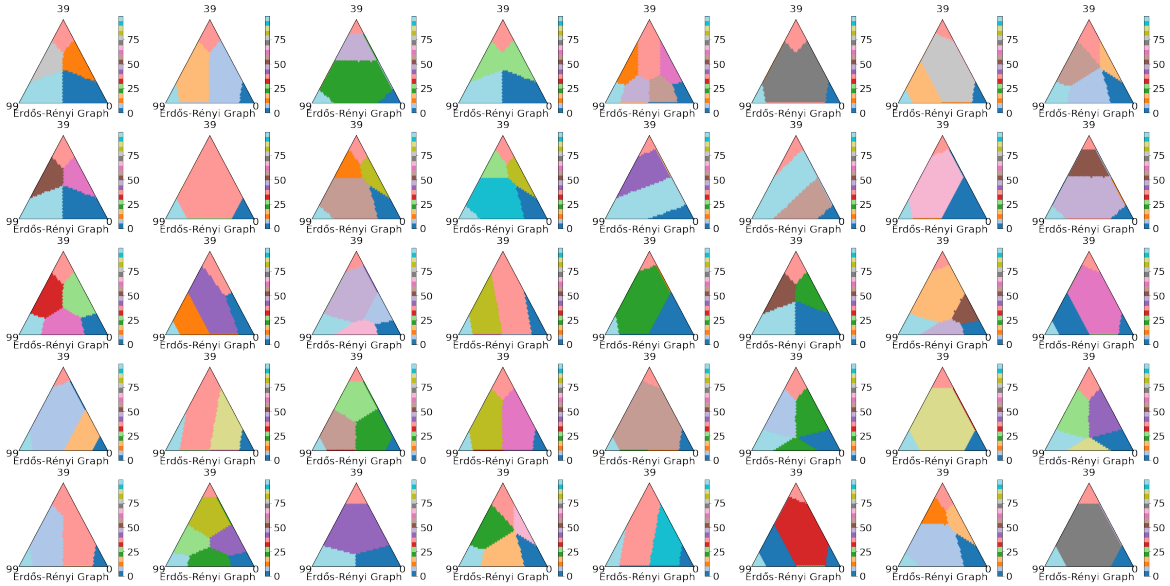


Figure 6.8: Classification regions in the probability simplex of 3-class classifiers faced with a 100-class problem where the classes are related by random Erdℋos-Rényi graphs.

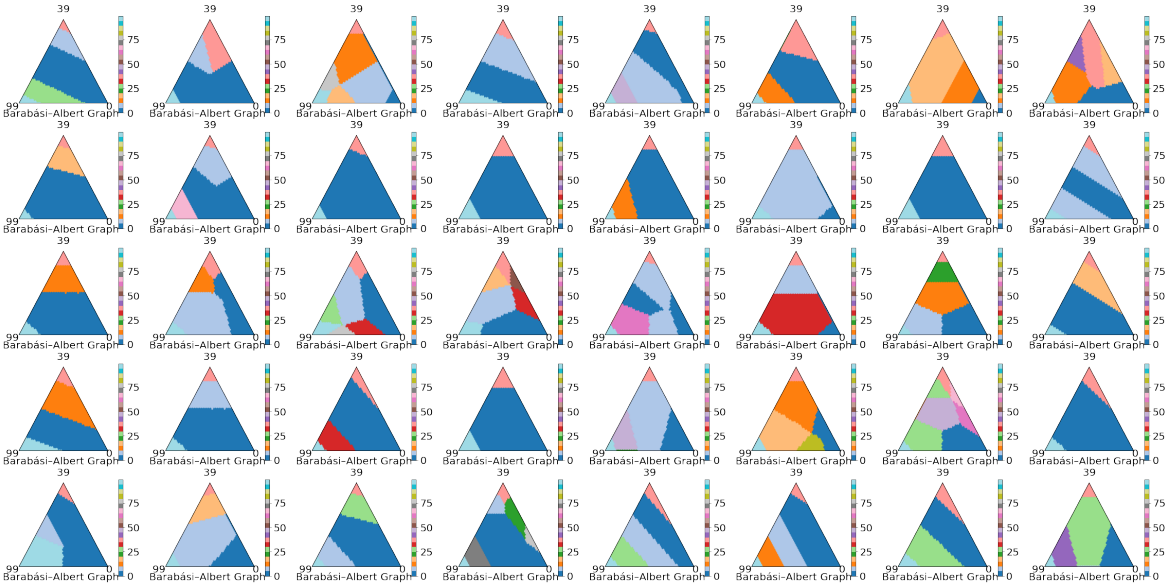


Figure 6.9: Classification regions in the probability simplex of 3-class classifiers faced with a 100-class problem where the classes are related by random Barabási–Albert graphs.

Chapter 7

AutoWS-Bench-101: Benchmarking Automated Weak Supervision with 100 Labels

This chapter is based on joint work with Xintong Li, Tzu-Heng Huang, Dyah Adila, Spencer Schoenberg, Cheng-Yu Liu, Lauren Pick, Haotian Ma, Aws Albarghouthi, and Frederic Sala, published at NeurIPS 2022 (Datasets and Benchmarks). The first two authors contributed equally.

The success of modern deep learning has been driven by large labeled datasets. Unfortunately, obtaining labeled data at scale is expensive and slow. Weak supervision (WS) [Ratner et al., 2016, 2017, 2019, Fu et al., 2020] is one of the most successful solutions to the problem of generating labeled datasets. The idea behind WS is simple: replace hand-labeling with acquiring and aggregating noisy-but-cheap label estimates from external sources—also called labeling functions (LFs)—to produce denoised estimates of true labels. WS is deployed widely and has had a significant impact. For example, in the medical domain, it has been used to re-create radiography datasets in a few hours that saved physician-months in manual labeling costs [Dunnmon et al., 2020]. It has led to the creation of the largest dataset of aortic valve malformations developed atop the large-scale UK Biobank [Fries et al., 2019, Sudlow et al., 2015]. In addition, WS has enjoyed wide industrial adoption. It is used in production by core teams within Google, Apple, and many more [Bach et al., 2019, Ré et al., 2020].

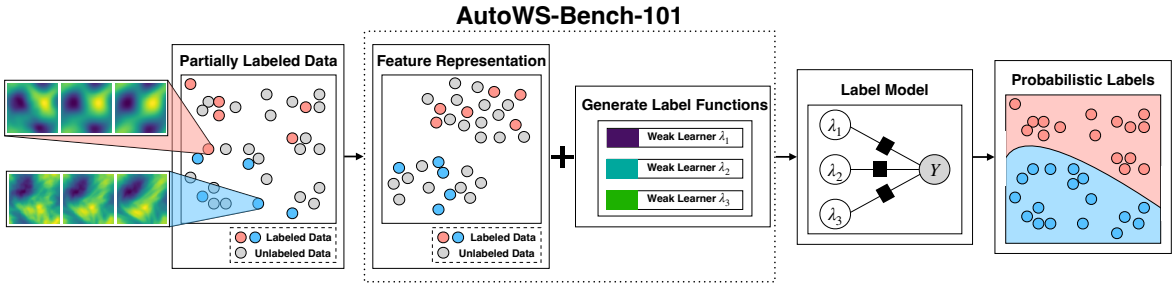


Figure 7.1: Overview of the AutoWS-Bench-101 pipeline.

Despite its massive success and widespread adoption, weak supervision can be challenging to apply. While much cheaper than manually labeling millions of examples, LF generation is sometimes costly. Obtaining high-quality LFs often requires training domain experts and painstaking iteration. Furthermore, creative choices must be made when applying WS to new modalities. For example, LFs that apply to text may not lift to image segmentation [Hooper et al., 2021] or recommendation [Shin et al., 2022] tasks.

Automated WS (AutoWS) methods are a best-of-both-worlds solution. These techniques generate LFs automatically by training models on a small initial set of ground truth labels [Varma and Ré, 2018, Boecking et al., 2021, Das et al., 2020]. At first glance, AutoWS techniques resemble few-shot learners. However, they are much more, forming *omnivorous frameworks* capable of incorporating and combining signals from many sources, including powerful few- and zero-shot models. Indeed, recent evidence [Chen et al., 2022, Smith et al., 2022] suggests that WS can exploit signal from powerful foundation models like CLIP, GPT-3, and T0++ [Radford et al., 2021, Brown et al., 2020a, Sanh et al., 2022].

AutoWS techniques have tremendous potential. For example, they

- can fuse multiple sources of signal, including foundation models, to maximize performance,
- inherit from WS the ability to denoise labels and to generalize beyond their constituent LFs,
- enable applying WS to domains and modalities where hand-crafting LFs is difficult or impossible.

Missing, however, is a thorough comparison of AutoWS techniques or an in-depth understanding of which components lead to good performance. For example, intuitively these methods rely on high-quality feature representations, but it is not clear how performance scales with the choice of representation. It is crucial to understand how these methods

compare to—and can be combined with—existing few- or zero-shot models. Motivated to understand these mechanics under a diverse data settings previously underexplored by WS, we propose AutoWS-Bench-101.

AutoWS-Bench-101 is a framework and benchmark for experimenting with, understanding, and evaluating AutoWS methods. It enables plug-and-play combinations of feature representations, LF candidate generation approaches, and LF selection procedures, conveniently built on top of the WRENCH [Zhang et al., 2021a] framework. It can be used with existing AutoWS methods like Snuba [Varma and Ré, 2018], interactive WS [Boecking et al., 2021], and GOGGLES [Das et al., 2020], or extensions of such techniques. A pipeline is shown in Figure 7.1. AutoWS-Bench-101 evaluation uses a diverse set of datasets on which WS has previously seen little application. Our benchmark reveals the following key insights about AutoWS:

1. Foundation models are only helpful to AutoWS for in-distribution or related tasks,
2. LFs that output multiple classes might be better than class-specialized LFs,
3. Foundation model usage can hurt coverage, the fraction of points labeled by AutoWS.

Once AutoWS methods are better understood, these methods have the promise of drastically increasing the application scope of WS. Our benchmark takes a crucial first step toward this understanding. We publicly release all experiment code and datasets for AutoWS-Bench-101 on GitHub.[1]

7.1 Background

We provide background on weak supervision (WS), automated WS (AutoWS), along with pretrained and foundation models. Finally, we formalize the background on AutoWS by providing a mathematical formulation of the problem setting.

Weak Supervision. WS methods generate labels for unlabeled data by combining multiple sources of imperfect label estimates [Ratner et al., 2016, 2017, Fu et al., 2020, Shin et al., 2022]. These easily obtained and applied sources are called labeling functions (LFs). They can be created from heuristic rules produced by an expert, knowledge-base queries, pre-trained model outputs, or by any other means cheaper than obtaining ground-truth labels [Gupta

[1] `https://github.com/Sala-Group/AutoWS-Bench-101`

and Manning, 2014, Karger et al., 2011, Dawid and Skene, 1979, Mintz et al., 2009, Hearst, 1992]. LFs provide a noisy estimate of the label, or otherwise abstain. If all the LFs abstain on a point, WS methods cannot label it; the fraction of points labeled by WS is the *coverage*.

Much of the WS literature focuses on the label model, the tool used to aggregate LF outputs to produce de-noised label estimates. New variants with different properties were introduced in [Ratner et al., 2016, 2019, Fu et al., 2020, Shin et al., 2022]. The recent benchmark WRENCH [Zhang et al., 2021a] focuses on evaluating such models. In contrast, we are focused on understanding the mechanics of AutoWS independent of the choice of label model.

Automated Weak Supervision. In many cases, the process of designing LFs is expensive or challenging. Several automated weak supervision (AutoWS) methods have been developed to automatically create LFs from some small initial set of labeled examples. Snuba [Varma and Ré, 2018] generates a large set of weak learners trained on combinations of features and iteratively selects LFs based on various performance criteria. Interactive Weak Supervision (IWS) [Boecking et al., 2021] also selects LFs from a pool of candidates by interactively learning from human feedback. While not explicitly an automated procedure, IWS is easily modified to not require a human in the loop. Finally, GOGGLES [Das et al., 2020] was introduced to address the challenge of automating the LF creation process, as well as the challenge of using WS for image data. GOGGLES computes pairwise affinity scores between feature representations of examples obtained by, for example, a pretrained VGG16 network, and learns a hierarchical generative model on the resulting affinity matrices. Additionally, several AutoWS methods have been developed specifically for the NLP domain. Some methods such as Darwin [Galhotra et al., 2021], PRBoost [Zhang et al., 2022a], and Nemo [Hsieh et al., 2022], like IWS, query annotators for feedback to assess the quality of synthesized rules. Other frameworks can be more complex and even more tailored to the NLP domain, such as ARI, which integrates inferred rules into pretrained NLP models [Pryzant et al., 2022].

Pre-trained and Foundation Models. Powerful pre-trained models are increasingly useful in modern machine learning. A standard approach to sidestep acquiring huge labeled datasets is to fine-tune a pre-trained model. For image data, this might be a ResNet [He et al., 2016b] trained on ImageNet. Self-supervised models [Chen et al., 2020, Grill et al., 2020] can

be used to acquire high-quality representations enabling users to train a simpler downstream model using fewer labeled points. Foundation models [Bommasani et al., 2021a], large-scale models trained on vast amounts of data, can be used as few- or zero-shot predictors. These approaches can be thought of as either competitors of AutoWS—if used as alternatives—or as AutoWS *ingredients*. For example, WS can be performed on top of foundation model embeddings, or use them to extend LF coverage [Chen et al., 2022], or use prompts as sources [Smith et al., 2022]. Understanding how pre-trained or foundation models compete with or contribute to AutoWS is a key element of this work.

Problem formulation. We provide a mathematical formulation of the AutoWS problem setting here. First, we begin with an unlabeled training set of n examples $X_{\text{train}} = [x_i]_{i=1}^{n}$ with $x_i \in \mathcal{X}$ and a labeled validation set of m examples $X_{\text{val}} = [x_j]_{j=n+1}^{n+m}$ with $x_j \in \mathcal{X}$, $Y_{\text{val}} = [y_j]_{j=n+1}^{n+m}$ with $y_j \in \mathcal{Y}$. Next, we embed examples in $\mathcal{X}$ into $\mathbb{R}^d$ using a feature extractor, which we denote as $\phi : \mathcal{X} \to \mathbb{R}^d$. Additionally, we denote the hypothesis class of synthetic LFs as $\mathcal{H}$ where $\forall h \in \mathcal{H}$, we have $h : \mathbb{R}^d \to \mathcal{Y} \cup \{-1\}$. We will denote an AutoWS method as a pair of algorithms, the first of which generates LFs and is parameterized by a feature extractor: $\mathcal{A}_{\text{WS}}^{\phi} = \left(\text{LF}_{\text{WS}}^{\phi}, \text{LM}_{\text{WS}}\right)$ where $\text{LF}_{\text{WS}}^{\phi} : \mathcal{X}^{n+m} \times \mathcal{Y}^{m} \to \mathcal{P}(\mathcal{H})$ and the second is the label model, which infers a weak label from one or more LF predictions: $\text{LM}_{\text{WS}} : \mathcal{Y}^{\geqslant 1} \to \mathcal{Y}$. We generate LFs using $\Lambda = \text{LF}_{\text{WS}}^{\phi}(X_{\text{train}}, X_{\text{val}}, Y_{\text{val}})$ and obtain LF predictions $\forall h_k \in \Lambda$ as $\lambda_i^k = h_k(\phi(x_i))$. We can furthermore obtain additional label predictions for each example from other sources, such as zero-shot predictions from foundation models or by using hand-designed LFs. We will denote the set of these additional predictions as $\widetilde{\lambda}_i = \left\{\widetilde{\lambda}_i^l\right\}_{l \geqslant 0}$, $\forall i \in [n+m]$. Finally, we obtain weak labels for the training set using the label model: $\widehat{y_i} = \text{LM}_{\text{WS}}(\{\lambda_i^k\}_{k=1}^{|\Lambda|} \cup \widetilde{\lambda}_i)$, $\forall i \in [n]$. In this work, we evaluate choices of ϕ and $\mathcal{A}_{\text{WS}}^{\phi}$.

7.2 Benchmark Goals and Methodology

We describe the goals of our proposed benchmark and our experimental methodology.

Goals. The goal of this work is to understand the effects of the key components of AutoWS—LFs, selection methods, and representations—on a diverse collection of datasets. We are especially interested in settings far from the ambit of conventional weak supervision—

most often used in text applications. We also seek to understand how AutoWS compares to leading zero-/few-shot methods when used as either an alternative or an ingredient.

Methodology. For ease-of-use, our benchmark is implemented as an extension of the recently introduced Weak supeRvision bENCHmark (WRENCH) [Zhang et al., 2021a]. We note that WRENCH and AutoWS-Bench-101 have fundamentally different goals: WRENCH evaluates label models, not AutoWS techniques. An additional difference is that WRENCH uses fixed pre-computed LFs and only includes a handful of image-based datasets without access to the original source images or videos.

We consider three well-known methods for generating LFs: Snuba, IWS, and GOGGLES [Varma and Ré, 2018, Boecking et al., 2021, Das et al., 2020], operating on a handful of baseline features: raw pixels, PCA with 100 principal components, the logits layer of a pre-trained ResNet-18 [He et al., 2016b], and zero-shot logits from CLIP [Radford et al., 2021]. The AutoWS-Bench-101 pipeline is shown in Figure 7.1. Below, we explain our choices in more detail.

7.2.1 Datasets

In order to evaluate AutoWS techniques on a range of challenging WS settings, we include a diverse set of 10 classification datasets. Image classification datasets are a salient example of the types of datasets that fit our goal, as these have proven to be a challenging domain for weak supervision. We include two such datasets, MNIST and CIFAR-10 [Deng, 2012, Krizhevsky, 2009], in AutoWS-Bench-101. We wish to avoid AutoWS methods overfitting to the image domain through the exploitation of feature representations specifically tailored to these tasks. For this reason we include two image-derived datasets that move beyond the scope of natural image classification. Spherical MNIST [Cohen et al., 2018b] consists of MNSIT digits stereographically projected onto the sphere, rotated, and mapped back to the plane. Permuted MNIST consists of MNIST digits whose pixels are permuted by a fixed bit-reversal permutation applied to the rows and columns. We additionally include three text datasets that are also used in WRENCH [Zhang et al., 2021a]: YouTube [Alberto et al., 2015], Yelp [Zhang et al., 2015, Ren et al., 2020], and IMDb [Maas et al., 2011b, Ren et al., 2020]. Finally, we include three tasks in more diverse application settings beyond image, image derived, or text datasets. These are the electrocardiogram (ECG) dataset of [Clifford

et al., 2017b], a turbulence classification variant of the Navier-Stokes PDE data used in [Li et al., 2021d], and EMBER, a tabular malware detection dataset [Anderson and Roth, 2018b]. Further details of the latter three datasets are provided below.

ECG. The ECG, or electrocardiogram dataset is commonly used to detect heart disease. It contains 60 second or shorter ECG recordings posed as a four-class classification problem: normal, disease, other, or noisy. This dataset was derived from the 2017 PhysioNet Challenge [Clifford et al., 2017b]. We use a fixed size of sliding window to process signals into a 1,000 ms time-series dataset.

Navier-Stokes. The Navier-Stokes equation is a partial differential equation that describes viscous fluid flows. A property of interest to the computational fluid dynamics community is the turbulence of these flows. In the case of the Navier-Stokes equation, this is governed by a viscosity parameter η. Navier-Stokes solutions are generated by [Li et al., 2021d]. We adapt the Navier-Stokes dataset at $\eta = 1e{-}3$ (non-turbulent) and $\eta = 1e{-}5$ (turbulent) to be a binary turbulence classification task.

EMBER. The EMBER dataset, or Elastic Malware Benchmark for Empowering Researchers, introduced by [Anderson and Roth, 2018b], is a tabular malware detection dataset extracted from malicious and benign Windows portable executable files. EMBER dataset is a binary classification task, and it includes several groups of feature, such as parsed features, format-agnostic features, and string information to detect malware. We use the open source implementation to extract features.[2]

7.2.2 Implementation Details

We adapt the open source implementations of Snuba, IWS, and GOGGLES to be compatible with WRENCH [Varma and Ré, 2018, Boecking et al., 2021, Das et al., 2020, Zhang et al., 2021a]. As AutoWS methods require a small set of labeled examples, we use 100 labeled examples in all experiments unless otherwise stated. We include a few-shot learning baseline—a logistic regression classifier trained on the same 100 labeled examples. Additionally, we include a semi-supervised baseline—label propagation [Wang and Zhang, 2008]. Finally,

[2] `https://github.com/elastic/ember`

we include zero-shot CLIP as another baseline. In the final evaluation of each method, we report output label model accuracy (see Table 7.1) and coverage (see Appendix).

Snuba and Multiclass Snuba. Unless otherwise stated, we use the Snuba implementation and hyperparameters [Varma and Ré, 2018]. This implementation only supports binary classification—however, five our datasets are multiclass. We adapt Snuba to multiclass settings in two ways. First, we run Snuba's LF synthesis procedure for each class as a one-vs-all problem. This results in unipolar LFs—a given LF will either vote for its assigned class or abstain. We also implemented a multipolar version of Snuba where LFs are themselves multiclass. By default, the search space of LFs is fixed to be decision stumps, logistic regression, or nearest neighbor classifiers. We found that allowing Snuba to consider both decision stumps and logistic regression classifiers for any given LF led to a slight improvement in performance. All of our experiments consider both of these. Each LF abstains according to a confidence threshold that is inferred by Snuba during the LF synthesis procedure. Finally, as in the original Snuba implementation, we aggregate the outputs of each LF using the label model.

Interactive Weak Supervision. We adapt the original implementation of IWS to our setting. IWS operates on an initial pool of binary LFs. For simplicity, we use Snuba's LF synthesis procedure to generate a large set of LFs that IWS can select from. Note that since we use Snuba's LF synthesis procedure, each LF has its own inferred abstention threshold. Specifically, we generate LFs with cardinality $D = 1$, the number of features that each LF can consider. Similarly, unless otherwise stated, we consider both decision stumps and logistic regression classifiers. Like Snuba, the original IWS implementation only supports binary classification. Since our benchmark is over 10-class problems, we run the IWS LF selection procedure once per class to produce unipolar LFs for each class before passing the outputs of these to the label model.

GOGGLES. GOGGLES computes pairwise affinity scores on a set of unlabeled examples which are used in a hierarchical generative model with to infer labels. To construct the affinity matrix, the default implementation uses the max-pooling layers of the pre-trained VGG-16 network to extract various levels of semantic representations. In our experiments, we replace these VGG-16 representations with our baseline feature representations. Pairwise

affinity scores are computed by cosine similarity between embeddings of each instance. After training a generative model to create clusters, GOGGLES uses a small amount of labeled datapoints to map clusters to classes. Note that GOGGLES does not abstain.

CLIP. We used standard CLIP [Radford et al., 2021] zero-shot prediction as a baseline, comparing image embeddings (except for the tabular dataset EMBER) with text embeddings of strings related to the class label. The complete list of these strings is provided in the Appendix.

BERT embeddings for text datasets. Our pipeline requires the features of each dataset to be compatible with all (or most) of the feature representation techniques that we evaluate. In order to make the text datasets that we evaluate work with our pipeline, we first extract feature vectors for our text datasets using BERT before applying other feature representations (except in the case of CLIP, which operates on text directly). For YouTube, Yelp, and IMDb, raw features refers to these BERT features, PCA embeddings refers to the PCA of BERT features, and ResNet-18 embeddings refers to BERT features passed through a pretrained ResNet-18 model. In the case of CLIP, we directly use the CLIP text encoder for both the input text and the candidate labels to produce a logits vector.

7.3 Analysis

AutoWS-Bench-101 evaluates combinations of AutoWS methods and feature representation methods on a diverse set of classification tasks. Specifically, we evaluate GOGGLES, Interactive Weak Supervision, Snuba, and a multi-polar variant of Snuba using raw features, PCA, the logits layer of a pretrained ResNet-18, and CLIP logits. We compare all of these to a logistic regression-based few-shot learning baseline, and a Label Propagation-based semi-supervised baseline [Wang and Zhang, 2008]. In order to robustly assess differences in the types of data domains under which various methods work best, we split our evaluations across image tasks (MNIST, CIFAR-10, Spherical MNIST, Permuted MNIST), diverse tasks (ECG, Navier-Stokes, EMBER), and text tasks (YouTube, Yelp, IMDb). We will go on to describe our analysis and results for determining which AutoWS and feature representation methods work well and do not work well on AutoWS-Bench-101.

Table 7.1: Test accuracies of AutoWS methods across features and datasets using 100 labels. N/A denotes an incompatibility between a method, feature representation, and/or task—see Appendix. Bolded and underlined scores indicate the first and second best results, respectively.

	MNIST	CIFAR-10	Spherical MNIST	Permuted MNIST	ECG	Navier-Stokes	Ember	YouTube	Yelp	IMDb
GOGGLES										
+Raw features	0.3778	0.1697	0.0897	0.3778	0.2710	0.5284	0.5283	0.5880	0.4732	0.5516
+PCA	0.4563	0.1989	0.1074	0.4561	0.2688	0.4863	0.6728	0.5880	0.5466	0.5628
+ResNet-18	0.3258	0.1768	0.1192	0.1894	0.3127	0.9353	0.5312	0.5200	0.4553	0.5152
+CLIP	0.3309	0.5436	0.1884	0.1405	N/A	0.7505	N/A	0.5280	0.4779	0.5000
IWS										
+Raw features	0.4128	0.1482	0.1154	0.3862	0.2615	0.5403	0.4997	0.4896	0.6034	0.5478
+PCA	0.3503	0.1047	0.1283	0.4970	0.2687	0.5155	0.4777	0.5217	0.5260	0.5045
+ResNet-18	0.0822	0.2120	0.1352	0.5101	0.4235	0.7995	0.7343	0.7840	0.5620	0.5286
+CLIP	0.6885	**0.7892**	0.1108	0.3495	N/A	N/A	N/A	N/A	N/A	N/A
Snuba										
+Raw features	0.2694	0.1624	0.0805	0.2308	0.2648	0.5235	0.7082	0.7520	0.6034	0.5936
+PCA	0.2313	0.0924	0.1526	0.2640	0.2598	0.8811	0.4905	0.6640	0.7187	0.5560
+ResNet-18	0.4219	0.1520	0.1554	0.1804	0.2506	0.9774	0.6994	0.7400	0.5674	0.5804
+CLIP	0.6617	0.7550	**0.3103**	0.2419	N/A	0.8414	N/A	0.6280	0.5608	0.5699
Snuba (multipolar)										
+Raw features	0.2374	0.1386	0.0991	0.2343	0.2596	0.5421	0.7264	0.7080	0.6484	0.6072
+PCA	0.5625	0.1390	0.2354	0.5625	0.2644	0.8968	0.5143	0.6280	0.7218	0.5020
+ResNet-18	0.3866	0.2036	0.1588	0.2283	0.3702	0.9742	**0.7559**	0.7320	0.5526	0.5864
+CLIP	0.3541	0.6788	0.2479	0.2785	N/A	0.7916	N/A	0.7759	0.5435	0.5681
Few-Shot (Logistic)										
+Raw features	0.7295	0.2315	0.1496	0.7295	0.2583	0.6316	0.6517	**0.8920**	**0.7871**	**0.6472**
+PCA	0.7306	0.2262	0.1483	**0.7305**	0.2548	0.7863	0.7488	0.8880	0.7868	0.6464
+ResNet-18	**0.7966**	0.3335	0.2544	0.3485	**0.4700**	**1.0000**	0.5329	0.7000	0.5682	0.5856
+CLIP	0.3579	0.6857	0.1011	0.1009	N/A	0.7658	N/A	0.7000	0.6347	0.5636
Semi-Supervised (Label Prop.)										
+Raw features	0.7066	0.1927	0.1374	0.7066	0.2644	0.5010	0.5003	0.8440	0.5447	0.5812
+PCA	0.7165	0.1914	0.1405	0.7156	0.2644	0.5011	0.5003	0.8440	0.5450	0.5812
+ResNet-18	0.7124	0.1277	0.1283	0.2106	0.2644	0.9700	0.5003	0.7000	0.5224	0.5500
+CLIP	0.1178	0.5780	0.1009	0.1009	N/A	0.5058	N/A	0.6480	0.5737	0.5400
CLIP Zero-Shot	0.5817	0.6989	0.2065	0.0899	N/A	0.6305	N/A	0.472	0.5223	0.5008

7.3.1 Comparison methodology

In addition to presenting a full set of results in Table 7.1, we adopt the performance profiles methodology of [Dolan and Moré, 2002c] to summarize many of our results in a holistic

manner. These are a robust way to visually compare methods in aggregate across noisy evaluations in a large number of environments (i.e., problem settings). The performance profile shows curves where points along each curve represent the fraction of settings where a given method is within a factor of the best method. This factor is denoted τ. Concretely, for each method s in a set of methods $\mathcal{S}$, we plot $\rho_s(\tau) = \frac{1}{|\mathcal{P}|}\left|\left\{p \in \mathcal{P} : \frac{\text{obj}_{p,s}}{\min_{s\in\mathcal{S}} \text{obj}_{p,s}} \leqslant \tau\right\}\right|$, where $\mathcal{P}$ is the set of problem settings/environments and $\text{obj}_{p,s}$ is the objective for s. The ideal curve has a high fraction, approaching the top-left corner.

The goal of AutoWS is to produce high accuracy and high coverage (that is, to label as many points as possible). When discussing accuracy, we set $\text{obj}_{p,s}$ to be standard classification error. For coverage, we set $\text{obj}_{p,s}$ to be $1-\text{coverage}$. We set $\mathcal{E}$ to be the set of feature representations, $\mathcal{W}$ to be the set of AutoWS methods, and $\mathcal{D}$ to be the set of datasets that we consider. We evaluate feature representations $\phi \in \mathcal{E}$ and AutoWS methods $\mathcal{A}_{\text{WS}}^{\phi} \in \mathcal{W}$ by defining their respective sets of environments to be datasets subject to either using a particular AutoWS method, or feature representation, respectively.

7.3.2 Comparison of feature representations and AutoWS methods

When do foundation models help AutoWS? First, we study the performance of different feature representation methods across datasets and AutoWS methods. Full results are presented in Table 7.1. We compare feature representation methods using performance profiles by defining an environment to be a dataset and AutoWS method pair. Specifically, we set $\mathcal{S} = \mathcal{E}$ and $\mathcal{P} = \mathcal{D} \times \mathcal{W}$. We then plot performance profile curves for each feature representation method as shown in Figure 7.2 (bottom row). We find that CLIP performs quite well on all of the image tasks that we evaluate, whereas ResNet-18 features and raw features were less performant compared to other methods. We postulate that this is because these tasks are somewhat similar to CLIP's training distribution. On diverse tasks, however, CLIP performs much worse—due to its incompatibilities with certain data modalities, such as time series (ECG) and embeddings of tabular data (EMBER), and when using IWS on Navier-Stokes, where the input dimensionality (2) of is too small to generate a large number of LFs. We hypothesize that this is because Navier-Stokes is farther from CLIP's training distribution. Surprisingly, ResNet-18 (which was pretrained on ImageNet) appears to perform quite well on diverse tasks; it achieves the best performance on ECG, and Navier-Stokes under using few-shot baseline and the best performance on EMBER when using Snuba with

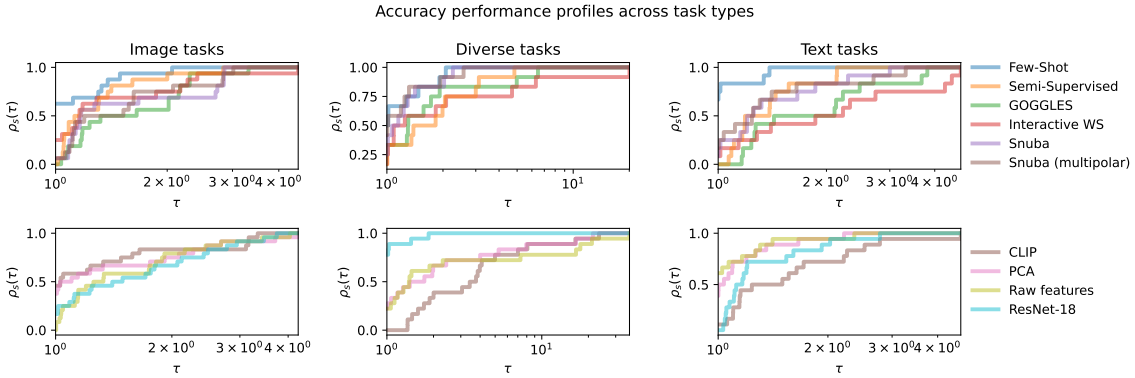


Figure 7.2: For image tasks (left)—MNIST, CIFAR-10, Spherical MNIST, Permuted MNIST—few-shot learning and CLIP both dominate. For diverse tasks (center)—ECG, Navier-Stokes, Ember—Snuba closes the gap with few-shot learning and CLIP underperforms or is non-applicable. For text tasks (right)—YouTube, Yelp, IMDb—few-shot learning and raw features (i.e., BERT embeddings), while the CLIP text encoder alone underperforms or faces compatibility issues with IWS.

multipolar LFs. We hypothesize that, while aimed at handling diverse downstream image tasks, ImageNet contains enough semantic information beyond images that it can handle, to at least a rudimentary degree, diverse tasks as well. This observation is in the same vein as prior work which shows that training on (seemingly unrelated) non-linguistic data improves downstream performance on natural language tasks [Papadimitriou and Jurafsky, 2020]. On text tasks, we find that CLIP underperforms, but we postulate that this is mainly due to the lack of image input in these settings. On the other hand, we find that raw features (i.e., text features extracted using BERT) outperforms other methods. Perhaps unsurprisingly, we find that our few-shot baseline (logistic regression) using raw BERT features outperforms all other methods on text tasks, as this corresponds to standard BERT fine-tuning, which furthermore suggests that this method should be *combined* with AutoWS methods. Our key finding is that CLIP helps AutoWS on in-distribution image tasks or image-derived tasks. CLIP does not help on diverse out-of-distribution tasks or on tasks where only text input is available, in which raw BERT features help.

How does AutoWS compare to few-shot learning? Next, we compare AutoWS to few-shot learning. Full results are in Table 7.1. For our performance profiles we set $\mathcal{S} = \mathcal{W}$ and $\mathcal{P} = \mathcal{D} \times \mathcal{E}$. The curves are shown in Figure 7.2 (top row). We find that few-shot

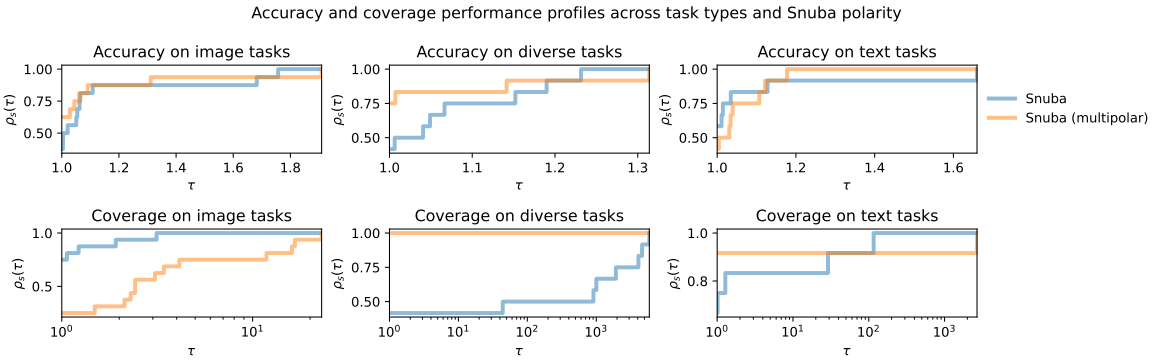


Figure 7.3: Using multipolar LFs typically improves the performance of Snuba (top). The difference is more pronounced on diverse tasks (top center). Multipolar LFs can result in lower coverage among image tasks (bottom left) which are notably all 10-class classification tasks. Conversely, multipolar LFs result in perfect coverage for the diverse tasks, which have fewer classes (bottom center).

learning typically ourperforms any other method on image tasks, attaining strong average performance. However, we note that in terms of absolute performance, IWS and Snuba perform the best on CIFAR-10 and Spherical MNIST (both when using CLIP), though they are less robust to changes in the underlying feature representation. As shown in Figure 7.2 (top center), we find that Snuba (both unipolar and multipolar) perform similarly to few-shot learning on diverse tasks. On Navier-Stokes, multipolar Snuba nearly matches the 100% accuracy of the few-shot baseline and on EMBER, multipolar Snuba outperforms few-shot learning using ResNet-18 features. In terms of absolute performance on all applicable datasets, at least two of the AutoWS methods also outperform raw zero-shot CLIP predictions. We again note that for all text tasks, few-shot learning using raw BERT features (i.e., fine-tuning BERT) outperforms all other methods. Our key finding is that few-shot learning is a strong baseline with good average performance across tasks, which suggests that it should be combined with AutoWS methods.

7.3.3 Evaluation of scientific questions using AutoWS-Bench-101

Are multiclass LFs better than class-specialized LFs? Next, we directly compare Snuba with unipolar LFs to Snuba with multipolar LFs. To our knowledge, a study of the value of unipolar vs. multipolar LFs has not been previously conducted in the WS literature. The top

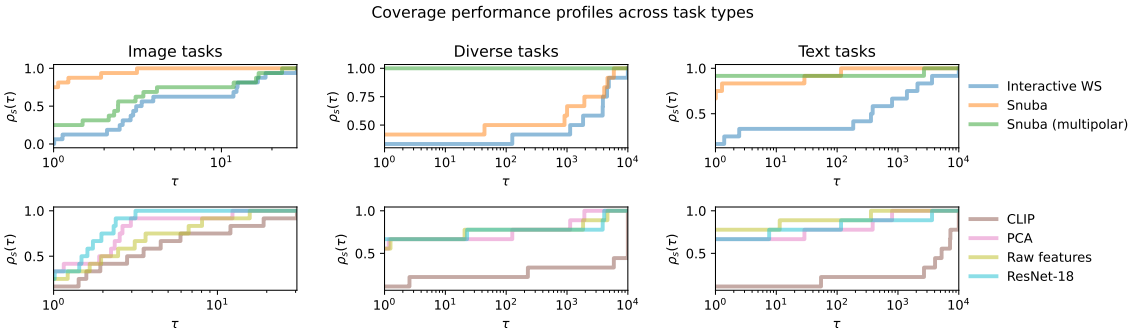


Figure 7.4: For image tasks (left)—MNIST, CIFAR-10, Spherical MNIST, Permuted MNIST—Snuba attains high coverage, and CLIP usage tends to worsen coverage. For non-image tasks (right)—ECG, Navier-Stokes, Ember—Multipolar Snuba always attains full coverage and CLIP is either non-applicable or attains poor coverage.

row of Figure 7.3 shows that multipolar Snuba can attain slightly better average performance than unipolar Snuba across all feature representations on image tasks, though in terms of absolute numbers in Table 7.1, unipolar Snuba significantly outperforms multipolar Snuba when using CLIP embeddings (typically the best feature representation for both methods). On diverse tasks, multipolar Snuba outperforms or nearly matches unipolar Snuba, across all feature representations. We furthermore find that on diverse tasks, multipolar Snuba attains perfect coverage in all cases, whereas unipolar Snuba often does not. We note that on these diverse tasks, ECG has 4 classes the other two, Navier-Stokes and EMBER, are binary. We find that on text tasks, multipolar Snuba tends to outperform unipolar Snuba subject to using the best feature representations for each. We also note that multipolar Snuba attains greater than or equal coverage to Snuba on the text tasks (which are all binary) in all but one instance—YouTube with CLIP embeddings. Surprisingly, we find that multipolar Snuba's high coverage does not translate to image tasks, where the number of classes is higher, and unipolar LFs tend to result in much higher coverage on average. We conclude that multipolar LFs tend to perform better on average but result in lower coverage when the number of classes increases.

Do the best methods still cover the label distribution? A critical aspect for a WS method to be practically useful is that it must be able to produce labels for all of the classes in the label distribution with the same frequency at which they occur in the underlying

data distribution. We thus perform an analysis of the coverage of methods in our pipeline subject to using different AutoWS methods and feature representations. We first consider the effect of using different AutoWS methods on coverage. We detail all of the coverage results across IWS, Snuba, and multipolar Snuba in the Appendix (note that GOGGLES by definition always produces 100% coverage). We present performance profile results of our coverage analysis in Figure 7.4. As before, we find that unipolar Snuba still attains the best coverage across 10-class image tasks, while multipolar Snuba attains the best coverage across diverse tasks and text tasks (with the only exception being YouTube with CLIP embeddings). In the bottom row of Figure 7.4, we show that methods using CLIP attain, on average, comparatively low coverage across all tasks. On diverse tasks, this can be attributed to the fact that CLIP is only applicable to Navier-Stokes, and not to ECG or EMBER (see Figure 7.4, bottom right). On text tasks, this is partly due to the inapplicability of 2D CLIP logits to IWS (see Appendix for a more detailed explanation of incompatibilities). However, CLIP is applicable to all image tasks, and still attains the worst coverage performance across feature representations (see Figure 7.4, bottom left). Conversely, ResNet-18 attains the best coverage on image tasks, and coverage is roughly similar between ResNet-18, PCA, and raw features on diverse and text tasks. From this analysis, we conclude that CLIP usage can hurt coverage.

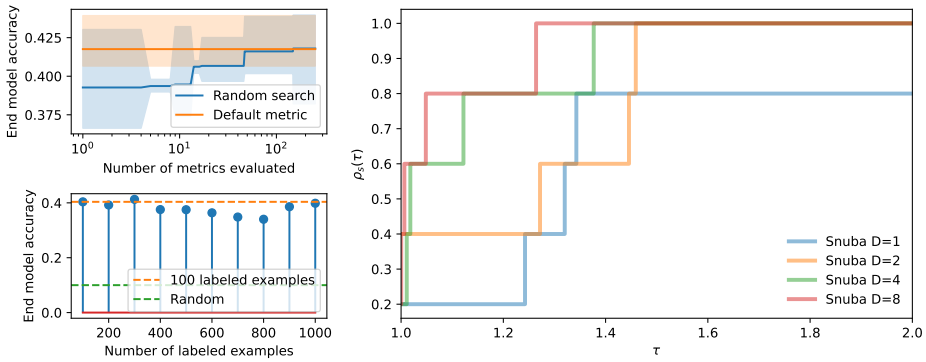


Figure 7.5: (Top left) Tuning Snuba's LF selection and abstention criteria do not lead to an improvement on MNIST. Results are reported as averages, minima, and maxima over 3 random seeds. We evaluate Snuba on MNIST using PCA and 500 feature combination samples with cardinality $D = 2$. (Bottom left) Increasing the number of labeled examples does not improve the performance of Snuba. (Right) Increasing cardinality can lead to some improvement with Snuba.

7.3.4 Ablation study of Snuba

We conclude our analyses with an ablation study of Snuba, the best performing AutoWS method that we evaluated. We consider the effects of varying the cardinality (the number of features that each LF can consider), the LF selection criterion used by Snuba (F1 score), and the effect of increasing the initial number of labels beyond the original 100.

Does increasing cardinality help Snuba? Snuba's cardinality parameter dictates how many components of the feature representation each LF can consider during its LF synthesis procedure. We evaluate the aggregate effect on performance of varying this parameter using performance profiles, as shown in Figure 7.5 (left). In this case, we set $\mathcal{S}$ to be Snuba with cardinalities $\mathsf{D} = 1, 2, 4,$ and 8. We consider only CLIP embeddings and all datasets on which CLIP is applicable: MNIST, CIFAR-10, Spherical MNIST, Permuted MNIST, and Navier-Stokes. This limits the maximum cardinality that we can consider to 10, which allows us to explore the full space of possible improvements for Snuba along the axis of cardinality. In these settings, increasing cardinality improves performance up to a point. Beyond $\mathsf{D} = 4$, we observe that Snuba ceases to improve much. From this, we conclude that increasing cardinality can lead to performance improvements up to a point.

Is F1 the right metric for MNIST when using Snuba? Given the poor performance of Snuba on MNIST, we evaluate Snuba's default choice to use F1 scores for LF selection and for determining abstain thresholds—micro F1 and weighted F1, respectively. We do so by first defining a search space over all applicable classification metrics available in scikit-learn [Pedregosa et al., 2011a]. These include Snuba's default choices, accuracy, balanced accuracy, precision, recall, Cohen's kappa coefficient, Jaccard index, Matthews correlation coefficient, and using the micro-averaged F1 score for both LF selection and to determine abstain thresholds. We perform a random search over convex combinations of these 9 metrics by sampling configurations from a Dirichlet distribution with parameter $\boldsymbol{\alpha} = \mathbf{1}_9$. As shown in Figure 7.5, we find no appreciable improvement over the default configuration. However, the best configuration found by random search has majority weight on micro-averaged F1 score, which only slightly differs from the default configuration in Snuba. These results indicate that Snuba's poor performance on MNIST is not trivially due to a poor choice of metric for LF selection.

Do more labels help Snuba? We ask if we can further improve performance on MNIST by increasing the number of labeled examples. In this experiment, we varied the number of labeled examples given to Snuba from 100 to 1000. We set the cardinality to $\mathrm{D} = 2$ and use raw pixel features. For efficiency, we randomly subsample 1000 feature combinations so as to avoid exploring the entire space of pairwise feature combinations. We answer the question in the negative: our results in Figure 7.5 show that increasing the number of labeled examples does not improve the performance of Snuba. This suggests that any performance with Snuba in this setting is likely algorithmic, and not due to a lack of labeled examples.

7.4 Conclusion

In this work, we proposed AutoWS-Bench-101, a benchmarking framework for AutoWS methods and beyond on a diverse set of classification tasks. We found that foundation models can improve the performance of AutoWS methods on in-distribution tasks but they can also hurt coverage, sometimes dramatically, on out-of-distribution tasks. We furthermore found that AutoWS methods using multipolar LFs might outperform unipolar LFs on average, but can result in lower coverage as the number of classes increases. One limitation is that we only considered CLIP—but there are many other applicable methods to be studied in future work. We hope this work takes a first step towards understanding how to build best-of-all-worlds automated weak supervision.

The appendix is organized as follows. First, we will detail the licenses for the datasets that are part of our benchmark. Second, we will provide additional insights that AutoWS-Bench-101 generated. These include a study of coverage for our techniques. We then perform additional ablations for GOGGLES and IWS. We also compare AutoWS with hand-crafted LFs, confirming that there are significant improvements available in cases where manual LFs are noisy. Next, we provide experimental details including the prompts that we used for our zero-shot CLIP experiments. Finally, we provide additional explanations of the incompatibilities between various methods in our pipeline. We conclude with societal and privacy implications for this work.

7.A Dataset licenses

The license information is provided below.

- MNIST: CC BY-SA 3.0
- CIFAR-10: CC BY 4.0 (on `https://www.tensorflow.org/datasets/catalog/cifar10`)
- Spherical MNIST: MIT
- Permuted MNIST: Apache 2.0
- Navier-Stokes: MIT
- ECG: ODC-BY 1.0
- EMBER: MIT
- YouTube: Apache 2.0
- Yelp: `https://s3-media0.fl.yelpcdn.com/assets/srv0/engineering_pages/dc1cabe7cb95/assets/vendor/Dataset_User_Agreement.pdf`
- IMDb: `https://www.imdb.com/conditions?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=3aefe545-f8d3-4562-976a-e5eb47d1bb18&pf_rd_r=FP5VRV15YGQSTMJVFZDM&pf_rd_s=center-1&pf_rd_t=60601&pf_rd_i=interfaces&ref_=fea_mn_lk2`

7.B Coverage analysis

Table 7.2 provides the results for coverage (the percentage of points in the dataset that get labeled) for the techniques we studied. We note that Snuba often produces relatively high coverage, especially with PCA-based feature representations.

Table 7.2: Coverage of AutoWS methods using various feature representations across datasets.

	MNIST	CIFAR-10	Spherical MNIST	Permuted MNIST	ECG	Navier-Stokes	Ember	YouTube	Yelp	IMDb
IWS										
+Raw features	0.3617	0.4616	0.3666	0.3656	0.5125	0.8163	1.0000	0.9640	1.0000	0.7908
+PCA	0.5107	0.2731	0.2696	0.4797	0.5448	0.9874	0.8873	0.9200	0.9616	0.9816
+ResNet-18	0.7422	0.3580	0.0917	0.1631	0.6109	1.0000	0.6142	1.0000	0.6368	0.8596
+CLIP	0.2758	0.7239	0.0993	0.1757	n/a	n/a	n/a	n/a	n/a	n/a
Snuba										
+Raw features	0.6842	0.6409	0.8387	0.8105	0.9090	0.5379	1.0000	1.0000	1.0000	1.0000
+PCA	0.9608	0.7183	0.9557	0.9716	0.9956	1.0000	0.8072	1.0000	0.9971	1.0000
+ResNet-18	0.8761	0.9772	0.9230	0.9320	0.8998	1.0000	0.5912	1.0000	0.9884	1.0000
+CLIP	0.7667	0.7262	0.1521	0.1323	n/a	0.4116	n/a	1.0000	0.4781	0.2716
Snuba (multipolar)										
+Raw features	0.7422	0.8129	0.4470	0.5407	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
+PCA	0.3472	0.5807	0.4763	0.3472	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
+ResNet-18	0.6185	0.6337	0.8232	0.7193	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
+CLIP	0.5027	0.3372	0.2033	0.7235	n/a	1.0000	n/a	0.7320	0.5926	0.2816

7.C Ablation study of GOGGLES

Next, we perform an ablation study of GOGGLES. In our analysis, we will evaluate three questions: whether GOGGLES improves when combining embedding types, how GOGGLES performs per-class, and whether or not the choice of inference model should be tuned.

Does GOGGLES improve with more embedding types? Instead of using a single type of feature representation to compute affinity scores between instance pairs, we leverage different types of embeddings to create multiple affinity matrices. These are then stacked and fed to the hierarchical generative model in GOGGLES for label inference. Through different types of embeddings, GOGGLES may capture correlations from different views. We evaluate the performance of GOGGLES with more embedding types in Table 7.3. Note that the ECG dataset and EMBER dataset use three types of embeddings only (they do not use CLIP embeddings) to create affinity matrices. Compared to the best single-embedding performance for each dataset in Table 7.1, we find that GOGGLES does not improve with more embedding types. Specifically, only on MNIST and EMBER is performance close to the single best embedding. We hypothesize that this is due to potentially poor alignment between these embeddings, and that with a more sophisticated approach, improvements versus single best are possible.

Table 7.3: Multiple types of feature representations for GOGGLES.

	MNIST	CIFAR-10	Spherical MNIST	Permuted MNIST	ECG	Navier-Stokes	Ember
GOGGLES							
Multiple Representations	0.425	0.2313	0.1072	0.4085	0.2767	0.4868	0.6728
Best Single Rep. Performance	0.4563	0.5436	0.1884	0.4561	0.3127	0.9353	0.6728

How does GOGGLES perform per-class? Next, we measured the per-class precision recall curves using GOGGLES. GOGGLES has perfect coverage on all tasks. However, we find that in several scenarios, the per-class performance varies considerably (e.g., MNIST, CIFAR-10).

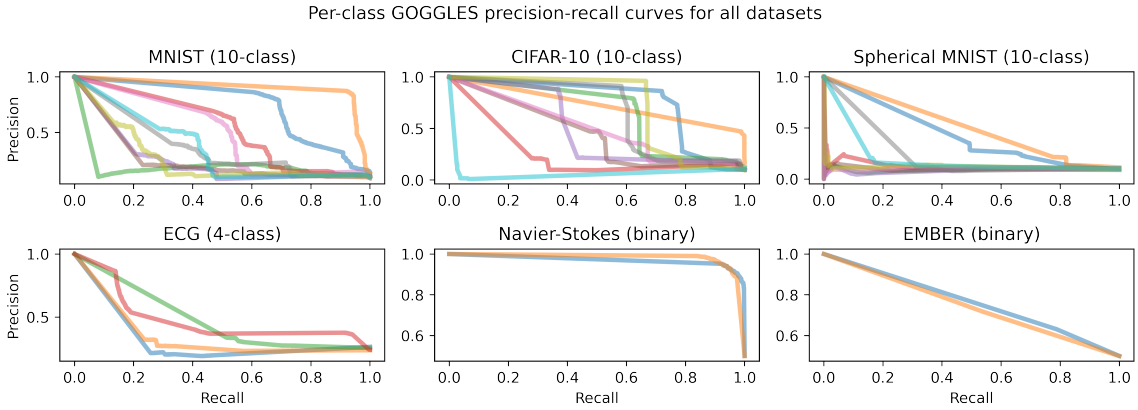


Figure 7.6: Class-wise precision recall curves using GOGGLES across datasets. Despite having 100% coverage on all tasks, GOGGLES has wildly varying per-class performance that becomes more pronounced as the number of classes increases. Permuted MNIST is not shown as the predictions are the same as on MNIST due to the permutation invariance of PCA.

Does GOGGLES improve with different clustering models? The default clustering model in GOGGLES is Gaussuan mixture model (GMM). We also compare the performance of label inference with another two clustering models: K-means clustering and Spectral clustering. The input of the clustering model is the affinity matrix computed by cosine similarity scores between embeddings of each example. After we train clustering model

Table 7.4: Different clustering methods for GOGGLES.

	MNIST	CIFAR-10	Spherical MNIST	Permuted MNIST	ECG	Navier-Stokes	Ember	YouTube	Yelp	IMDb
GOGGLES (GMM)										
+Raw features	0.3778	0.1697	0.0897	0.3778	0.2710	0.5284	0.5283	0.5880	0.4732	0.5516
+PCA	**0.4563**	0.1989	0.1074	**0.4561**	0.2688	0.4863	**0.6728**	0.5880	**0.5466**	0.5628
+ResNet-18	0.3258	0.1768	0.1192	0.1894	**0.3127**	0.9353	0.5312	0.5200	0.4553	0.5152
+CLIP	0.3309	0.5436	**0.1884**	0.1405	n/a	0.7505	n/a	0.5280	0.4779	0.5000
GOGGLES (K-means)										
+Raw features	0.3808	0.1416	0.1092	0.3808	0.2715	0.5226	0.5294	0.5320	0.4750	0.5344
+PCA	0.4099	0.1738	0.1136	0.3960	0.2672	0.4826	0.6720	0.5960	0.5413	**0.5720**
+ResNet-18	0.3294	0.1877	0.1112	0.2001	0.2814	0.9416	0.5296	0.5200	0.4574	0.5136
+CLIP	0.3730	**0.5604**	0.1342	0.1400	n/a	0.7063	n/a	0.5280	0.4779	0.5000
GOGGLES (Spectral)										
+Raw features	0.0355	0.1006	0.0938	0.0355	0.2305	0.5142	0.5294	0.5320	0.4626	0.5308
+PCA	0.1308	0.1002	0.1033	0.0814	0.2356	0.4874	0.6639	**0.6120**	0.5413	0.4356
+ResNet-18	0.1541	0.1165	0.0705	0.1130	0.2383	0.9168	0.5249	0.5480	0.5405	0.5152
+CLIP	0.1923	0.0953	0.1130	0.1480	n/a	**0.9858**	n/a	0.5160	0.4779	0.4996

and obtain clusters, GOGGLES use our 100 labeled examples to map clusters to classes. In Table 7.4, we see that GMM often outperforms the other clustering models on seven of the ten datasets. We conclude that GMM clustering is often good enough, but that a different cluster model might improve performance slightly.

7.D Ablation study of Interactive Weak Supervision

We perform an ablation study of IWS. In particular, we evaluate the effect of using IWS in it's original interactive form using a human-in-the-loop against the automatic selection rule used in the rest of our experiments. We furthermore evaluate the effect of tuning the accuracy threshold parameter of the automatic selection rule.

Does Interactive Weak Supervision require a human-in-the-loop? We evaluate the effect of relying on the automatic selection rule of IWS as opposed to using the original human-in-the-loop during the LF selection process. We consider three datasets: MNIST, Navier-Stokes, and Yelp, each subject to using the highest performing embedding using IWS according to Table 7.1 (CLIP, ResNet-18, raw BERT features, respectively). In the human-in-the loop setting, during each round of interaction, we prompt the user with a set of several scores: coverage, precision, recall, and accuracy, each computed on the validation set of 100 labels. Given this information, we ask whether or not the LF should be used. We

find that on MNIST, the human-in-the-loop procedure results in very similar accuracies and coverage scores to that of the automated procedure. On Navier-Stokes, however, the human-in-the-loop procedure results in much higher accuracy while retaining the perfect coverage of the automated procedure. Finally, on Yelp, the human-in-the-loop procedure actually worsens performance and significantly reduces coverage. We postulate that this is because the difficulty of estimating the quality of the synthesized LFs varies considerably dataset-to-dataset, given these scores. It is furthermore not always clear that using a similar strategy to that of the automated procedure is the best strategy. From this analysis, we conclude that a practitioner should consider trying the human-in-the-loop procedure, but does not necessarily guarantee improved results over the automated procedure.

Table 7.5: The automatic selection rule used in IWS leads to improved accuracy and coverage over its human-in-the-loop counterpart on MNIST, Navier-Stokes, and Yelp. Results are presented in the form of "accuracy (coverage)" pairs.

	MNIST + CLIP	Navier-Stokes + ResNet-18	Yelp + Raw features
Human-in-the-loop	0.6826 (0.2766)	0.9753 (1.0000)	0.5675 (0.5377)
Automatic	0.6885 (0.2758)	0.7995 (1.0000)	0.6034 (1.0000)

Does the automatic selection rule of Interactive Weak Supervision require tuning? We compare the effect of tuning the accuracy threshold used to automatically select LFs when using IWS under its automatic settings (i.e., without a human-in-the-loop). We perform this analysis using the same data settings as in the previous analysis: MNIST with CLIP features, Navier-Stokes with ResNet-18 features, and Yelp with raw features (BERT). We present these results in Figure 7.7. We find that tuning this threshold parameter does not improve performance by much, if at all—and if the accuracy threshold parameter is set to be too large, then coverage can drop, as is the case for Navier-Stokes and Yelp (see Figure 7.7 center and right). In the case of MNIST, this parameter had essentially no impact on accuracy or coverage.

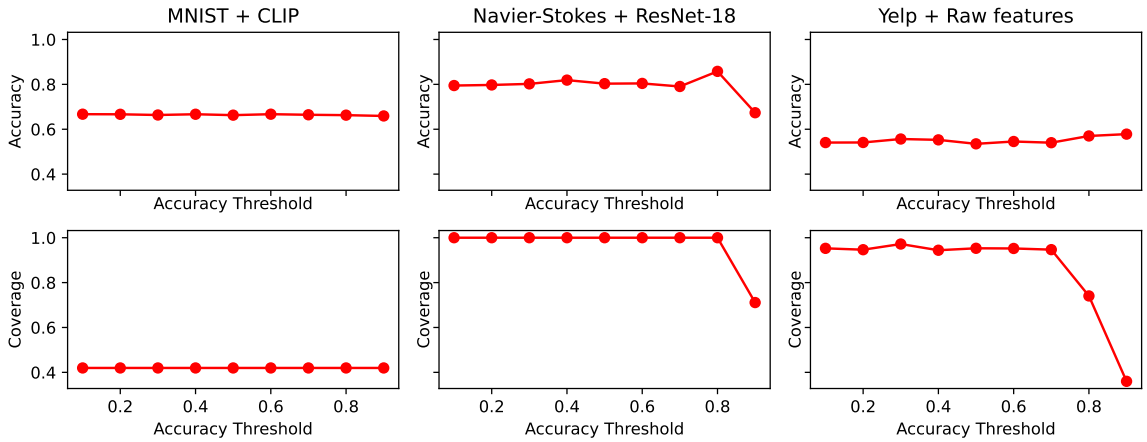


Figure 7.7: Tuning the threshold parameter of the automated version of IWS does not lead to significant improvements in performance, and can hurt coverage if it is set to be too high.

7.E Comparison to hand-designed LFs

So far, all of our results have looked at various flavors of AutoWS, and, in addition, examined few-/zero-shot capable models like CLIP. A natural question is whether AutoWS is competitive against manual WS. Indeed, this is the original motivation for AutoWS methods. This question is challenging to benchmark, since it is a function of the quality of the LFs. For example, a perfectly accurate manually-written LF cannot be improved upon. Instead, we provide a simple proof-of-concept.

We compare against a set of pre-existing manually-written LFs from [Fu et al., 2020] for the basketball dataset. These were also used as part of the WRENCH [Zhang et al., 2021a] benchmark. We then ran SNUBA on the dataset using our standard configuration. We found that SNUBA performs nearly as well as the gold (supervised) setting, while the human LFs are nearly 20 points worse. An additional question is whether SNUBA-generated LFs combine well with manual LFs. While this is a promising direction for future research, simply including both sets of LFs did not outperform the automatically generated ones.

7.F CLIP Zero-shot Prompts

We provide the prompts that we used with CLIP.

Table 7.6: Comparison between Snuba and hand-designed LFs on the Basketball dataset.

	Gold	Human LFs	Snuba LFs	Human LFs + Snuba LFs
F1 Score	64.97	43.18	62.55	61.39

Table 7.7: Prompts used to get zero-shot CLIP embedding

Dataset	Prompt
MNIST	"a photo of the number: "[label]"."
CIFAR-10	"a photo of a [label]."
Spherical MNIST	"a photo of the number: "[label]" stereographically projected onto a sphere and back to the plane."
Permuted MNIST	"a photo of the number: "[label]" permuted by a bit reversal permutation."
Navier-Stokes	["The Navier-Stokes equation in the turbulent regime with low viscosity: 1e-5.", "The Navier-Stokes equation with high viscosity: 1e-3."]
YouTube	["Ham", "Spam"]
Yelp	["Negative", "Positive"]
IMDb	["Negative", "Positive"]

7.G Incompatibilities between methods, feature representations, and tasks

As noted in Table 7.1, several methods, feature representations, and tasks are incompatible, resulting in several missing entries denoted as n/a. Here, we itemize and elaborate on the reasons for each missing entry.

- CLIP features and ECG: CLIP does not support ECG as it comprises time-series vectors.
- CLIP features and Ember: CLIP does not support Ember as it comprises feature vectors.
- CLIP features and IWS on binary classification tasks: For binary classification tasks, CLIP produces feature vectors with only two entries corresponding to its per-class logits. This only allows for a small handful of LFs to be generated. However, in this case, IWS requires more LFs than can be generated from this two-dimensional logits

vector while still using the same procedure used elsewhere. This limitation applies to Navier-Stokes, YouTube, Yelp, and IMDb, as they are all binary classification tasks.

Part III

Adaptation to Scientific Domains

Chapter 8

The AUP Score: Aggregating Performance Across Diverse Tasks

This chapter presents the Area Under the Performance Profile (AUP) score, which was introduced in the AutoML Decathlon [Roberts et al., 2022a] (Chapter 9) and subsequently adopted in MLGym [Nathani et al., 2025], the AutoML Cup, and MLCommons AlgoPerf [Dahl et al., 2023]. The material in this chapter is drawn from joint work with Deepak Nathani, Lovish Madaan, and others on MLGym.

Contributions: The author developed the AUP score and advised on the evaluation methodology.

A recurring challenge across the preceding chapters is the problem of comparing methods across a diverse set of tasks, each with its own domain-specific metric. Common approaches such as averaging scores or rankings across tasks can weight metrics in undesirable ways and disproportionately penalize certain methods. Averaging across different metrics may unfairly weight them based on their relative scales, and averaging ranks can disproportionately penalize methods that effectively solve a task but are tied with others. Rather than naive averaging, we employ performance profile curves [Dolan and Moré, 2002b], which allow us to compare relative performance gains across both methods and tasks. Performance profiles were originally developed to compare optimization techniques across a set of optimization problems. Since then, they have been used by the AutoML community to compare AutoML methods across diverse domains, each with their own domain-specific metrics [Tu et al., 2022a, Roberts et al., 2022b].

One challenge when using performance profiles is that they produce a curve for each method (where a higher curve is better), rather than a direct ranking of methods. To address this, the AutoML Decathlon [Roberts et al., 2022a] introduced the AUP score, which computes the area under the performance profile curve for each method, where a higher value constitutes better performance. Variants of the AUP score have since been used to score the AutoML Cup and MLCommons AlgoPerf [Dahl et al., 2023] competitions. In this chapter, we define performance profiles and the AUP score, and demonstrate their usage within MLGym [Nathani et al., 2025], a recent benchmark for evaluating LLM agents on AI research tasks.

8.1 Performance Profiles and the AUP Score

For a given method m, its performance profile curve is defined as

$$\rho_m(\tau) = \frac{1}{|T|} |\{t \in T : \log_{10} r_{t,m} \leqslant \tau\}| \qquad r_{t,m} = \frac{\ell_{t,m}}{\min\{\ell_{t,m} : m \in M\}} \tag{8.1}$$

where M is the set of all methods, T is the set of tasks, $\ell_{t,m}$ is the performance metric for a method m on task t, and $r_{t,m}$ is a quantity called the *performance ratio*.

Importantly, this definition assumes that the performance metric for each task, $\ell_{t,\cdot}$, must be defined such that lower scores are better—we discuss modifications to support other scores below. Performance profiles are parameterized by a threshold, τ, on the distance between the method m and the best scoring methods on each of the tasks. At a given threshold τ, performance profiles compute the proportion of tasks for which the method m is within τ of the best method for each task.

In order to derive a final score for each method $m \in M$, we compute the AUP score as

$$\text{AUP}_m = \int_1^{\tau_{\max}} \rho_m(\tau) d\tau, \tag{8.2}$$

where $\tau_{\max}$ is the minimum τ for which $\rho_m(\tau) = 1$ for all $m \in M$.

8.1.1 Practical Considerations

In practice, several adaptations are needed to apply performance profiles and AUP scores:

- Metric Direction Handling. For metrics where higher values are better (e.g., accuracy), we invert the performance ratio calculation and use the maximum score instead of the minimum:

$$r_{t,m} = \frac{\max\{\ell_{t,m} : m \in M\}}{\ell_{t,m}}. \tag{8.3}$$

- Infeasible Methods. Methods that fail to produce a valid solution or underperform the baseline are marked as *infeasible*. The score of an infeasible method is set to $(1+\varepsilon) \times r_{t,m_{\text{baseline}}}$, where $r_{t,m_{\text{baseline}}}$ is the score obtained by the baseline method on task t and ε is a small constant (e.g., $\varepsilon = 0.05$).

8.2 Application: Evaluating LLM Agents on Research Tasks

To demonstrate the AUP score in practice, we apply it to evaluate frontier LLMs on MLGym-Bench [Nathani et al., 2025], a benchmark of 13 diverse AI research tasks spanning computer vision, natural language processing, reinforcement learning, and game theory.

Figure 8.1 shows the performance profiles for Best Attempt and Best Submission across five frontier models. Table 8.1 shows the corresponding AUP scores.

Model	Best Attempt AUP@4	Best Submission AUP@4
Llama3.1-405b-instruct	1.015	1.039
Claude-3.5-Sonnet	1.142	1.135
Gemini-1.5-Pro	1.140	1.125
GPT-4o	1.000	1.029
OpenAI O1	**1.150**	**1.176**

Table 8.1: AUP@4 scores for the best attempt and best submission across all models.

OpenAI O1-preview achieves the highest AUP score for both Best Attempt and Best Submission, with Gemini 1.5 Pro and Claude-3.5-Sonnet close behind. While O1-preview is not dominant on every individual task, it is consistently among the top-performing models across most tasks—precisely the kind of aggregate consistency that the AUP score is designed to capture.

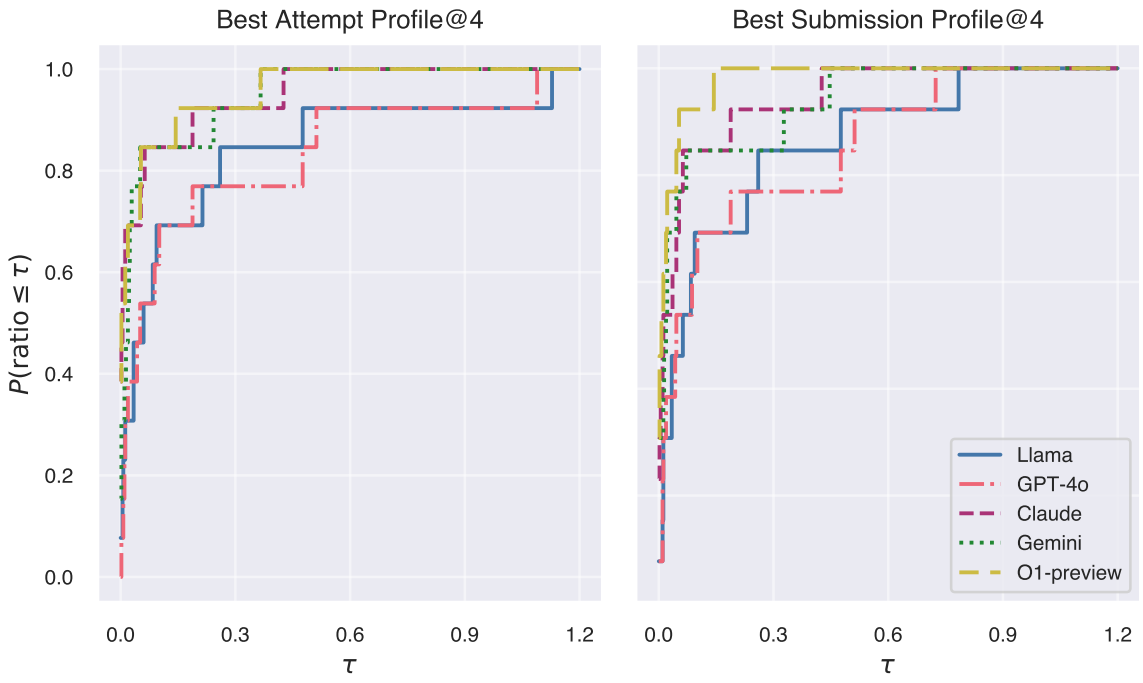


Figure 8.1: Performance profiles comparing Best Attempt@4 and Best Submission@4 across all models and tasks. The x-axis shows the performance ratio threshold τ and the y-axis shows the fraction of tasks where a model achieves performance within τ of the best model.

The AUP score also enables cost-performance analysis. Figure 8.2 shows that Gemini-1.5-Pro achieves 99% of O1's AUP while being approximately 9$\times$ cheaper, striking the best balance between performance and cost.

8.3 Discussion

The AUP score provides a principled way to aggregate performance across tasks with heterogeneous metrics. Unlike simple averaging, it accounts for the relative difficulty of each task and the distribution of method performance. Unlike rank-based aggregation, it preserves information about the magnitude of performance differences. These properties make it well-suited for the diverse-task evaluation settings studied throughout this thesis, and its adoption in multiple subsequent competitions and benchmarks validates its practical utility.

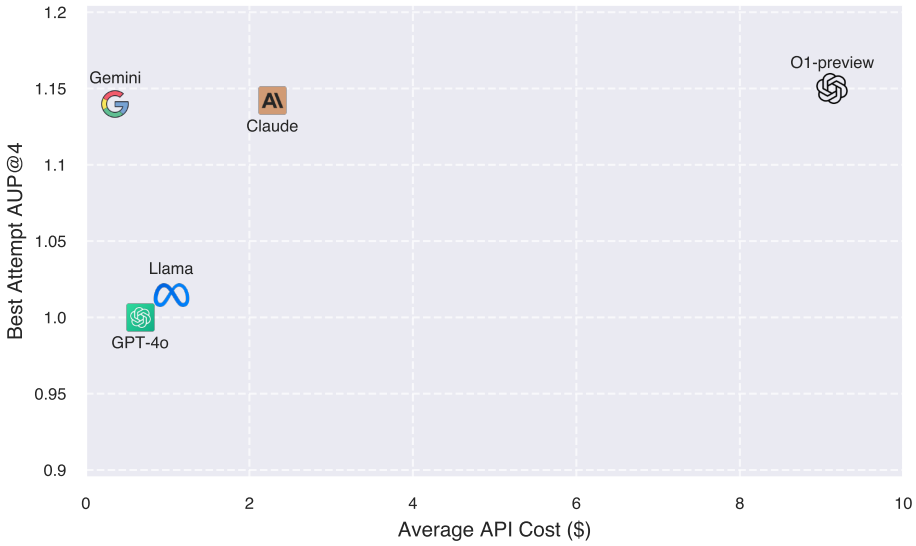


Figure 8.2: Best Attempt AUP@4 vs. API cost for all models. The AUP score enables direct comparison of cost-performance tradeoffs across models evaluated on tasks with different metrics.

Chapter 9

AutoML Decathlon: Diverse Tasks, Modern Methods, and Efficiency at Scale

This chapter is based on joint work with Samuel Guo, Cong Xu, Ameet Talwalkar, and others, published at NeurIPS 2022 (Competition Track). The first three authors contributed equally.

Automated Machine Learning (AutoML) is an important field offering substantial possibilities in expanding the use of machine learning to the broader community. It reduces the need for human involvement in the process of building machine learning pipelines by automating design choices that would have otherwise required experience. The AutoML vision is to produce performant systems with virtually no need for human input or domain expertise. If fully achieved, this goal will have significant implications for both industrial applications [Jumper et al., 2021, Degrave et al., 2022], academic research [Real et al., 2020a, Fawzi et al., 2022], and more.

Benchmarks are a crucial tool for measuring progress towards the AutoML vision. While successful as a yardstick for AutoML techniques in certain scenarios, much of the prior efforts in benchmarking have had a myopic focus on domains that have already received significant attention from the ML community. These include vision and language. AutoML techniques can lead to improvements on problems in these well-studied domains, but its larger promise lies in the long tail of less-studied ML tasks from diverse domains. For these

under-studied tasks, dramatically less human effort has gone into designing tailor-made ML pipelines.

AutoML for diverse tasks is an emerging area aiming to take advantage of these opportunities. This promising line of work includes a number of recently-proposed methods [Shen et al., 2023, 2022, Roberts et al., 2021b] and benchmarks [Tu et al., 2022b, Roberts et al., 2022c]. To further accelerate progress in this area, we ran the AutoML Decathlon, an AutoML competition centered on diverse tasks. Participants were given a set of 10 machine learning tasks which were diverse along various axes: domain, input type/shape, output type/shape, learning objective, evaluation metric, and scale. To discourage overfitting to a specific domain or task type, a separate, unobserved set of 10 additional diverse tasks were used to evaluate the submitted methods. Ultimately, we received 24 submissions, with nine of them outperforming our baselines at the conclusion of the competition.

This report consists of a full breakdown of the design of the competition, including task details, evaluation, infrastructure, hackathons, an analysis of the results, a set of lessons learned, and implications for the future of AutoML. In particular, we highlight the following key takeaways from the AutoML Decathlon:

- top methods combined standard AutoML approaches with a wide variety of neural network architectures, large scale transfer learning, and simpler ML models,
- existing methods targeting AutoML for diverse tasks perform strongly,
- methods that performed strongly across task types outperformed specialized methods.

We conclude that a combination of modern ML methods—transfer learning, modern architectures, and differentiable NAS—with advanced Hyperparameter Optimization (HPO) and ensembling is a promising direction for the field of AutoML for diverse tasks.

9.1 Competition setup

The competition consisted of two phases: the development phase and the test phase. During the development phase, participants developed their methods for 10 public tasks with set time and submission limits. The test phase consisted of offline evaluations of final submissions on 10 private test tasks. Unlike previous AutoML competitions [Guyon et al., 2019, Liu et al., 2020], participants in the AutoML Decathlon were encouraged to consider a wide range

of approaches, from traditional hyperparameter optimization to modern techniques like NAS and large-scale transfer learning. The competition was challenging, but also offered a valuable opportunity for participants to gain experience in working with complex tasks and finding innovative solutions to difficult problems.

We hosted the competition on CodaLab [Pavao et al., 2022].[1] To ensure a smooth and timely execution of the competition, we made several modifications to the CodaLab Docker image, including virtual partitioning of a multi-GPU dual-socket server, prevention of test dataset label leaks, increase in shared memory size, fixation of CPU/GPU affinity, backup of the entire working directory for each submission for post-analysis, and various bug fixes.

Moreover, to accommodate the various deep learning frameworks used by participants, we provided a customized application Docker image.[2] This image included popular deep learning frameworks, such as TensorFlow [Abadi et al., 2015], PyTorch [Paszke et al., 2019], XGBoost [Chen and Guestrin, 2016a], and scikit-learn [Pedregosa et al., 2011b]. This ensured that participants could use their preferred framework without any compatibility issues.

9.1.1 Tasks

The competition is based on a set of 20 tasks that have been carefully selected to cover a diverse range of practical applications across scientific, technological, and industrial domains. In the development phase, we released 10 tasks, with task metadata, as shown in Table 9.1. These tasks differ in their domains, which include spherically projected images, financial timeseries, audio, and the natural sciences. They also cover different problem types such as regression, single-label, and multi-label classification. These tasks contain several thousand to hundreds of thousands of observations, with sizes ranging from 23 MB to 36 GB, making them complex and challenging to work with.

The aim of selecting 10 tasks in the evaluation phase, shown in Table 9.1, was to have a diverse set of data that is representative of the development phase, while avoiding a situation where the leaderboard is completely different from the development phase. For example, to complement the Navier Stokes task [Li et al., 2021d], we chose to use the shallow water PDE [Takamoto et al., 2022] as a test task, and to complement the Crypto development task, we included a high-frequency stock prediction test task [Ntakaris et al.]. Moreover, these

[1] `https://codalab.lisn.upsaclay.fr/competitions/6325`
[2] `https://hub.docker.com/r/automldec/decathlon`

tasks use different evaluation metrics for each of the tasks, although the evaluation metrics that were given as part of the development tasks were representative of those of the test tasks. While competitors had access to the development tasks throughout the competition, the test tasks remained private for the duration of the competition and was used solely for evaluation purposes. This ensured that participants were not able to access the test data beforehand, making the competition fair for all. Finally, to enable the use of modern AutoML methods, we allocated a relatively large compute budget for each task in comparison to other AutoML competitions—small tasks are allocated up to 5 hours and large tasks are allowed up to 20 hours on a single V100 GPU.

9.1.2 Evaluation and scoring metrics

Each individual task had a metric, standardized such that a lower score indicated better performance. These are all common metrics—such as 0-1 error, L2 relative error, the negative log likelihood (NLL)—or simple transformations thereof, e.g. 1—F1. To rank each method's performance on all tasks, we developed a score based on performance profiles [Dolan and Moré, 2001], called the Area Under Performance profile curve (AUP). First, the performance profile curve parameterized by τ is defined in the formula below:

$$\rho_s(\tau) = \frac{1}{|P|}\left|p \in P : \frac{\mathrm{LPM}_{p,s}}{\min_{s\in S}\mathrm{LPM}_{p,s}} \leqslant \tau\right|,$$

where the performance metric of method s on task p as $\mathrm{LPM}_{p,s}$. The AUP score of task s is computed by integrating this expression with respect to $\log(\tau)$. The upper bound, u, of the integral must be the smallest value such that $\rho_s(10^u) = 1\ \forall s$, or equivalently, this is the upper bound of the standard performance profile curve. Then the AUP score is given as

$$\mathrm{AUP}_s = \int_0^u \rho_s(\tau)\,d\sigma\ ,\ \sigma = \log_{10}(\tau).$$

This metric measures how often a method is within a multiplicative factor of the best score on the leaderboard for each task. The formula uses individual task scores to create a performance profile curve, plotting the proportion of task instances in which a particular task method is within a factor τ of the optimal method's score against $\log_{10}(\tau)$, then computing the area under the curve.

Table 9.1: Development and test task metadata. All listed metadata except for the domain was available to competitors during the competition.

Task	N	Type	Domain	In Dim.	Out Dim.	Size	Metric
Navier-Stokes [Li et al., 2021d]	1200	Continuous	PDE	(20, 1, 64, 64)	(64, 64)	396 MB	L2 rel. error
Spherical [Cohen et al., 2018a]	60000	Single-label	Image	(1, 3, 60, 60)	(100)	619 MB	0-1 Error
NinaPro [Atzori et al., 2012b]	3956	Single-label	CompBio	(1, 16, 52, 1)	(18)	26 MB	0-1 Error
FSD50K [Fonseca et al., 2017a]	51000	Multilabel	Audio	(1, 96, 101, 1)	(200)	36 GB	1 - MAP
Cosmic [Zhang and Bloom, 2020a]	5250	Multilabel	Image	(1, 1, 128, 128)	(128, 128)	7.7 GB	1 - AUROC
ECG [Clifford et al., 2017a]	330000	Single-label	CompBio	(1, 1, 1000, 1)	(4)	2.5 GB	1 - F1 Score
DeepSEA [et al., 2004]	250000	Multilabel	CompBio	(1, 4, 1000, 1)	(36)	896 MB	1 - AUROC
Nottingham [Bai et al., 2018]	1200	Multilabel	Audio	(1792, 88, 1, 1)	(88)	23 MB	NLL
Crypto*	1559	Continuous	Time-series	(3000, 13, 1, 1)	(600)	524 MB	L2 rel. error
EMBER [Anderson and Roth, 2018a]	900000	Single-label	Tabular	(1, 2381, 1, 1)	(2)	7.2 GB	0-1 Error

*The Crypto dataset was generously provided by Morgan Stanley.

Task	N	Type	Domain	In Dim.	Out Dim.	Size	Metric
SphericalTinyImageNet [Le and Yang, 2015]	100000	Single-label	Image	(1, 3, 30, 30)	(200)	1.6 GB	0-1 Error
Satellite [Petitjean et al., 2012]	900000	Single-label	Time-series	(46, 1, 1, 1)	(24)	359 MB	1 - F1 Score
JSBChorales [Bai et al., 2018]	18104	Multilabel	Audio	(40, 88, 1, 1)	(88)	100 KB	NLL
HumanGait [Vajdi et al., 2019]	47276	Single-label	Time-series	(128, 3, 1, 1)	(88)	155 MB	1 - MAP
SpokenMNIST [Jackson et al., 2018]	2700	Single-label	Audio	(1, 4, 64, 64)	(10)	47 MB	0-1 Error
Stock [Ntakaris et al.]	7234	Continuous	Time-series	(1000, 40, 1, 1)	(200)	2.4 GB	L2 rel. error
MYO [Côté-Allard et al., 2019]	224406	Single-label	Time-series	(12, 8, 7, 1)	(7)	722 MB	0-1 Error
ShallowWater [Takamoto et al., 2022]	3600	Continuous	PDE	(10, 1, 64, 64)	(64, 64)	625 MB	L2 rel. error
Year [Bertin-Mahieux et al., 2011]	463715	Continuous	Audio	(1, 90, 1, 1)	(1)	181 MB	L2 rel. error
HumanProteinAtlas [Sullivan et al., 2018]	3107	Multilabel	CompBio	(1, 1, 128, 128)	(128, 128)	777 MB	1 - AUROC

This AUP metric offers certain advantages over other commonly-used aggregate metrics such as average rank. AUP is more holistic because it factors in the degree of difference in performance between methods, rather than just the relative order of performance. For example, if there exists a task on which many methods achieve strong but relatively similar scores, the 10th-ranked method may not be far off in performance to the 1st-ranked method. In this case, the AUP metric would correctly account for the fact that the former method is still very good, whereas it would be penalized heavily by the average rank metric. Throughout the competition, we maintained a leaderboard of competitors' AUP on the development tasks, and the final ranking was determined by the AUP on the test tasks.

9.1.3 Compute resources

Hewlett Packard Enterprise (HPE) allocated three Apollo 6500 GPU servers, each equipped with eight NVIDIA V100s, and $50,000 in cloud credit resources. These servers were the compute backend that connected to the worker queue on CodaLab. When demand peaked approaching the end date of the competition, HPE also allocated up to 18 Google Cloud instances with the same V100 GPUs.

To ensure fairness, the CodaLab Docker images were modified to provide a virtual partition of the Apollo server that matches the same setup of the cloud instance. Each submission ran on a virtual machine equipped with an NVIDIA Tesla V100 GPU, eight vCPUs, and 30 GB of memory. This provided a powerful computing environment for participants to develop their methods. Additionally, one VM is dedicated entirely to the job of one participant during its execution, with minimum interference from other participants.

9.1.4 Development phase hackathons

With the goals of introducing university students at all levels to the problem of AutoML for diverse tasks and giving them initial exposure to the competition, we hosted three separate hackathons at Carnegie Mellon University, the University of Wisconsin-Madison, and the AutoML Fall School in Freiburg, Germany.[3] These hackathons were in-person and took place during consecutive weeks in October 2022.

[3] `https://sites.google.com/view/automl-fall-school-2022/home`

Each hackathon was run separately over the course of roughly 36 hours. An introductory presentation to the hackathon, the overarching competition, and the topic and motivation of AutoML was delivered before participants were made free to begin their development. Due to the brief time frames of the hackathons, their focus was to offer initial engagement with the competition and AutoML as whole, and ideally garner continuing interest until the competition's conclusion from some teams. As an additional incentive for engagement, we offered Amazon gift cards as prizes to teams that submitted a method to the official competition that surpassed certain baselines, or gave a presentation on the methods they implemented or researched. We provided hackathon participants with a self-contained Jupyter notebook that simulated and walked through the real competition pipeline and automatically downloaded a subset of the development tasks. Additionally, as many students were unfamiliar with AutoML, we provided a list of basic suggested approaches. The hackathons resulted in several submissions to the competition that explored a wide range of approaches. Some teams from the hackathons became serious competitors after improving their methods for the remainder of the competition.

9.1.5 Test phase

On the last day of the development phase, we encountered a platform bug which prevented submissions from being automatically moved to the test phase. To account for this, we allotted extra time for participants to make their final submissions via email, and we proceeded with the test phase manually. Consequently, we worked with teams to migrate their code to our manual test phase runs. We provide full details about how this process was conducted in Appendix 9.D.

Results post-processing. During the test phase, two teams—Team TrueFit and Team TEG—both achieved perfect test performance on the HumanGait task, which resulted in scores of 0.0. Several other entries achieved similarly high scores, differing only by 1-2 misclassified examples, and were affected by the magnitudes of their confidence scores. However, perfect scores of 0.0 are not supported by performance profiles because these scores appear as a divisor—therefore, we could not directly compute the final AUP scores using these entries. We considered several options for fixing this issue: adding a small epsilon to the perfect scores $\epsilon_{\text{perfect}} \in (0, \text{next\_best}]$, adding a small epsilon to all of the

scores $\epsilon_{\text{all}} > 0$, removing the task entirely, or setting a nonzero best possible score for the task $\epsilon_{\text{min\_score}} > 0$. Here, next_best refers to the next best nonzero score. We found that under reasonable choices of $\epsilon_{\text{perfect}}$, ϵ_{all}, or $\epsilon_{\text{min\_score}}$, all of the solutions resulted in the same winner. Ultimately, we set a minimum possible score for the task, and used a value of $\epsilon_{\text{min\_score}} = 1e-7 <$ next_best.

9.2 Results and analysis

In the test phase, nine submissions ultimately outperformed the provided baselines. In this section, we provide details about the top-5 highest performing methods on the test tasks.

Table 9.2: Development and test task leaderboards. These have been trimmed to show only the top-11 methods to include all methods that achieved the best score on any of the tasks—the complete leaderboards are given in Appendix 9.A. Methods are ranked according to their AUP scores (not shown). Lower is better for all tasks.

Team	Navier Stokes	Spherical	NinaPro	Cosmic	ECG	Deep-SEA	Nott-ingham	Crypto	Ember	FSD50K
euxhenh	0.1659	0.7332	0.1290	**0.0243**	0.3792	0.3767	0.0241	**0.0020**	0.0462	0.8560
Freiburg-AutoML-Lab	**0.0858**	0.7928	0.1290	0.0477	0.8910	**0.3208**	**0.0148**	0.9990	0.0392	0.7211
TEG-AutoML	0.1892	0.9771	0.1426	0.4983	**0.3205**	0.4992	0.1168	0.0039	0.0685	0.7702
KaiWu	0.1892	0.9771	0.1426	0.4983	**0.3205**	0.4992	0.1168	0.0039	0.0685	0.7702
Team 42	0.9999	0.9016	0.1244	0.4607	0.4685	0.4277	0.0486	0.0270	**0.0247**	**0.6150**
TrueFit	0.6648	0.8809	**0.1168**	0.4983	0.6730	0.3873	0.0177	0.0638	0.0257	0.9738
Minions	0.2925	0.9671	0.2367	0.4716	0.4864	0.4989	0.2607	0.0058	0.0547	0.9598
XGBBaseline	0.2848	0.9660	0.2109	0.4689	0.5400	0.4992	0.2607	0.0055	0.0686	0.9636
dragon_bra	0.9999	0.9443	0.1973	0.4983	0.5576	0.4990	0.2674	0.0021	0.0775	0.9587
miracle-flex	0.2538	0.8947	0.2367	0.4983	0.8910	0.4992	0.0548	0.0111	0.4043	0.9738
automl	0.9999	**0.7316**	0.1684	0.4540	0.8910	0.4992	0.0506	0.0074	0.4043	0.9738

Team	Shallow Water	Satellite	Spherical Tiny ImageNet	JSB Chorales	Human Gait	Spoken MNIST	Stock	MYO	Year	HPA
TrueFit	0.0697	0.3764	**0.9798**	0.1228	**1e-7***	0.0067	0.0325	0.0279	0.0042	0.0219
TEG-AutoML	0.0474	0.3003	0.9903	0.1110	**1e-7***	0.0067	0.0822	0.1331	0.0033	**0.0190**
Freiburg-AutoML-Lab	**0.0004**	**0.2256**	0.9896	0.1017	2.281e-6	0.1200	0.0412	0.0400	0.0031	0.3883
euxhenh	0.0964	0.4059	0.9965	0.1273	1.171e-5	0.0100	**0.0208**	0.0226	0.0034	0.0302
tak	0.0408	0.2423	0.9895	0.4735	1.087e-7	0.0333	0.0490	0.3742	0.0033	0.2198
automl	0.0461	0.2378	0.9950	0.5013	9.417e-7	**0.0033**	0.0519	0.3742	0.0034	0.2227
42	0.0343	0.2460	0.9836	0.5160	7.244e-7	0.0100	0.0660	0.3742	0.0033	0.2091
Paris-Saclay	0.0119	0.2334	0.9949	**0.0973**	0.9192	0.0433	0.0620	**0.0082**	**0.0030**	0.1793
dragon_bra	0.0006	0.3330	0.9945	3.8154	0.0068	0.1167	0.0597	0.0461	0.0032	0.6127
XGBBaseline	**0.0004**	0.6072	0.9832	4.1342	0.2557	0.1333	0.0832	0.0362	0.0033	0.3871
miracle-flex	0.0293	0.5721	0.9995	0.1134	0.0010	0.4333	0.0570	0.3742	0.0710	0.2059

*Post-processed to avoid scores of 0.0, which are not supported by AUP.

9.2.1 Top-5 approaches

Team TrueFit. Team TrueFit finished in first place. They focused on architectural range, parameter tuning, and efficient experiments. They began with a transformer with time-based tokenization for sequence and signal data, and included a CNN for image classification, a U-Net for segmentation, and LightGBM for tabular data. They sampled hyperparameters from wide yet reasonable distributions, with random then evolutionary sampling during training. All models were tuned over 6-15 cross-validation runs, followed by full retraining. They note that including an MLP, time-series module, and similar techniques would further extend their winning solution. Further discussion and source code is available on GitHub.[4]

Team TEG-AutoML. The second place team, Team TEG-AutoML, designed a solution based on DASH [Shen et al., 2022]. This solution involved tailoring various templates to potential downstream tasks to guide a combination of differentiable NAS and HPO. Their NAS approach used a wide ResNet [Zagoruyko and Komodakis, 2016a] backbone with attention modules [Hu et al., 2018] and self-distillation heads could transform into a specialized network for the given task. Normalization was applied to the inputs in order to improve optimization. Their solution also incorporated a trial scheduling mechanism, allowing for multiple runs of their full AutoML pipeline within the time budget, attempting several random seeds and matched templates. They included system-level optimizations such as automatic mixed precision, pin-memory, and persistent workers to speed up the data loading and training process. Their solution achieved a strong second-place ranking, demonstrating the versatility of morphism-based AutoML for the problem of diverse tasks.

Team Freiburg-AutoML-Lab. The third place team, Team Freiburg-AutoML-Lab, created a taxonomy that determines the task type based on the input shape and trains multiple models that work well for this task. Afterward, they created ensembles for all possible model combinations and they selected the best-performing ensemble. Their handcrafted decision tree maps datasets to a subset of the model portfolio, which consists primarily of neural networks. Furthermore, their solution involved the use of pre-trained models, early stopping, and an "LR range test" [Smith, 2017], to find learning rates for the neural

[4] `https://github.com/truefit-ai/auto-ml`

networks. This solution could be extended by using Meta-Learning or applying HPO. The implementation of their approach is available on GitHub.[5]

Team euxhenh. Team euxhenh finished in fourth place with a solution that aimed to pick the right model family for a given task type. This was determined a hand-designed decision tree by considering the input dimensions of the task. Apart from tasks whose input consist of 1D feature vectors (which use XGBoost), all other task types were assigned deep learning-based methods. These varied from gated recurrent unit (GRU) networks for time series data to wide ResNets for channeled inputs, with special consideration given to 2D time-series and tasks with 2D outputs. The solution used simple model architectures (some containing only two layers) and it used fixed hyperparameters. This could be further improved by incorporating HPO or NAS techniques. Overall, this simple approach managed to beat the baselines by a large margin. The implementation is available on GitHub.[6]

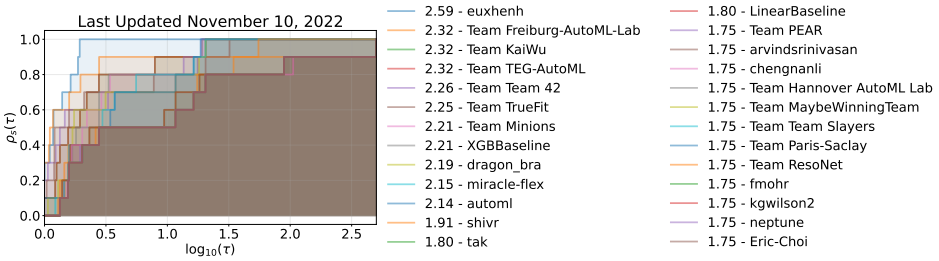
Team tak. Team tak finished in fifth place with an ensemble-based solution containing 2 main components. The first component included an HPO phase. Inspired by the H2O AutoML approach,[7] for each algorithm they executed a grid search to find the best hyperparameters for each algorithm they consider. The second component was a weighted average of different algorithms with the selected hyperparameters. They observed that no single algorithm worked best in all scenarios. They conclude that they needed to execute enough experiments to choose the best algorithms, hyperparameters, and ensemble weights.

Discussion of top methods. The top-5 teams combined existing HPO and/or portfolio selection methods with a wide variety of modern, specialized neural network architectures, pretrained models for transfer learning, and NAS techniques in their solutions. Team TrueFit used transformers, CNNs, and a U-Net architecture. Team TEG-AutoML used a variant of the wide ResNet architecture with attention modules in conjunction with DASH, while Team Freiburg-AutoML-Lab uses pretrained models for transfer learning with ensemble selection. Team euxhenh used GRU, ConvGRU, and wide ResNet architectures, and Team tak used an ensemble method inspired by H2O, an existing AutoML package. Four of the

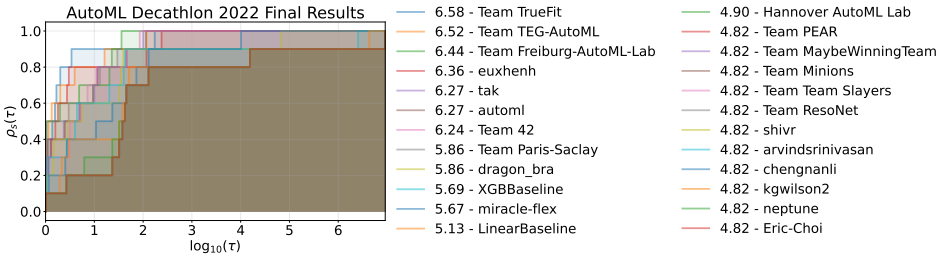
[5] `https://github.com/automl/AutoML-Lab_Decathlon`
[6] `https://github.com/euxhenh/automl-decathlon-2022-eh`
[7] `https://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html`

five top teams used some form of HPO in their solutions, and all of the teams included simpler ML models as fallback methods. *We conclude that a key feature of the top solutions was the combination of modern machine learning methods—modern architectures, transfer learning, and differentiable NAS—with advanced HPO or ensembling.*

Figure 9.1: Final AUP plots. The ordering of the legend from top-left to bottom-right indicates the ranking according to the AUP scores (where higher is better).



(a) Results on dev tasks, as of original competition deadline.



(b) Final results on the test tasks.

9.2.2 Comparison to baselines

We included the source code for two baselines at the beginning of the competition: a linear model and a stronger XGBoost [Chen and Guestrin, 2016a] baseline. At the conclusion of the competition, we compared the final leaderboard results to two additional baselines: DASH [Shen et al., 2022] and AutoGluon [Erickson et al., 2020], which are both targeted specifically at the diverse tasks problem.

Linear baseline. The linear baseline consisted of a single fully-connected layer with appropriate input and output dimensions.

XGBoost baseline. The XGBoost baseline was implemented using the XGBoost package. XGBoost uses gradient boosted tree models for classification and regression tasks, and it supports multiple outputs.

DASH Baseline. We evaluated DASH on the test tasks using 1D, 2D, and 3D wide ResNet [Zagoruyko and Komodakis, 2016a] backbones. DASH uses a search space of several convolutional kernel sizes and dilation rates and an efficient differentiable search method.

AutoGluon Baseline. Finally, we evaluated an AutoGluon baseline. Because AutoGluon's Tabular Predictors are designed for single-label tasks, this baseline falls back to XGBoost for multi-output tasks.

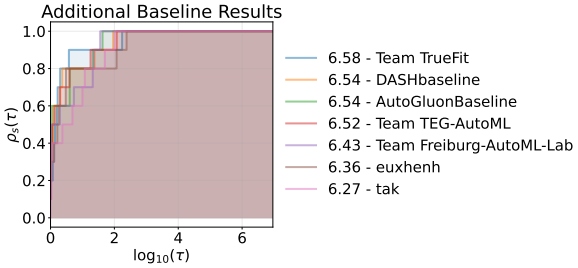


Figure 9.2: AUP plot comparing additional baselines to the top-5 submissions. DASH and AutoGluon achieve competitive performance.

Discussion of baselines. We ran these baselines with minimal tuning. Of these methods, DASH and AutoGluon achieved competitive performance—ranking in second and third place, respectively. We provide a more in-depth discussion of baseline performance in Appendix 9.B. *We conclude that these two methods, both of which were designed specifically for the diverse tasks problem, are strong baselines that should be considered in future work on AutoML for diverse tasks.*

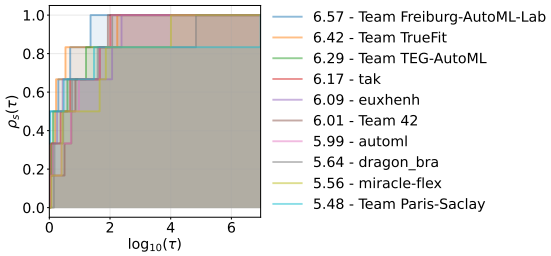
9.2.3 Results by task type

We compare the top-10 rankings of submissions among different subsets of the test tasks. In particular, we compare submissions among the following categories of tasks: time-dependent, non-time-dependent, single-label classification, and multilabel or continuous test tasks. We chose these categories to compare across different input (Figures 9.3(a), 9.3(b)) and output characteristics (Figures 9.3(c), 9.3(d)) of the tasks.

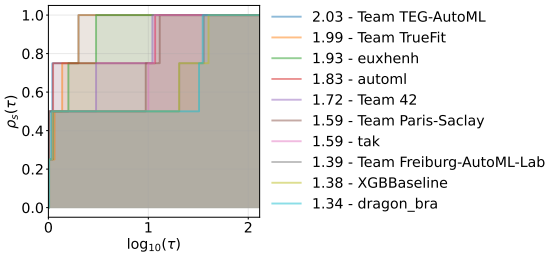
The winning submission, Team TrueFit, performs the best on single-label classification tasks (Figure 9.3(c)), and is ranked second in all other categories. Team TEG-AutoML

achieves the highest performance on tasks with no time dependence (Figure 9.3(b)), and is in the top-5 for all other subsets (Figures 9.3(a), 9.3(c), 9.3(d)). Team Freiburg-AutoML-Lab is first for time-dependent and multilabel/continuous tasks (Figures 9.3(a), 9.3(d)), but is less performant on single-label classification tasks and tasks with no time dependence (Figures 9.3(b), 9.3(c)). *We conclude that some submissions were more specialized for tasks with certain input or output characteristics, while submissions that performed strongly across all task types ultimately performed best overall.*

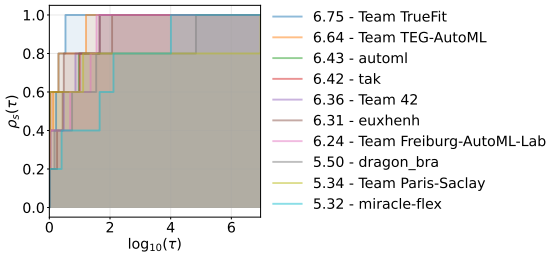
Figure 9.3: Top-10 rankings across different types of test tasks. Top overall submissions perform well on each subset, although the rankings differ across subsets.



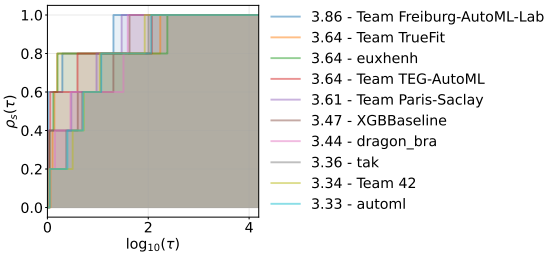
(a) Test tasks *with time dependence.*



(b) Test tasks *with no time dependence.*



(c) *Single-label classification* test tasks.



(d) *Mutlilabel or continuous* test tasks.

9.3 Conclusion

We ran the AutoML Decathlon 2022 competition to stoke interest in AutoML for diverse tasks—an emerging and promising research area within AutoML. We received a total of 24 submissions during the development phase, with nine of them outperforming our two baselines in the test phase of the competition. We also ran a series of hackathons to prepare participants for the competition and to further promote the AutoML for diverse tasks problem. In our post-hoc analysis, we found that the top submission even outperformed existing

state-of-the-art AutoML methods targeted at diverse tasks. *We conclude that combining advanced HPO with modern ML methods—including transfer learning, modern architectures, and NAS—is a promising direction for the field of AutoML for diverse tasks.*

In future work, we hope to run a follow-up competition that explores diverse tasks beyond standard supervised learning in diverse domains—e.g., limited access to labeled data. More generally in the longer term, we hope to continue to promote and explore this emerging research area via competitions, blogs, benchmarks, and new methods.

9.A Development and Test Leaderboards

In this section, we provide the complete development and test leaderboards. These are shown in Figure 9.3.

9.B Additional Baseline Performance

The DASH baseline maintains strong results both in overall performance as well as individual task performance, claiming the top method spot in a couple tasks. As mentioned, the AutoGluon baseline substitutes XGBoost for multioutput tasks: ShallowWater, JSBChorales, Stock, and HPA. The AutoGluon baseline owes it's excellent ShallowWater result to XGBoost, but on JSBChorales and HPA the performance is not as strong. On the remaining tasks, AutoGluon performs well.

9.C Analysis of Per-task Rank and Compute Time

Figure 9.4(a) shows the per-task rank and average ranks across tasks for each team that outperformed the linear baseline. Team TrueFit achieves top performance on SphericalTiny-Imagenet and Humangait with an average rank of 4.4. Team TEG-AutoML wins on HPA and Humangait with an average rank of 5.2. Finally, team Freiburg-AutoML-Lab wins on Satellite and ShallowWater with an average rank of 4.4. If we used average rank instead of AUP, both TrueFit and Freiburg-AutoML-Lab would be the winning teams. However, AUP was able to identify the significant performance gap between Freiburg-AutoML-Lab and optimal solutions on Humangait and HPA.

Table 9.3: Development and test task leaderboards.

Team	Navier Stokes	Spherical	NinaPro	Cosmic	ECG	Deep-SEA	Nott-ingham	Crypto	Ember	FSD50K
euxhenh	0.1659	0.7332	0.1290	**0.0243**	0.3792	0.3767	0.0241	**0.0020**	0.0462	0.8560
Freiburg-AutoML-Lab	**0.0858**	0.7928	0.1290	0.0477	0.8910	**0.3208**	**0.0148**	0.9990	0.0392	0.7211
TEG-AutoML	0.1892	0.9771	0.1426	0.4983	**0.3205**	0.4992	0.1168	0.0039	0.0685	0.7702
KaiWu	0.1892	0.9771	0.1426	0.4983	**0.3205**	0.4992	0.1168	0.0039	0.0685	0.7702
Team 42	0.9999	0.9016	0.1244	0.4607	0.4685	0.4277	0.0486	0.0270	**0.0247**	**0.6150**
TrueFit	0.6648	0.8809	**0.1168**	0.4983	0.6730	0.3873	0.0177	0.0638	0.0257	0.9738
Minions	0.2925	0.9671	0.2367	0.4716	0.4864	0.4989	0.2607	0.0058	0.0547	0.9598
XGBBaseline	0.2848	0.9660	0.2109	0.4689	0.5400	0.4992	0.2607	0.0055	0.0686	0.9636
dragon_bra	0.9999	0.9443	0.1973	0.4983	0.5576	0.4990	0.2674	0.0021	0.0775	0.9587
miracle-flex	0.2538	0.8947	0.2367	0.4983	0.8910	0.4992	0.0548	0.0111	0.4043	0.9738
automl	0.9999	**0.7316**	0.1684	0.4540	0.8910	0.4992	0.0506	0.0074	0.4043	0.9738
shivr	0.9955	0.9760	0.2352	0.4258	0.8340	0.4138	0.8135	0.0687	0.4043	0.8906
tak	0.9999	0.9771	0.1897	0.4442	0.7430	0.4460	1.3154	0.9990	0.2326	0.9715
LinearBaseline	0.9999	0.9771	0.1897	0.4442	0.7430	0.4460	1.3154	0.9990	0.2326	0.9738
PEAR	0.9999	0.9771	0.2276	0.4983	0.8910	0.4992	1.3154	0.9990	0.4043	0.9738
arvindsrinivasan	0.9999	0.9771	0.1942	0.4983	0.8910	0.4992	1.5607	0.9990	0.4043	0.9738
chengnanli	0.9999	0.9771	0.1942	0.4983	0.8910	0.4992	1.5607	0.9990	0.4043	0.9738
Hannover AutoML Lab	0.9999	0.9771	0.2367	0.4983	0.8910	0.4992	1.3154	0.9990	0.4043	0.9738
Slayers	0.9999	0.9771	0.2367	0.4983	0.8910	0.4992	1.3154	0.9990	0.4043	0.9738
fmohr	0.9999	0.9771	0.2367	0.4983	0.8910	0.4992	1.3154	0.9990	0.4043	0.9738
Eric-Choi	0.9999	0.9771	0.2367	0.4983	0.8910	0.4992	1.3154	0.9990	0.4043	0.9738
neptune	0.9999	0.9771	0.2367	0.4983	0.8910	0.4992	1.3154	0.9990	0.4043	0.9738
ResoNet	0.9999	0.9771	0.2367	0.4983	0.8910	0.4992	1.3154	0.9990	0.4043	0.9738
MaybeWinningTeam	0.9999	0.9771	0.2367	0.4983	0.8910	0.4992	1.3154	0.9990	0.4043	0.9738
kgwilson2	0.9999	0.9771	0.2367	0.4983	0.8910	0.4992	1.3154	0.9990	0.4043	0.9738
Paris-Saclay	0.9999	0.9771	0.2367	0.4983	0.8910	0.4992	1.3154	0.9990	0.4043	0.9738

Team	Shallow Water	Satellite	Spherical Tiny ImageNet	JSB Chorales	Human Gait	Spoken MNIST	Stock	MYO	Year	HPA
TrueFit	0.0697	0.3764	**0.9798**	0.1228	**1e-7***	0.0067	0.0325	0.0279	0.0042	0.0219
TEG-AutoML	0.0474	0.3003	0.9903	0.1110	**1e-7***	0.0067	0.0822	0.1331	0.0033	**0.0190**
Freiburg-AutoML-Lab	**0.0004**	**0.2256**	0.9896	0.1017	2.281e-6	0.1200	0.0412	0.0400	0.0031	0.3883
euxhenh	0.0964	0.4059	0.9965	0.1273	1.171e-5	0.0100	**0.0208**	0.0226	0.0034	0.0302
tak	0.0408	0.2423	0.9895	0.4735	1.087e-7	0.0333	0.0490	0.3742	0.0033	0.2198
automl	0.0461	0.2378	0.9950	0.5013	9.417e-7	**0.0033**	0.0519	0.3742	0.0034	0.2227
42	0.0343	0.2460	0.9836	0.5160	7.244e-7	0.0100	0.0660	0.3742	0.0033	0.2091
Paris-Saclay	0.0119	0.2334	0.9949	**0.0973**	0.9192	0.0433	0.0620	**0.0082**	**0.0030**	0.1793
dragon_bra	0.0006	0.3330	0.9945	3.8154	0.0068	0.1167	0.0597	0.0461	0.0032	0.6127
XGBBaseline	**0.0004**	0.6072	0.9832	4.1342	0.2557	0.1333	0.0832	0.0362	0.0033	0.3871
miracle-flex	0.0293	0.5721	0.9995	0.1134	0.0010	0.4333	0.0570	0.3742	0.0710	0.2059
LinearBaseline	6.2550	0.4844	0.9927	0.2899	0.4331	0.1700	0.7873	0.3742	0.0059	0.6127
Hannover AutoML Lab	6.2550	0.6072	0.9943	4.1342	0.9192	0.4333	0.1292	0.3742	0.0710	0.6127
Eric-Choi	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
arvindsrinivasan	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
MaybeWinningTeam	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
kgwilson2	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
Minions	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
chengnanli	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
neptune	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
ResoNet	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
PEAR	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
shivr	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
Slayers	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127

*Post-processed to avoid scores of 0.0, which are not supported by AUP.

Table 9.4: Test leaderboard with additional post-competition baseline comparisons. Bolded scores indicate the best competition submission for each task. Bold underlined scores indicate post-competition baselines that tied or outperformed the best competition submission.

Team	Shallow Water	Satellite	Spherical Tiny ImageNet	JSB Chorales	Human Gait	Spoken MNIST	Stock	MYO	Year	HPA
TrueFit	0.0697	0.3764	**0.9798**	0.1228	**1e-7***	0.0067	0.0325	0.0279	0.0042	0.0219
DASHbaseline	0.0367	0.2689	0.9943	0.1198	**1e-7***	0.0800	0.0485	0.0146	0.0033	**0.0185**
AutoGluonBaseline	**0.0004**	0.2445	0.9949	4.1342	**1e-7***	0.0133	0.0832	**0.0074**	0.0033	0.3871
TEG-AutoML	0.0474	0.3003	0.9903	0.1110	**1e-7***	0.0067	0.0822	0.1331	0.0033	**0.0190**
Freiburg-AutoML-Lab	**0.0004**	**0.2256**	0.9896	0.1017	2.281e-6	0.1200	0.0412	0.0400	0.0031	0.3883
euxhenh	0.0964	0.4059	0.9965	0.1273	1.171e-5	0.0100	**0.0208**	0.0226	0.0034	0.0302
tak	0.0408	0.2423	0.9895	0.4735	1.087e-7	0.0333	0.0490	0.3742	0.0033	0.2198
automl	0.0461	0.2378	0.9950	0.5013	9.417e-7	**0.0033**	0.0519	0.3742	0.0034	0.2227
42	0.0343	0.2460	0.9836	0.5160	7.244e-7	0.0100	0.0660	0.3742	0.0033	0.2091
Paris-Saclay	0.0119	0.2334	0.9949	**0.0973**	0.9192	0.0433	0.0620	**0.0082**	**0.0030**	0.1793
dragon_bra	0.0006	0.3330	0.9945	3.8154	0.0068	0.1167	0.0597	0.0461	0.0032	0.6127
XGBBaseline	**0.0004**	0.6072	0.9832	4.1342	0.2557	0.1333	0.0832	0.0362	0.0033	0.3871
miracle-flex	0.0293	0.5721	0.9995	0.1134	0.0010	0.4333	0.0570	0.3742	0.0710	0.2059
LinearBaseline	6.2550	0.4844	0.9927	0.2899	0.4331	0.1700	0.7873	0.3742	0.0059	0.6127
Hannover AutoML Lab	6.2550	0.6072	0.9943	4.1342	0.9192	0.4333	0.1292	0.3742	0.0710	0.6127
Eric-Choi	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
arvindsrinivasan	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
MaybeWinningTeam	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
kgwilson2	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
Minions	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
chengnanli	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
neptune	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
ResoNet	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
PEAR	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
shivr	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127
Slayers	6.2550	0.6072	0.9995	4.1342	0.9192	0.4333	0.7873	0.3742	0.0710	0.6127

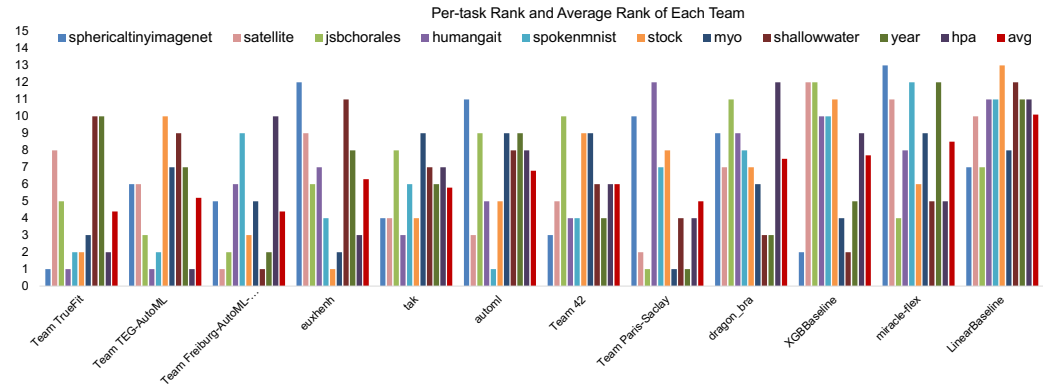
*Post-processed to avoid scores of 0.0, which are not supported by AUP.

Figure 9.4(b) shows the compute time distribution of the top-10 teams for each task. Both inter-task and intra-task variations were observed. Some teams, such as Team Paris-Saclay and miracle-flex, tended to use all of their compute time budget for each task to exhaustively search for better solutions. The NAS-based solution from Team TEG-AutoML also takes significantly longer time on average compared to most teams. The Satellite task was the most time-consuming, with an average compute time of 7 hours, followed by ShallowWater at 5 hours. The SpokenMINST task is the least time-consuming with an average compute time of 1 hour.

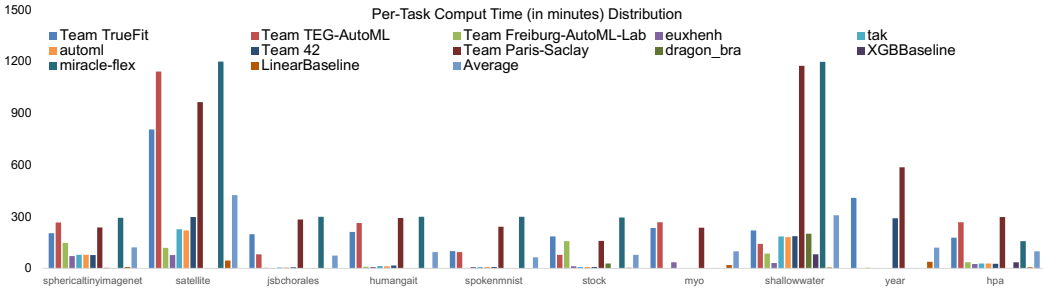
9.D Test Phase Migration

Due to technical issues with CodaLab, participants were unable to make submissions on the final day of the competition. To address this, participants were asked to email their final

Figure 9.4: Per-task analyses.



(a) Per-task ranks and average rank of each team that outperforms the linear baseline.



(b) Compute time distribution of each team on test tasks.

submissions according to the following rules:

- no major changes to the method were allowed,
- no new files can be added,
- no new functions can be implemented,
- modifications or deletions of the existing code should be within 20 lines of code, and
- updating the submission was allowed only once.

Additionally, we ensured that no hints of the test tasks were given. The submissions were checked on the development tasks before being promoted to the test phase. This period also served as a debugging phase, during which minor code issues were resolved. Small edits of up to 20 lines of code were allowed for issues such as file path discrepancies, numerical errors, or environment problems. The organizers were able to fix all minor issues in the final submissions within a day. For example, a function name typo was found for Team

TrueFit in their submitted “model.py,” and a floating-point stability issue was solved for Team TEG in their submitted “dataloaders.py” within 3 lines of code change.

Chapter 10

NAS-Bench-360: Benchmarking Neural Architecture Search on Diverse Tasks

This chapter is based on joint work with Renbo Tu, Mikhail Khodak, Junhong Shen, Frederic Sala, and Ameet Talwalkar, published at NeurIPS 2022 (Datasets and Benchmarks). The first two authors contributed equally.

Neural architecture search (NAS) aims to automate the design of deep neural networks, ensuring performance on par with hand-crafted architectures while reducing human labor devoted to tedious architecture tuning [Elsken et al., 2019]. With the growing number of application areas of ML, and thus of use-cases for automating it, NAS has experienced an intense amount of study in well-established machine learning domains, with significant progress in search space design [Zoph et al., 2018, Liu et al., 2019c, Cai et al., 2019], search efficiency [Pham et al., 2018], and search algorithms [Xu et al., 2020, Li et al., 2021a, White et al., 2021]. Notably, the field has largely been dominated by methods designed for and evaluated on benchmarks in computer vision [Liu et al., 2019c, Ying et al., 2019, Dong and Yang, 2020], yet the use of NAS techniques may be especially impactful in under-explored or under-resourced domains where less is known about useful architecture design patterns. There have been a few recent efforts to diversify these benchmarks to settings such as vision-based transfer learning [Duan et al., 2021a] and speech and language processing [Mehrotra et al., 2021, Klyuchnikov et al., 2020]; however, evaluating NAS methods on such well-studied tasks using traditional CNN search spaces does not give a good indication of their utility on more far-afield applications, which have often necessitated the design of

custom neural operations [Cohen et al., 2018a, Li et al., 2021c].

We make progress towards studying NAS on more diverse tasks by introducing a suite of benchmark datasets drawn from various data domains that we collectively call NAS-Bench-360. This benchmark consists of an organized setup of ten suitable datasets that represent diverse application domains, dataset sizes, problem dimensionalities, and learning objectives. We also include a standard image classification task as a baseline point of comparison, as many new methods continue to be designed for that setting. Note that the core component of NAS-Bench-360 is *not* a typical NAS benchmark, which often involves precomputing all architectures in some fixed search space. In contrast, our contribution is explicitly intended to be agnostic of the search space being used, as different search spaces may work well for different tasks. Thus NAS-Bench-360 is a task-oriented NAS benchmark with the intended use-case of evaluating NAS method and search space pairs on a wide variety of domains. However, to aid research, three of our tasks—for two of which we contribute the precompute—do come with trained architectures from the NAS-Bench-201 search space [Dong and Yang, 2020].

Experimentally, we demonstrate the usefulness of NAS-Bench-360 by performing a set of analyses evaluating whether the success of NAS in computer vision is indicative of strong performance on the much broader set of problems to which NAS can be applied. Specifically, we report performance comparisons between NAS methods, investigate the validity of existing NAS hypotheses made solely on computer vision tasks, and extend an existing analysis of zero-cost proxies already-enabled by our benchmark [White et al., 2022]. From these analyses, we arrive at the following conclusions:

- Resource-constrained practitioners may be better of using a fixed CNN rather than NAS (Figure 10.1).
- NAS-Bench-201 analyses on computer vision tasks do not generalize to diverse tasks.
- Zero-cost proxies perform inconsistently on diverse tasks, corroborating earlier findings [White et al., 2022].

We have released all datasets, experiment code, precomputed models, seeds, and environments used in our experiments.[1] Releasing our code, random seeds, and environments in the form of Docker containers assures reproducibility of all experimental results presented in this work and encourages the same level of reproducibility for future research performed

[1] `https://github.com/rtu715/NAS-Bench-360`

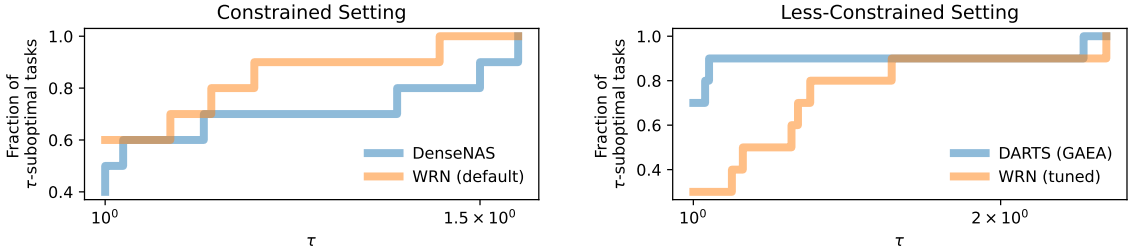


Figure 10.1: Performance profiles on NAS-Bench-360 comparing NAS methods (blue) to a fixed CNN (orange), specifically a Wide ResNet (WRN) [Zagoruyko and Komodakis, 2016b]. Resource-constrained practitioners might be better off not using NAS (left), while less constrained practitioners can still benefit (right). The y-axis is the fraction of tasks on which error is within a factor τ of the optimal method, i.e. higher is better.

using NAS-Bench-360.

10.1 Related Work

Benchmarks have been critical to the development of NAS in recent years. This includes standard evaluation datasets and protocols, of which the most popular are the CIFAR-10 and ImageNet routines used by DARTS [Liu et al., 2019c]. Another important type of benchmark has been tabular benchmarks such as NAS-Bench-101 [Ying et al., 2019], NAS-Bench-201 [Dong and Yang, 2020], NAS-Bench-1Shot1 [Zela et al., 2020], and TransNAS-Bench-101 [Duan et al., 2021b]; these benchmarks exhaustively evaluate all architectures in their search spaces, which is made computationally feasible by defining simple searched cells. Consequently, they are less expressive than the DARTS cell [Liu et al., 2019c], often regarded as the most powerful search space in the cell-based regime. Notably, the full NAS-Bench-360 benchmark is *not* intended to be a tabular benchmark, i.e. we do *not* evaluate every architecture from a fixed search space on all ten of our tasks; instead, the focus is on the organization of a suite of tasks for assessing both NAS algorithms and search spaces, which would necessarily be restricted by fixing a search space for a tabular benchmark. Pre-computing on an expansive search space such as DARTS, with 10^{18} possible architectures, is computationally intractable. Architectures found on lesser search spaces are most likely suboptimal: a vanilla Wide ResNet (WRN) outperforms all networks in the NAS-Bench-201 search space on CIFAR-100. Nonetheless, we find that including precompute results for all of NAS-Bench-201 on two of our tasks is useful in evaluating various claims in the NAS literature centered on computer

vision tasks.

While NAS methods and benchmarks have generally been focused on computer vision, recent work such as AutoML-Zero [Real et al., 2020b] and XD-operations [Roberts et al., 2021a] has started moving towards a more generically applicable set of tools for AutoML. However, even more recent benchmarks that do go beyond the most popular vision datasets have continued to focus on well-studied tasks, including vision-based transfer learning [Duan et al., 2021a], speech recognition [Mehrotra et al., 2021], and natural language processing [Klyuchnikov et al., 2020]. We aim to go beyond such areas to evaluate the potential of NAS to automate the application of ML in truly under-explored domains. One analogous work to ours in the field of meta-learning is the Meta-Dataset benchmark of few-shot tasks [Triantafillou et al., 2020], which similarly aimed to establish a wide-ranging set of evaluations for that field. For our inclusion of diverse tasks, we title our benchmark NAS-Bench-360 to resemble the idea of a 360-degree camera that covers all possible directions.

10.2 NAS-Bench-360: A Suite of Diverse and Practical Tasks

In this section, we introduce the NAS setting targeted by our benchmark, our motivation for organizing a new set of diverse tasks as a NAS evaluation suite, and our task-selection methodology. We report evaluations of specific algorithms on this new benchmark in the next section.

10.2.1 Neural Architecture Search: Problem Formulation and Baselines

For completeness and clarity, we first formally discuss the architecture search problem itself, starting with the extended hypothesis class formulation [Li et al., 2021a]. Here the goal is to use a dataset of points $x \in \mathcal{X}$ to find parameters $w \in \mathcal{W}$ and $a \in \mathcal{A}$ of a parameterized function $f_{w,a} : \mathcal{X} \mapsto \mathbb{R}_{\geqslant 0}$ that minimize the expectation $\mathbb{E}_{x \sim \mathcal{D}} f_{w,a}(x)$ for some test distribution $\mathcal{D}$ over $\mathcal{X}$; here $\mathcal{X}$ is the input space, $\mathcal{W}$ is the space of model weights, and $\mathcal{A}$ is the set of architectures. For generality, we do not require the training points to be drawn from $\mathcal{D}$ to

allow for domain adaptation, as is the case for one of our tasks, and we do not require the loss to be supervised. Note also that the goal here does not depend on computational or memory efficiency, which we do not focus on in our evaluations; our restriction is only that the entire pipeline can be run on an NVIDIA V100 GPU.

Notably, this formulation makes no distinction between the model weights w and architectures a, treating both as parameters of a larger model. Indeed, the goal of NAS may be seen as similar to model design, except now we include the design of an (often discrete) *architecture space* $\mathcal{A}$ such that it is easy to find an architecture $a \in \mathcal{A}$ and model weights $w \in \mathcal{W}$ whose test loss $\mathbb{E}_{\mathcal{D}} f_{w,a}$ is low using a search algorithm. This can be done in a one-shot manner—simultaneously optimizing a and w—or using the standard approach of first finding an architecture a and then keeping it fixed while training model weights w using a pre-specified algorithm such as stochastic gradient descent (SGD). This formulation divides NAS algorithms into two camps: one-shot, weight-sharing methods and non-weight-sharing ones such as random search, which operate by repeatedly sampling architectures and evaluating them. The formulation also includes non-NAS methods by allowing the architecture search space to be a singleton. When the sole architecture is a standard and common network such as WRN [Zagoruyko and Komodakis, 2016b], this yields a natural baseline with an algorithm searching for training hyperparameters, not architectures. For our empirical investigation, we compare the performance of state-of-the-art NAS approaches against that of the three baselines: WRN, PerceiverIO [Jaegle et al., 2022], and XGBoost [Chen and Guestrin, 2016b].

10.2.2 Task Selection: Motivation and Methodology

Curating a diverse, practical set of tasks for the study of NAS is our primary motivation behind this work. We observe that past NAS benchmarks focused on creating larger search spaces and more sophisticated search methods for neural networks. However, the utility of these search spaces and methods are only evaluated on canonical computer vision datasets. On a broader range of problems, whether these new methods can improve upon simple baselines remains an open question. This calls for the introduction of new datasets lest NAS research overfits to the biases of CIFAR-10 and ImageNet. By identifying these possible biases, future directions in NAS research can be better primed to suit the needs of practitioners and to increase the deployment of NAS.

Table 10.1: Task metadata for NAS-Bench-360. Metrics are standardized such that lower is better.

Task name	Size	Dim.	Type	Learning objective	Metric	New to NAS?
CIFAR-100 [Krizhevksy, 2009]	60K	2D	Point	Classify natural images into 100 classes	0-1 error	no, widely used
Spherical [Cohen et al., 2018a]	60K	2D	Point	Classify spherically projected images into 100 classes	0-1 error	✓
NinaPro [Atzori et al., 2012a]	3956	2D	Point	Classify sEMG signals into 18 classes of hand gestures	0-1 error	✓
FSD50K [Fonseca et al., 2017b]	51K	2D	Point (multi-label)	Classify sound events in log-mel spectrograms with 200 labels	1—mAP	✓
Darcy Flow [Li et al., 2021c]	1100	2D	Dense	Predict the final state of a fluid from its initial conditions	relative ℓ_2	no, used in [Roberts et al., 2021a]
PSICOV [Adhikari, 2020c]	3606	2D	Dense	Predict pairwise distances between residuals from pairwise sequence features	MAE_8	no, used in [Roberts et al., 2021a]
Cosmic [Zhang and Bloom, 2020b]	5250	2D	Dense	Predict probabilistic maps to identify cosmic rays in telescope images	1 - AUROC	✓
ECG [Clifford et al., 2017c]	330K	1D	Point	Detecting atrial cardiac disease from ECG recordings	1—F1	✓
Satellite [Petitjean et al., 2012]	1M	1D	Point	Classify satellite image pixel time series into 24 land cover types	0-1 error	✓
DeepSEA [Consortium et al., 2004]	250K	1D	Point (multi-label)	Predicting chromatin and binding states of DNA sequences	1—AUROC	no, used in [Zhang et al., 2021b,c]

Summarized in Table 10.1, NAS-Bench-360 consists of problems that are conducive to processing by convolutional neural networks, which includes a trove of applications associated with spatial and temporal data, spanning single and multiple dimensions. Most current NAS methods are not implemented to search for other types of architectures to process tabular data and graph data. Therefore, we have set this scope for our investigation. During the selection of tasks, diversity is our primary consideration. We define the following axes of diversity to govern our task-filtering process: the first is problem dimensionality, including both 2D with matrix inputs and 1D with sequence inputs; the second is dataset size, for which our selection spans the scale from 1,000 to 1,000,000; the third is problem type , divisible into tasks requiring a singular prediction (point prediction) and multiple predictions (dense prediction); fourth and finally, diversity is achieved through selecting tasks from various learning objectives from applications of deep learning, where introducing NAS could improve upon the performance of handcrafted neural networks.

In lieu of providing raw data, we perform data pre-processing locally and store the processed data on a public Amazon Web Services S3 data bucket with download links available on our website. Our data treatment largely follows the procedure defined by the researchers who provided them. This enhances reproducibility by ensuring the uniformity of input data for different pipelines. Additional information about the datasets, pre-processing, and augmentation steps are described in the Appendix.

10.3 Experimental design

Having detailed our construction of NAS-Bench-360, in this section we will establish the experimental setup for our analyses in the following section, which demonstrates the usefulness of NAS-Bench-360 for evaluating NAS methods on diverse tasks. We first specify the NAS methods and baselines we compare, followed by the details of the experimental setup and intended use of the benchmark. Finally, we provide details of the precomputed NAS-Bench-201 search space for two representative diverse tasks from NAS-Bench-360: NinaPro and Darcy Flow.

10.3.1 Baselines and Search Procedures

Our initial experiments follow two practitioners with different resource settings: one with enough compute to tune a WRN (less-constrained) and another who can only train it once with the default hyperparameters (constrained). Given these two scenarios, we compare against NAS methods that each practitioner would be able to run. In both cases, we focus on two well-known search paradigms: cell-based NAS (using DARTS [Liu et al., 2019c]) and macro NAS (using DenseNAS [Fang et al., 2020]). We further compare these approaches to two customized NAS methods: Auto-DeepLab [Liu et al., 2019b] for 2D dense prediction and AMBER [Zhang et al., 2021c] for 1D prediction, as well as general-purpose baselines: Perceiver IO [Jaegle et al., 2022] and XGBoost [Chen and Guestrin, 2016b]. Additional details are provided in the Appendix.

10.3.2 Experimental Setup

Below we discuss the main reporting details of our empirical evaluation.

Table 10.2: Performance of NAS and baselines across NAS-Bench-360. Methods are divided into efficient methods (e.g. DenseNAS and fixed WRN) that take 1-10 GPU-hours, more expensive methods (e.g. DARTS and tuned WRN) that take 10-100+ GPU-hours, and specialized methods (Auto-DL and AMBER). All results are averages of three random seeds, and lower is better for all metrics. The best performing method is shown in bold and the best non-expert-designed method is underlined.

Search space	Search algorithm	CIFAR-100	Spherical	Darcy Flow	PSICOV	Cosmic
WRN	default	23.35±0.05	85.77±0.71	0.073±0.001	3.84±0.05	0.245±0.02
DenseNAS	random	25.49±0.41	71.23±1.65	0.071±0.006	3.70±0.06	0.309±0.04
DenseNAS	original	25.98±0.38	72.99±0.95	0.100±0.010	3.84±0.15	0.383±0.04
Perceiver IO	default	70.04±0.44	82.57±0.19	0.240±0.010	8.06±0.06	0.485±0.01
XGBoost	default	84.83±4.15	96.92±0.02	0.085±0.000	n/a*	0.232±0.00
WRN	ASHA	23.39±0.01	75.46±0.40	0.066±0.000	3.84±0.05	0.251±0.02
DARTS	GAEA	24.02±1.92	**48.23±2.87**	0.026±0.001	**2.94±0.13**	0.229±0.04
Auto-DL	DARTS	n/a	n/a	0.049±0.005	6.73±0.73	0.495±0.00
Expert	default	**19.39±0.20**	67.41±0.76	**0.008±0.001**	3.35±0.14	**0.127±0.01**

Search space	Search algorithm	NinaPro	FSD50K	ECG	Satellite	DeepSEA
WRN	default	**6.78±0.26**	0.92±0.001	0.43±0.01	15.49±0.03	0.40±0.001
DenseNAS	random	8.45±0.56	**0.60±0.001**	0.42±0.01	13.91±0.13	0.40±0.001
DenseNAS	original	10.17±1.31	0.64±0.002	0.40±0.01	13.81±0.69	0.40±0.001
Perceiver IO	default	22.22±1.80	0.72±0.002	0.66±0.01	15.93±0.08	0.38±0.004
XGBoost	default	21.90±0.70	0.98±0.002	0.56±0.00	36.36±0.02	0.50±0.000
WRN	ASHA	7.34±0.76	0.91±0.030	0.43±0.01	15.84±0.52	0.41±0.002
DARTS	GAEA	17.67±1.39	0.94±0.020	0.34±0.01	**12.51±0.24**	0.36±0.020
AMBER	ENAS	n/a	n/a	0.33±0.02	12.97±0.07	0.32±0.010
Expert	default	8.73±0.90	0.62±0.004	**0.28±0.00**	19.80±0.00	**0.30±0.024**

* did not fit on a single V100 GPU.

Table 10.3: Median rank and performance improvement over WRN across NAS-Bench-360.

Search space Search algorithm	WRN default	DenseNAS original	DenseNAS random	WRN ASHA	DARTS GAEA	Auto-DL DARTS	AMBER ENAS
Median rank	4.0	4.0	4.0	3.5	1.5	6.0†	1.0†
% better than WRN*	0.0%	2.53%	0.0%	0.1%	14.6%	-75.3%†	20.0%†

* relative improvement over the default (untuned) WRN baseline
† metric computed only on the subset of three tasks on which the method was evaluated

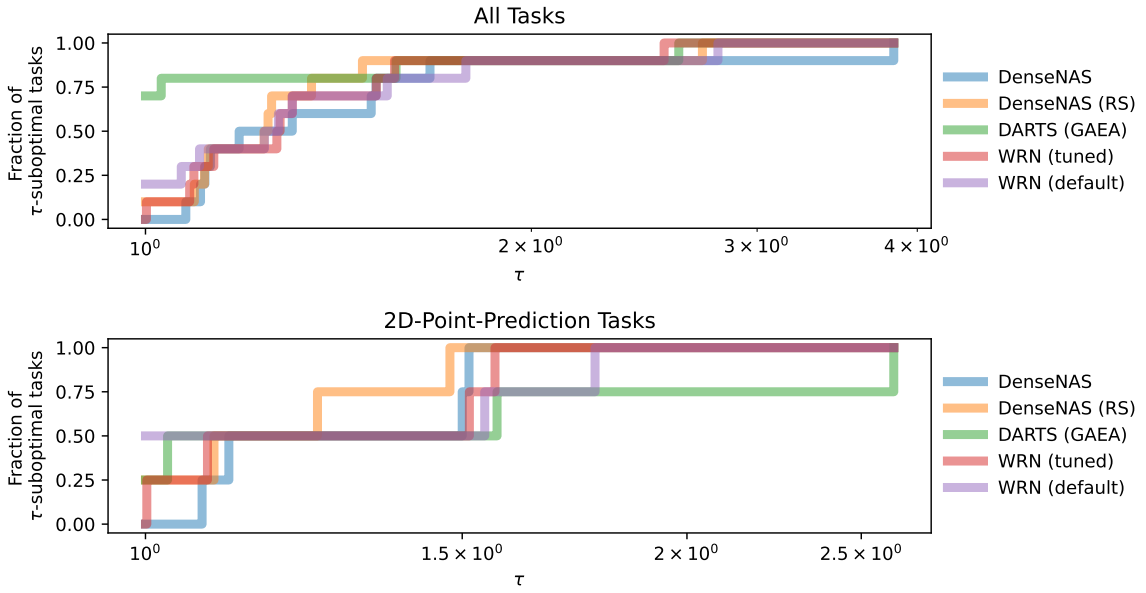


Figure 10.2: In our investigation of modern methods on NAS-Bench-360, we find that methods like GAEA DARTS can be strong in aggregate, as shown in the performance profiles on all tasks (top), but worse on salient subsets such as 2D point tasks (bottom). The y-axis is the fraction of tasks on which error is within a factor τ of the optimal method, i.e. higher is better.

- Hyperparameter tuning: As detailed in the Appendix, we use the same hyperparameter ranges across all tasks to tune WRN. We use ASHA [Li et al., 2018b] to search over these hyperparameters and give it a budget on each task that matches the total search and retraining budget of DARTS (GAEA).
- Aggregation metrics: To aggregate results across tasks, we use the median rank of each method and its performance improvement over WRN for direct comparison via a singe number, as demonstrated in Table 10.3. However, since these metrics can be sensitive to small differences in performance, we also employ performance profiles [Dolan and Moré, 2002a] to mitigate that effect while still accounting for outliers. As described in Figure 10.1, these curves denote for each τ the fraction of tasks on which a method is no worse than a τ-factor from the optimal. Concretely, we plot $\rho_s(\tau) = \frac{1}{|\mathcal{P}|}\left|\left\{p \in \mathcal{P} : \frac{\text{error}_{p,s}}{\min_{s \in \mathcal{S}} \text{error}_{p,s}} \leqslant \tau\right\}\right|$ given some method $s \in \mathcal{S}$ on tasks $\mathcal{P}$.

- Software and hardware: We adopt the free, open-source software *Determined*[2] for experiment management, hyperparameter tuning, and cloud deployment. All experiments are performed on a single p3.2xlarge instance with a 16GB NVIDIA V100 GPU. While evaluation on NAS-Bench-360 indeed assumes access to at least a single V100 GPU, we reiterate that we provide the precomputed NAS-Bench-201 search space for two of our tasks in cases where GPU access is limited. Costs in GPU-hours are in the appendix.

10.3.3 Precomputing NAS-Bench-201 on NinaPro and Darcy Flow

The intended goal of NAS-Bench-360 is to evaluate the performance of *NAS search method and search space pairs* on diverse tasks, which precludes the precomputation of all architectures in general due to the lack of a single fixed search space. A complete lack of precomputed architectures would be perhaps limiting for many NAS researchers, who rely on precomputed NAS benchmarks when developing new search methods. In an effort to address this potential limitation, we precompute all architectures in the NAS-Bench-201 [Dong and Yang, 2020] search space on two representative tasks in NAS-Bench-360: NinaPro and Darcy Flow. We follow the same experimental procedure as in the original NAS-Bench-201 benchmark [Dong and Yang, 2020] to generate the precompute results, except where they vary the number of models trained for each architecture between one and three, we fix the number of trials per architecture to one. Note that NAS-Bench-201 already includes precompute for CIFAR-100, a dataset we include in NAS-Bench-360.

10.4 Analysis

We conclude our presentation of NAS-Bench-360 with three sets of analyses. The first, a performance analysis of NAS methods and fixed baselines across diverse tasks, reveals new insights about the capabilities and robustness of current NAS methods and demonstrates how our benchmark can enable critical next steps in NAS research. In our second analysis, we evaluate claims from the NAS literature originally made using computer vision tasks, and show that they do not generalize to diverse tasks; this demonstrates how NAS research can

[2] `https://github.com/determined-ai/determined`

benefit from our contribution in the future. Finally, we extend an existing analysis of zero-cost proxy methods on diverse tasks that already uses NAS-Bench-360 [White et al., 2022].

10.4.1 Performance across diverse tasks using NAS-Bench-360

As discussed in Section 10.3.2, we start by considering two practitioners faced with a choice of spending their limited compute on a (possibly tuned) fixed-architecture CNN or trying to find a better architecture using NAS. With this study, we investigate whether modern NAS methods perform well beyond the tasks for which they were designed.

1. A surface-level analysis suggests that under light resource constraints, modern NAS in the form of DARTS (GAEA) is quite robust to a wide variety of tasks: Table 10.3 shows it is the highest-ranked domain-independent method and attains the most significant improvement over the fixed WRN baseline. The performance profile in Figure 10.2 (left) also seems favorable, although it is overtaken by tuned WRN at a higher τ-suboptimality. However, a closer look at 2D point tasks in Figure 10.2 (right) reveals that DARTS is quite poor there, despite its design domain being image classification; in particular, it performs very poorly on NinaPro and FSD50K. Furthermore, on tasks where it performs well, it can still lag behind expert architectures; for example, on Darcy Flow, networks that use FNO [Li et al., 2021c] or XD-operations [Roberts et al., 2021a] do much better. Overall, our results suggest that this practitioner can apply NAS and expect to see some improvement, but also risks catastrophically poor performance (e.g. FSD50K) or not getting truly state-of-the-art results (e.g. Darcy Flow).
2. Under stronger budget constraints, our experiments strongly suggest that a practitioner should simply apply the default WRN to their problem rather than undergo the additional complexity of using DenseNAS, as the latter attains little-to-no improvement over the former in Table 10.3 and has a usually-worse performance profiles Figure 10.2. On the other hand, DenseNAS performs well on FSD50K—it outperforms all methods even while DARTS (GAEA) fails.

These first experiments suggest that the modern NAS methods are not always robust to diverse tasks, especially under resource-constrained settings. We believe that NAS-Bench-360's main roles as a future benchmark include developing an understanding of the multi-domain performance of existing approaches and guiding research into better NAS methods. While the latter is beyond the scope of this chapter, our additional experiments

demonstrate how NAS-Bench-360 facilitates the former.

Notably, several of our results address the question of the relative importance of search space vs. search algorithm. For example, Table 10.3 shows that on DenseNAS, random search is nearly identical to the more sophisticated weight-sharing scheme of the original paper; the two algorithms' performance profiles are also difficult to distinguish in Figure 10.2. Furthermore, AMBER—a 1D NAS method whose search space includes larger-kernel convolutions for handling such tasks—does better than GAEA even though it uses an older search algorithm (ENAS). These both suggest that search space design, including the use of a wider variety of operations, may be at least as crucial for success as the search algorithm. This point is reinforced by example tasks such as Darcy Flow, where architectures with more exotic operations substantially outperform our best results, as discussed earlier.

NAS-Bench-360 also reveals failure points of several methods, not just of general ones that usually perform quite well such as DARTS (GAEA) but also the objective-specific approach Auto-DL, which despite being designed for dense prediction tasks does poorly on all those considered here. Understanding when and why these performance drops happen is critical to developing a more robust NAS that is useful not just on average but in more challenging settings.

10.4.2 Do past NAS-Bench-201 analyses generalize to NAS-Bench-360?

Existing NAS-benches have been widely used for analyses such as (1) comparing performances of different architectures across tasks, (2) quickly evaluating search methods, and (3) investigating design choices that impact performance. In this section we show via the NAS-Bench-201 search space that the conclusions of past analyses cannot be assumed to hold on tasks beyond computer vision.

10.4.2.1 Architecture transferability

We start by using the precomputed results outlined in Section 10.3.3 to show in Figure 10.3 the rank of each architecture across different datasets, indexed on the x-axis by its rank on CIFAR-100. This reveals that while architecture rankings are highly correlated on image classification datasets—as pointed out by the authors of the original benchmark [Dong and Yang, 2020]—the rankings become uncorrelated when evaluated on a more diversified

set of tasks. Therefore, NAS evaluations should be done across domains to verify true generalizability, and NAS-Bench-360 is especially useful for this purpose.

10.4.2.2 Search algorithm performance

Using the precomptued evaluations on two new datasets, we evaluate all ten typical NAS algorithms originally studied on NAS-Bench-201 [Dong and Yang, 2020]. The results are shown in the Appendix. With similar wall clock time, the non-weight-sharing NAS algorithms that we evaluate: REINFORCE [Williams, 1992], random search (RS) [Bergstra and Bengio, 2012], regularized evolution (REA) [Real et al., 2019], BOHB [Falkner et al., 2018], and Hyperband [Li et al., 2018a] consistently perform well. Our results corroborate the strong performance of non-weight-sharing methods on this search space.

On the other hand, our experiments reveal some important differences for weight-sharing methods. In particular, unlike in past experiments on the NAS-Bench-201 search space, DARTS does not always yield a network of all skip-connection on Darcy Flow, despite this behavior on image classification and NinaPro. Instead, both first-order and second-order DARTS often pick convolution operations and sometimes achieve good performance, although still worse overall than the best non-weight-sharing methods. These results together with the ranking demonstrate that evaluating methods and search spaces on vision tasks alone does not give a full picture of their capabilities and limitations, a problem alleviated by NAS-Bench-360.

10.4.2.3 Operation redundancy

Our final analysis using the NAS-Bench-201 search space is to investigate the conclusions of a more recent study on the redundancy of operations [Wan et al., 2022]. We find that the operation redundancy phenomenon they outline is task-dependent and does not generalize to tasks beyond the three vision tasks—CIFAR-10, CIFAR-100, and ImageNet16-120—that they study. To conduct our study we follow their procedure to obtain "operation importance" distributions for each operation in the NAS-Bench-201 search space for NinaPro and Darcy Flow; additionally, we reproduce their results on CIFAR-10, CIFAR-100, and ImageNet16-120. *Operation importance* measures the incremental effect of each operation choice in the NAS-Bench-201 search space—1x1 convolutions (c1), 3x3 convolutions (c3), skip connections (skip), and 3x3 average pooling (ap3)—on performance [Wan et al., 2022]. The original analy-

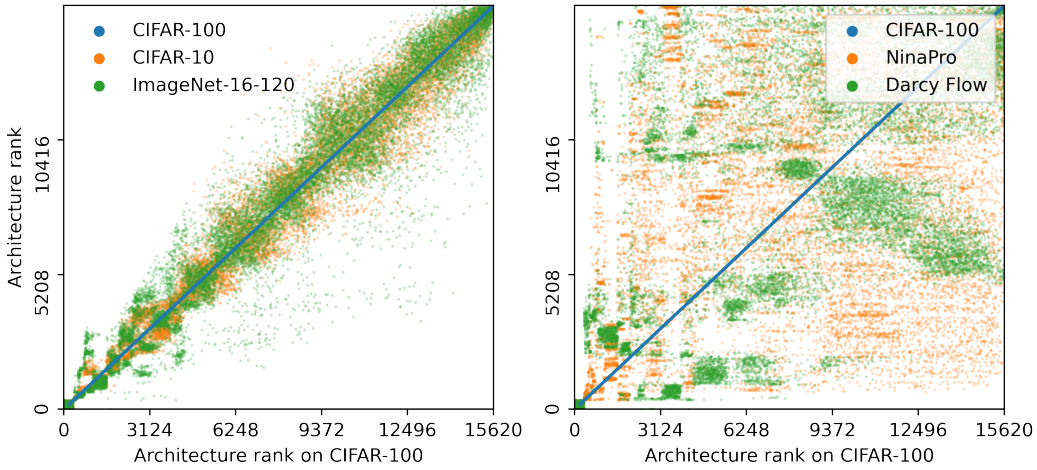


Figure 10.3: Architecture rankings between computer vision tasks correlate on NAS-Bench-201 [Dong and Yang, 2020] (left, sorted by performance on CIFAR-100) but are uncorrelated between CIFAR-100 and two NAS-Bench-360 tasks, NinaPro and Darcy Flow (right).

sis found that the operation importance distributions are roughly similar across the original NAS-Bench-201 computer vision datasets, which we confirm and show in Figure 10.4. However, we found that the operation importance distributions were drastically different for NinaPro and Darcy Flow, which we also show in Figure 10.4. Not only are their distributions different from those of the computer vision tasks in the original analysis, but the operation importance distribution for NinaPro differs significantly from that of Darcy Flow. This tells us that *different operations are more useful for different tasks*, and using NAS-Bench-360, we find that we cannot conclude that any of these operations are universally redundant or useful in a given search space across tasks. In other words, using NAS-Bench-360, we find that the original claim that "existing search spaces contain a high degree of redundancy" [Wan et al., 2022] does not hold when considering diverse tasks beyond computer vision.

10.4.3 Zero-cost proxies on diverse tasks

We conclude with an analysis of TE-NAS [Chen et al., 2021c], a zero-cost proxy inspired by neural tangent kernel (NTK) analysis, on four NAS-Bench-360 tasks. Zero-cost proxies [Mellor et al., 2021, Abdelfattah et al., 2021] are the subject of a recent direction in NAS research that aims to construct quick-to-evaluate measures of architecture performance without doing any training. Recently, [White et al., 2022] evaluated several zero-cost proxies on

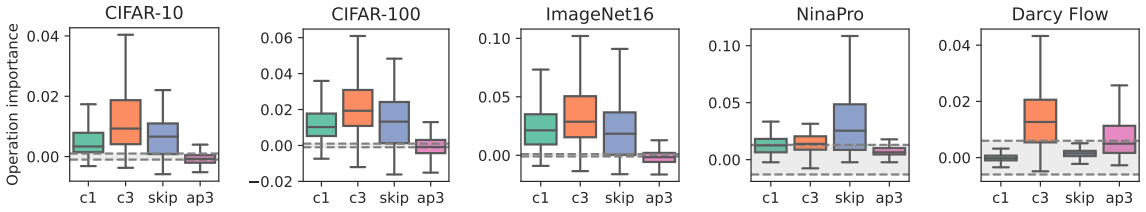


Figure 10.4: Different operations are important for different tasks. While prior work [Wan et al., 2022] shows that the operation importance distributions are stable across computer vision tasks—as shown by the high similarity of the three plots on the left—we find that they differ significantly for NinaPro and Darcy Flow.

Table 10.4: Performance comparison of TE-NAS and GAEA using the DARTS search space on CIFAR-100, Spherical, NinaPro, and Darcy Flow. Lower is better for all metrics.

	CIFAR-100	Spherical	NinaPro	Darcy Flow
TE-NAS	24.32	56.87	**9.71**	**0.012**
GAEA	**24.02**	**48.23**	17.67	0.026

tasks from NAS-Bench-360 (Spherical, NinaPro, and Darcy Flow), as well as on TransNAS-Bench-101 [Duan et al., 2021a]. One major weakness of zero-cost proxies that they point out is that zero-cost proxies are not much more computationally efficient than weight-sharing methods, as the total compute cost is still dominated by the evaluation of the searched architecture [White et al., 2022]. For example, this renders TE-NAS in the DARTS search space comparable to GAEA DARTS in terms of computational efficiency. The authors of [White et al., 2022] also point out that the performance of different zero-cost proxies vary considerably across diverse datasets, even subject to the same search space. Performance may be strong on some tasks, but weak on others.

To expand such study of zero-cost proxies, we look at one that [White et al., 2022] do not consider—TE-NAS—and evaluate its performance on the DARTS space using four NAS-Bench-360 tasks: CIFAR-100, Spherical, NinaPro, and Darcy Flow. The results of this evaluation are shown in Table 10.4. Unlike many other zero-cost-proxies [Mellor et al., 2021], the fact that TE-NAS is constructed from a domain-agnostic NTK analysis rather than experiments makes it a potential candidate for good performance on diverse tasks. However, Table 10.4 shows that performance does vary considerably across tasks, as observed for other proxies by [White et al., 2022]. In-particular, TE-NAS performs okay on NinaPro and

beats all methods in Table 10.2 on Darcy Flow—where its performance approaches that of the expert-designed FNO [Li et al., 2021c]—but does poorly on Spherical. This evaluation adds evidence to existing scientific findings already enabled by NAS-Bench-360 [White et al., 2022] and provides additional evidence for the need to evaluate all NAS methods, including zero-cost proxies, on diverse tasks.

10.5 Conclusion

NAS-Bench-360 is a new performance benchmark consisting of ten diverse tasks derived from various fields of research and practice. It is designed for reproducible research on an academic budget that will guide the development of NAS methods and other automated approaches towards more robust performance across different domains. In initial results, we have demonstrated both the need for such a benchmark and the utility of NAS-Bench-360 specifically for developing new search spaces and algorithms. We also provide precompute architectures from the NAS-Bench-201 search space on two of the ten tasks. While the precomputed architectures on these two tasks are useful for analysis on their own, adding more precomputed search spaces and tasks is an area of further improvement. We welcome researchers to use the NAS-Bench-360 tasks to develop new procedures for automating ML.

10.A Tabular Benchmark Results

The precomputed evaluation on the NinaPro and Darcy Flow tasks significantly reduce runtimes of NAS algorithms. We evaluate the 10 search methods from NAS-Bench-201 [Dong and Yang, 2020], which comprises 6 non-weight sharing methods and 4 weight-sharing methods. The time budget of non-weight-sharing algorithms are set to match that of DARTS-V1. The non-weight-sharing methods are averaged over 500 runs, and the weight-sharing methods are averaged over 3 runs.

10.B Dataset Descriptions

CIFAR-100: Standard Image Classification. As a starting point of comparison to existing benchmarks, we include the CIFAR-100 task [Krizhevksy, 2009], which contains

Table 10.5: Results of 12 NAS algorithms across three datasets of NAS-Bench-360. We report the mean and standard deviation. All metrics are errors such that lower is better. The best methods in the non-WS and WS groups are bolded.

Algorithm	NinaPro	Darcy Flow	CIFAR-100
REINFORCE [Williams, 1992]	**8.07±0.73**	0.0247±0.006	28.14±0.89
RS [Bergstra and Bengio, 2012]	8.09±0.71	0.0252±0.006	28.45±0.97
REA [Real et al., 2019]	8.15±0.85	0.0244±0.006	27.77±0.84
BOHB [Falkner et al., 2018]	8.17±0.57	0.0194±0.002	28.00±0.86
Hyperband [Li et al., 2018a]	8.16±0.57	**0.0191±0.002**	**27.66±0.67**
DARTS-V1 [Liu et al., 2019c]	22.06±2.00	0.178±0.107	38.74±4.43
DARTS-V2 [Liu et al., 2019c]	22.06±2.00	0.150±0.093	39.51±4.95
RSWS [Li and Talwalkar, 2019]	9.82±1.49	0.221±0.045	31.74±0.96
GDAS [Dong and Yang, 2019b]	17.61±6.39	0.180±0.103	31.87±2.50
SETN [Dong and Yang, 2019a]	14.56±7.30	0.253±0.000	30.64±1.72
ENAS [Pham et al., 2018]	11.56±1.12	0.253±0.000	29.33±0.62
GAEA (ERM) [Li et al., 2021a]	**7.69±0.19**	**0.026±0.003**	**26.73±0.18**

*We did not include the zero operation in GAEA(ERM), as in the original paper.

RGB images from natural settings to be classified into 100 fine-grained categories. CIFAR-100 is preferred over CIFAR-10 because it is more challenging and suffers less from over-fitting in previous research.

Spherical: Classifying Spherically Projected CIFAR-100 Images. To test NAS methods applied to natural-image-like data, we consider the task of classifying spherical projections of the CIFAR-100 images, which we call Spherical. In addition to scientific interest, spherical image data is also present in various applications, such as omnidirectional vision in robotics and weather modeling in meteorology, as sensors usually produce distorted image signals in real-life settings. To create Spherical CIFAR, we project the planar signals of the CIFAR images to the northern hemisphere and add a random rotation to produce spherical signals for each channel following the procedure specified in [Cohen et al., 2018a]. The resulting images are 60×60 pixels with RGB channels.

NinaPro: Classifying Electromyography Signals. NinaPro moves away from the image domain to classify hand gestures indicated by electromyography signals. For this, we use a subset of the NinaPro DB5 dataset [Atzori et al., 2012a] in which two Myo armbands

collect EMG signals from 10 test individuals who hold 18 different hand gestures to be classified. These armbands leverage data from muscle movement, which is collected using electrodes in the form of wave signals. Each wave signal is then sampled using a wavelength and frequency prescribed in [Côté-Allard et al., 2019] to produce 2D signals.

FSD50K: Labeling Sound Events. FSD50K [Fonseca et al., 2020] is derived from the larger Freesound dataset [Fonseca et al., 2017b] of Youtube videos with 51,000 clips totaling more than 100 hours of sound. These clips are manually labeled and equally distributed in 200 classes from the AudioSet ontology [Gemmeke et al., 2017]. Each clip could receive multiple labels. Unlike TIMIT [Garofolo, 1993], FSD50K does not focus exclusively on sounds of spoken language but includes sound events from physical sources and production mechanisms. The mean average precision (mAP) is used to evaluate classification results.

Darcy Flow: Solving Partial Differential Equations (PDEs). Our first regression task, Darcy Flow, focuses on learning a map from the initial conditions of a PDE to the solution at a later timestep. This application aims to replace traditional solvers with learned neural networks, which can output a result in a single forward pass. The input is a 2d grid specifying the initial conditions of a fluid, and the output is a 2d grid specifying the fluid state at a later time, with the ground truth being the result computed by a traditional solver. This we use the same Darcy Flow dataset that was used in [Li et al., 2021c]. We report the mean square error (MSE or ℓ_2).

PSICOV: Protein Distance Prediction. PSICOV studies the use of neural networks in the protein folding prediction pipeline, which has recently received significant attention to the success of methods like AlphaFold [Jumper et al., 2020]. While the dataset and method they use are too large-scale for our purposes, we consider a smaller set of protein structures to tackle the specific problem of inter-residual distance predictions outlined in [Adhikari, 2020c]. 2D large-scale features are extracted from protein sequences, resulting in input feature maps with a massive number of channels. Correspondingly, the labels are pairwise-distance matrices with the same spatial dimension. The evaluation metric is mean absolute error (MAE or ℓ_1) computed on distances below 8 Å, referred to as MAE_8.

Cosmic: Identifying Cosmic Ray Contamination. Images from space-based facilities are prone to corruption by charged particles collectively referred to as "cosmic rays." Cosmic rays on images should be identified and masked before the images are used for further analysis [Zhang and Bloom, 2020b]. The Cosmic task uses imaging data of local resolved galaxies collected from the Hubble Space Telescope. Inputs and outputs are same-size 2D matrices, with the output predicting whether each pixel in the input is an artifact of cosmic rays. We report the false-negative rate (FNR) of identification results.

ECG: Detecting Heart Disease. Electrocardiograms are frequently used in medicine to diagnose sinus rhythm irregularities. The ECG task is based on the 2017 PhysioNet Challenge [Clifford et al., 2017c], with 9 to 60-second ECG recordings sampled at 300 Hz and labeled using four classes: normal, disease, other, or noisy rhythms. Recordings are processed using a fixed sliding window of 1,000 ms and stride of 500 ms. We report the F1-score according to the challenge's guidelines.

Satellite: Satellite Image Time Series Analysis. Satellite image time series (SITS) are becoming more widely available in earth monitoring applications. Our dataset comes from Formosat-2 satellite images acquired over Toulouse, France [Petitjean et al., 2012]. Available in multiple channels, SITS track the land cover changes over several years as each pixel in the image represents a geographical region. The goal of the Satellite task is to generate land cover maps for geo-surveying. Specifically, a series of pixels in a given color channel which constitutes a time series to be classified into 24 land cover types.

DeepSEA: Predicting Functional Effects From Genetic Sequences. Predicting chromatin effects of genetic sequence alterations is a significant challenge in the field to understand genetic diseases. DeepSEA [Zhou and Troyanskaya, 2015], provides a compendium of genomic profiles from the Encyclopedia of DNA Elements (ENCODE) project [Consortium et al., 2004] to train a predictive model estimating the behavior of chromatin proteins, divided into 919 categories. Due to computation constraints, we subsample 36 of these categories as per [Zhang et al., 2021b] and further take 5% of the training data for prediction. We report the area under the receiver operating characteristic (AUROC) following the previous work.

10.C Baselines

Wide ResNet With Hyperparameter Tuning. Architectures based on ResNet [He et al., 2016a] are a common first choice by practitioners faced with a new domain [Fonseca et al., 2020, Adhikari, 2020c]; it is thus a natural source of fixed-architecture baselines for our study. We use the Wide ResNet variant [Zagoruyko and Komodakis, 2016b] with 16 layers and a widen factor of 4, and apply its original training routine directly for the constrained practitioner. For the other practitioner, we wrap the training procedure with a hyperparameter tuner, ASHA [Li et al., 2018b], an asynchronous version of Hyperband [Li et al., 2017]. Given a range for each hyperparameter, ASHA uniformly samples configurations and uses brackets of elimination: at each round, each configuration is trained for some epochs, before the algorithm selects the best-performing portion based on validation metrics. This procedure is useful for finding suitable hyperparameters in an easy-to-use, automated fashion.

Cell-Based Search Using DARTS. The first NAS paradigm we consider is cell-based NAS. These methods first search for a genotype, a cell containing neural operations. During evaluation, an architecture is constructed by replicating the searched cell and stacking them together. The most popular search space for this approach is DARTS [Liu et al., 2019c], which assigns one of eight operations to six edges in two types of cells: "normal" cells preserve the shape of the input while "reduction" cells downsample it. For dense tasks, we do not use the reduction cell to prevent introducing a bottleneck. For 1D tasks, all convolutions in the cell are converted from 2D to 1D. Finally, to adhere to standard ML practices, we do *not* adapt the standard DARTS pipeline from the original paper. As this search space has been heavily studied, we use as a search routine a recent approach, GAEA PC-DARTS (GAEA), that achieves some of the best-known results on CIFAR-10 and ImageNet [Li et al., 2021a]. This NAS approach, due to its heavy retraining routine, is compared to the tuned WRN baseline of the less-resource-constrained practitioner.

Macro NAS Using DenseNAS. The second NAS paradigm we consider is macro NAS. Instead of building from a fixed cell, it requires the specification of a supernet with different inter-connected blocks. These blocks and connections are then pruned to construct an architecture. For our benchmark, we choose a recent search space, DenseNAS [Fang et al., 2020],

which has near state-of-the-art results on ImageNet. DenseNAS searches for architectures with densely-connected, customizable routing blocks to emulate DenseNet [Huang et al., 2017]. In our experiments, we use the ResNet-based search space, DenseNAS-R1, which contains all neural operations of the WRN backbone. For 2D tasks, we adapt two super networks from the one used for ImageNet as inputs to the search algorithm. The super network for dense tasks maintains the same spatial dimensions without downsampling to avoid bottlenecks, and we lower the learning rate for evaluating architectures to prevent divergence. For transferring to 1D tasks, all network operations are switched from 2D to 1D. Other training and evaluation procedures are identical to those in the original paper and uniform across all tasks. DenseNAS is quick to search and evaluate, making it comparable to the fixed WRN baseline.

We apply another search method to the fixed DenseNAS space to study the relative importance of search algorithms. Random search is implemented through randomly sampling architectures from the DenseNAS space and validating them after a brief training period of 10 epochs before evaluating the best performer. To ensure fairness of comparison, we allot equal GPU hours to random search and regular DenseNAS search, additionally applying a soft constraint that random architecture model sizes should not surpass DenseNAS searched architecture sizes significantly.

Domain-Specific NAS Baselines: Auto-DL and AMBER. To study the importance of search spaces, we further handpick two domain-specific NAS approaches applicable only to a subset of the tasks. Using an encoder-decoder architecture, Auto-DeepLab (Auto-DL) [Liu et al., 2019b] is designed for dense prediction, e.g., semantic segmentation. While the decoder is fixed, Auto-DL searches for an encoder via first-order DARTS. We evaluate Auto-DL on Darcy Flow, PSICOV, and Cosmic tasks.

AMBER [Zhang et al., 2021c] aims to automate neural network design for 1D genomic data. This framework establishes a 12-layer network backbone and parametrizes a long-short term memory network (LSTM) as a controller to search for suitable 1D operations and residual connections, following the ENAS [Pham et al., 2018] optimization protocol. At each step, the controller samples architectures to compute reward based on area under the receiver operating characteristics (AUROC) before outputting the highest-reward architecture. We evaluate AMBER on the ECG, Satellite, and DeepSEA tasks.

General-Purpose Baselines: Perceiver IO and XGBoost. As the overarching theme of NAS-Bench-360 is evaluate NAS methods on a wide variety of diverse tasks and when to even use NAS methods over fixed baselines, general-purpose non-NAS methods are obvious points of comparison. We evaluate two such baselines on NAS-Bench-360: the recent transformer-based Perceiver IO [Jaegle et al., 2022], and the popular non-deep learning baseline, XGBoost [Chen and Guestrin, 2016b]. Perceiver IO is a general-purpose transformer architecture that is designed to handle arbitrary input and output dimensionalities with minimal changes to its encoder and decoder networks—as such, we evaluate Perceiver IO on all 10 NAS-Bench-360 tasks. We note that Perceiver IO tends to perform best in settings with more data than in present in the NAS-Bench-360 tasks. Similarly, the popular gradient-boosting method, XGBoost, is applicable to a wide variety of tasks and learning objectives, including single-output and multi-output classification and regression problems, which covers all 10 tasks in NAS-Bench-360. For efficiency and comparison to deep learning methods, we employ the GPU-based implementation of histogram gradient-boosting in XGBoost.

10.D Comparison of NAS with Expert Architectures

Hand-crafted networks are selected according to best-effort search. In Table 10.2, we compare all NAS methods to these hand-crafted networks, denoted as expert architectures. Figure 10.6 illustrates a comparison between best-performing NAS methods vs. best non-NAS methods. Surprisingly, GAEA PC-DARTS beats all the baselines on a portion of the tasks.

Here is a brief summary of these expert models and their citations:

1. DenseNet-BC (CIFAR-100): a more sophisticated version of ResNet, achieving state-of-the-art performance on vision classification [Huang et al., 2017].

2. S2CNN (Spherical): a spherical CNN contains special operations designed for spherical signals, state-of-the-art on spherically-projected MNIST [Cohen et al., 2018a].

3. Fourier Neural Operator (FNO) Network (Darcy Flow): via parametrization in Fourier space, FNO can efficiently learn a family of partial differential equations and their mapping to solutions [Li et al., 2021c].

4. DEEPCON (PSICOV): a dilated-convolution neural network combined with dropout to optimize for protein distance prediction [Adhikari, 2020b].

5. deepCR-mask (Cosmic): a modified version of UNet retaining data dimension to keep pixels at the borders to suit astronomy applications, state-of-the-art on this task [Zhang and Bloom, 2020b].

6. Attention-based model (NinaPro): a lightweight feed-forward neural network adopting attention modules in place of convolutions [Josephs et al., 2020].

7. VGG-like (FSD50K): a smaller VGG network with output features combining both global max pooling and average pooling for audio [Fonseca et al., 2020].

8. ResNet-1D (ECG): ResNet with 1D convolution, using a larger kernel size of 16 and a stride of 2 for all convolutions. The architecture is state-of-the-art on several time-series prediction tasks in medicine [Hong et al., 2020].

9. ROCKET (Satellite): a simple linear classifier with random convolution kernel as a feature extractor, achieving state-of-the-art performance on UCR time-series prediction tasks [Dempster et al., 2020].

10. DeepSEA model (DeepSEA): the original 1D convolution model accompanying the dataset, state-of-the-art on DeepSEA itself [Zhou and Troyanskaya, 2015].

10.E Experiment Details

10.E.1 Hyperparameter Tuning and Backbone

We use a wide residual network with 16 layers and a widening factor of 4 (WRN-16-4) for all tasks.

For tuning hyperparameters, we use ASHA's default elimination schedule and search over 7 to 256 randomly sampled hyperparameter configurations matching GAEA PC-DART's runtime. The maximum epochs that a single configuration could be trained is equal to that of Wide ResNet's default, 200.

We have selected the following hyperparameter ranges for tuning the Wide ResNet backbone:

Table 10.6: Experiment training runtimes of NAS-Bench-360 (GPU hours)

Task	GAEA	DenseNAS	WRN	AMBER / Auto-DeepLab
CIFAR-100	9.5	2.5	2	n/a
Spherical	16.5	2.5	2	n/a
Darcy Flow	6.5	0.5	0.5	5.5
PSICOV	18	24	18.5	19
Cosmic	21.5	2.5	4	17.5
NinaPro	0.5	0.2	0.2	n/a
FSD50K	37	4.5	4	n/a
ECG	140	6.5	5	27
Satellite	28	3	4.5	26
DeepSEA	39.5	2	1.5	28

- $\log_{10}$(learning rate): Unif[-4, -1]
- momentum: Unif{0.0, 0.3, 0.6, 0.9}
- $\log_{10}$(weight decay): Unif[-5, -2]
- dropout: Unif{0.0, 0.3, 0.6}
- batch size: 128 (all point tasks except FSD50K), 4 (Darcy Flow), 8 (PSICOV, Cosmic), 256(FSD50K, ECG, Deepsea), 4096 (Satellite)

10.E.2 Reference Runtimes

Using a Nvidia V100 GPU with 16GB of memory, we have recorded the following runtimes for each experiment in this benchmark in Table 10.6. Overall, GAEA PC-DARTS is more costly than backbone with hyperparameter optimization, which is more costly than DenseNAS. The protein tasks requires heavy computation since the data is not static but generated during training.

10.E.3 Model Sizes and FLOPS Statistics

Full information of model parameter counts and FLOPs can be found in Table 10.8 and Table 10.9.

10.E.4 Adjustments for Dense Prediction Tasks

On the wide ResNet backbone, we add an adaptive averaging pooling operation to upsample the features back to their original dimensions before output. On the DARTS space, we prevent downsampling and keep spatial dimensions unchanged by disabling reduction cells and replacing them with normal cells. On DenseNAS, we configure the super-network to contain only blocks with the original spatial dimensions.

10.E.5 Adjustments for 1D Prediction Tasks

The WRN-1D does not have a convolution stem and uses larger kernel sizes of 8, 5, 3 in each convolution block. We substitute 2D operations with 1D operations within the DARTS and DenseNAS search spaces.

10.E.6 Random Seeds

For main experiments, we fix the random seed to be 0, 1, 2 for each of the 3 trials respectively.

For AMBER experiments, we completed three trials as the package did not offer the option of setting random seeds.

10.E.7 Correlation Between Performance and Model Size

We plot performances of 30 random architectures from the DenseNAS search space across three tasks in Figure 10.5. From our random search experiment, larger models searched by NAS are not always better-performing. We study the Pearson correlation coefficient between test performance vs. model size in number of parameters for three tasks: FSD50K, Cosmic, and ECG. On Cosmic and ECG, the correlation is very weak ($r = 0.15$ and $r = 0.19$ respectively). On FSD50K, a stronger correlation ($r = 0.79$) is observed but performance varies significantly even for architectures of the same size.

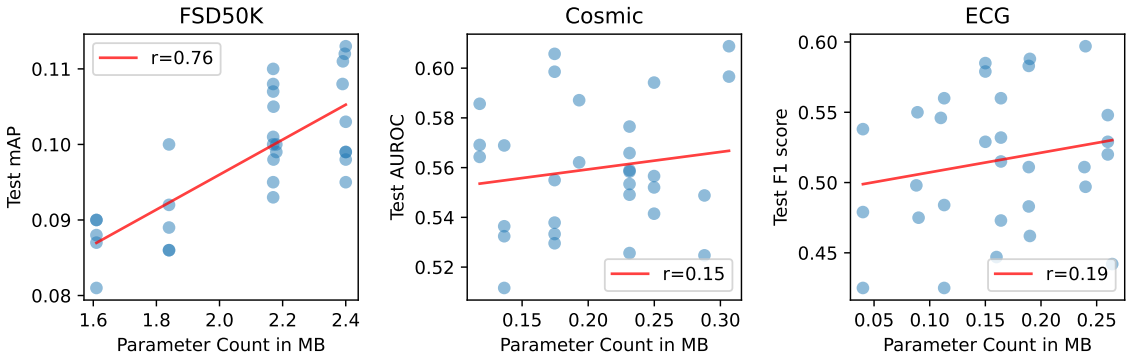


Figure 10.5: Performances by model sizes for three sample tasks.

Table 10.7: Performance of the expert-designed FNO architecture (designed for Darcy Flow) across all dense tasks.

	Darcy Flow	PSICOV	Cosmic
Expert (Darcy Flow)	**0.008±0.001**	4.43±0.055	0.301±0.01
Expert	**0.008±0.001**	**3.35±0.140**	**0.127±0.01**

10.E.8 Internal Validation of Dense Task Diversity

We conduct an internal validation of the diversity of a subset of the NAS-Bench-360 tasks, namely the dense prediction tasks Darcy Flow, PSICOV, and Cosmic. We compare the expert architectures for each task to the expert architecture for Darcy Flow: the FNO-based architecture. The results of this study are shown in Table 10.7. We observe a drop in performance using the expert FNO architecture on our three dense tasks. FNO was designed for Darcy Flow and achieves the best performance on this task across all methods. In contrast, we find that FNO performance is considerably worse on the other two dense tasks, PSICOV and Cosmic.

10.F Dataset Details

10.F.1 Data License

- CIFAR-100: CC BY 4.0 (on `https://www.tensorflow.org/datasets/catalog/cifar100`)

- Spherical CIFAR-100: CC BY-SA
- NinaPro: CC BY-ND
- FSD50k: CC BY 4.0
- Darcy Flow: MIT
- DeepCov, PSICOV: GPL
- Cosmic: CC BY 4.0
- ECG: ODC-BY 1.0
- Satellite: GPL 3.0
- Deepsea: CC BY 4.0

10.F.2 Data Preprocessing Details

CIFAR-100. While the 10,000 testing images are kept aside only for evaluating architectures, the 50,000 training images are randomly partitioned into 40,000 for architecture search and 10,000 for validation. On all of the 50,000 training images, we apply standard CIFAR augmentations including random crops and horizontal flipping, and finally normalize them using a pre-calculated mean and standard deviation of this set. On the 10,000 testing images, we only apply normalization with the same constants.

Spherical. With the same split ratios CIFAR-100, the generated spherical image data is directly used for training and evaluation without data augmentation and pre-processing.

NinaPro. Containing less than 4,000 samples, the data is comprised of single-channel signals with an irregular shape of 16*52 pixels. This task also differs from CIFAR for its class imbalance, as over 65% of all gestures are the neutral position. We split the data using the same ratio as CIFAR, resulting in 2638 samples for training, and 659 samples for validation and testing each. No additional pre-processing is performed.

FSD50K. The raw sound files are first resampled at a frequency of 22,050 Hz and transformed into 96-band, log-mel spectrograms, which is a representation of the sound's power spectrum. Small overlapping audio chunks of 1 second are obtained from these larger clips, resulting in an input size of 101*96 (101 frames of 96-band spectrograms). As data augmentation, background noise of the same frequency is also mixed into 75% of the training data. We split 4,170 clips into the validation set and 10,231 clips into the test set following the original paper. During training, we train on one randomly-sampled chunk, instead of all chunks, from each clip.

Darcy Flow. We use scripts provided by [Li et al., 2021c] to generate the PDEs and their solutions, for a total of 900 data points for training, 100 for validation, and 100 for testing. All input data is normalized with constants calculated on the training set before fed into the neural network and de-normalized following an encode-decode scheme. The solutions, or labels, for the training set are also encoded and decoded this way. The test labels are not processed.

PSICOV. We adopt the chosen subset of DeepCov proteins in [Adhikari, 2020c], consisting of 3,456 proteins each with 128*128 feature maps across 57 channels. 100 proteins from this set are used for validation and the rest for training. Test data for final evaluation is gathered from another set of 150 proteins, PSICOV. Since these produce feature maps that are larger (512*512), we run the prediction network over all of its non-overlapping 128*128 patches.

Cosmic. We use data from a specific filter, F435W, of the space telescope, representing the 3605–4882 Å spectral range. Image stamps of 256*256 pixels are taken from large images. The dataset contains 4,347 stamps for training, and 420 for test, and 483 for validation to match the test set size.

ECG. From the sliding window approach, 12,186 single lead recordings are converted into more than 330,000 recording segments comprised of 261,740 for training, 33,281 for validation, and 33,494 for test. Each segment is of the shape 1*1,000, representing one channel of 1,000-long temporal sequence.

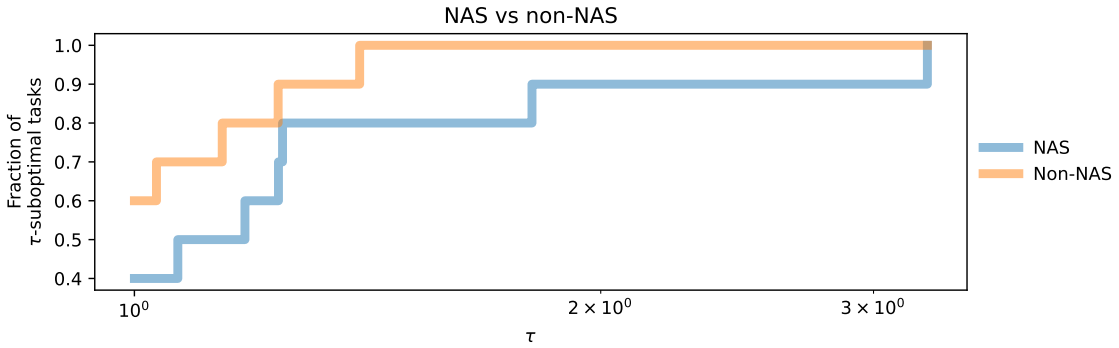


Figure 10.6: Performance profiles on all tasks for best-performing NAS vs. Non-NAS. The y-value indicates the fraction of tasks on which a plotted method's error is within a multiplicative factor τ of the lowest error achieved by all plotted methods..

Satellite. Each satellite time-series is single-channel of length 46 (1*46). After applying standard normalization, we divide the one million entries to 800,000 for training, 100,000 for validation, and 100,000 for test. Zero-padding to 48-length sequences is required for DenseNAS' downsampling network.

DeepSEA. The genome sequences are 1,000-base pair (bp) long and represented as a 1000×4 binary matrix, as each bp is represented as an one-hot encoding corresponding to either A,C,T,G at that location. Total training set size is 71,753. Validation and test sizes that are not subsampled are 2,490 and 149,400 respectively.

10.G Ethics and Responsible Use

Within our array of tasks, only NinaPro, ECG, and DeepSEA contain human-derived data. Our chosen subset of NinaPro contains only muscle movement data without any exposure of personal information. The original experiments to acquire NinaPro data are approved by the ethics commission of the canton of Valais, Switzerland [Atzori et al., 2012a]. The ECG data derives from an open challenge and is provided by the medical device company AliveCor, under the GPL license allowing it for public use. The DeepSEA data derived from ENCODE is part of an international collaborative effort, which is overseen and funded by the National Human Genome Research Institute (NHGRI). For other datasets, we have listed the data licenses in the appendix. While we do not view the specific datasets in NAS-Bench-360 as

Table 10.8: Paramater counts of searched and baseline models for all tasks of NAS-Bench-360. Searched model sizes are reported as mean±standard deviation of three random seeds. Results are reported in millions (M). Architectures with the best performance are bolded.

Search space	Search algo.	CIFAR-100	Spherical	Darcy Flow	PSICOV	Cosmic
DenseNAS	random	1.74±0.12	2.23±0.47	1.00±0.18	1.21±0.16	0.25±0.06
DenseNAS	original	2.03±0.53	1.84±0.15	0.38±0.13	0.93±0.36	0.15±0.16
DARTS	GAEA	4.92±0.28	**1.67±0.14**	0.63±0.08	**0.53±0.05**	0.43±0.15
Auto-DL	DARTS	n/a	n/a	22.98±3.49	6.50±1.84	7.61±2.14
WRN	default	2.77	2.77	2.75	2.76	2.75
Expert	default	**3.08**	0.16	**1.19**	0.60	**0.10**

Search space	Search algo.	NinaPro	FSD50K	ECG	Satellite	DeepSEA
DenseNAS	random	6.80±0.46	**2.40±0.00**	0.18±0.05	0.79±0.16	0.25±0.04
DenseNAS	original	6.69±0.53	1.45±0.00	0.11±0.05	1.08±0.63	0.19±0.00
DARTS	GAEA	3.35±0.48	0.81±0.11	3.31±0.07	**3.35±0.35**	2.91±0.47
AMBER	ENAS	n/a	n/a	6.61±0.33	6.22±1.36	8.44±1.47
WRN	default	**2.75**	2.80	0.50	0.51	0.51
Expert	default	1.36	0.35	**16.5**	0.48	**60.9**

Table 10.9: FLOPS of searched and baseline models for all tasks of NAS-Bench-360. Searched model FLOPS are reported as mean±standard deviation of three random seeds. Results are reported in GFLOPS. Architectures with the best performance are bolded.

Search space	Search algo.	CIFAR-100	Spherical	Darcy Flow	PSICOV	Cosmic
DenseNAS	random	0.46±0.07	0.91±0.07	14.42±2.58	39.80±5.09	8.42±2.11
DenseNAS	original	0.44±0.53	1.84±0.15	5.43±1.82	30.51±11.90	5.00±5.30
DARTS	GAEA	1.42±0.09	**1.91±0.65**	9.33±1.13	**17.74±1.68**	14.27±4.90
Auto-DL	DARTS	n/a	n/a	2.54±1.20	3.43±1.27	2.44±0.26
WRN	default	0.78	2.78	39.72	90.58	90.06
Expert	default	**1.18**	n/a	**n/a**	0.01	**1.96**

Search space	Search algo.	NinaPro	FSD50K	ECG	Satellite	DeepSEA
DenseNAS	random	1.02±0.06	**0.40±0.00**	0.11±0.02	0.02±0.01	0.15±0.02
DenseNAS	original	0.97±0.14	0.80±0.00	0.16±0.03	0.02±0.01	0.10±0.00
DARTS	GAEA	0.89±0.12	2.57±0.47	2.28±0.05	**0.11±0.07**	2.01±0.33
AMBER	ENAS	n/a	n/a	0.03±0.01	0.03±0.01	0.04±0.01
WRN	default	**0.64**	7.56	1.02	0.04	1.02
Expert	default	0.02	0.66	**0.70**	0.01	**0.12**

*some expert models contain non-standard modules without FLOPS count.

potential candidates for misuse, the broader goal of applying NAS to new domains comes with inherent risks that may require mitigation on an application-by-application basis.

Chapter 11

Rethinking Neural Operations for Diverse Tasks

This chapter is based on joint work with Mikhail Khodak, Tri Dao, Liam Li, Christopher Ré, and Ameet Talwalkar, published at NeurIPS 2021. The first two authors contributed equally.

Automated machine learning (AutoML) and neural architecture search (NAS) are often motivated by a vision of democratizing ML by reducing the need for expert design on a variety of tasks. While NAS has grown rapidly with developments such as weight-sharing [Pham et al., 2018] and "NAS-benches" [Ying et al., 2019, Zela et al., 2020], most efforts focus on search spaces that glue together established primitives for well-studied tasks like vision and text [Liu et al., 2019c, Li and Talwalkar, 2019, Xu et al., 2020, Li et al., 2021a] or on issues such as latency [Cai et al., 2020, Fang et al., 2020]. In this work, we revisit the broader vision of NAS and propose to move towards much more general search spaces while still exploiting successful network topologies. To do so we focus on expanding the set of operations, which is usually fairly small; for example, that of the well-studied DARTS space has eight elements: a few types of convolution and pooling layers [Liu et al., 2019c]. The baseline approach for expanding this set—adding operations one-by-one—scales poorly and will not result in new operations when faced with new types of data.

Our core contribution is a re-imagining of NAS operation spaces that drastically expands this set in a principled fashion to include both standard operations as well as a wide range of new ones. To do so we exploit the fact that most standard operations used in

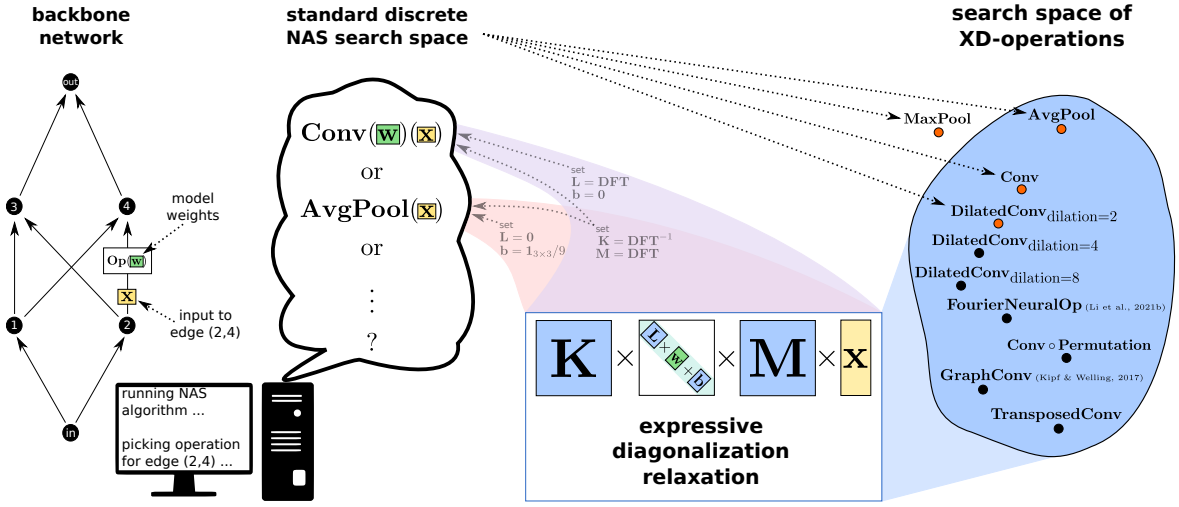


Figure 11.1: Diagram of our search space depicting a NAS method picking an operation for an edge in a backbone network (left). Instead of choosing from a discrete search space, we use a relaxation based on the convolution's diagonalization by the discrete Fourier transform in which the DFTs are replaced by K-matrices [Dao et al., 2020] K, L, and M (middle); these are the main architecture parameters of our new search space over Expressive Diagonalization (XD) operations. This space contains most operations considered in standard NAS and many other important operations in a variety of domains (right).

modern NAS return linear transforms diagonalized by the discrete Fourier transform (DFT). Replacing the DFT matrices in the diagonal decomposition by a more expressive family of efficient linear transforms known as *Kaleidoscope* or *K-matrices* [Dao et al., 2020] yields the set of Expressive Diagonalization (XD) Operations, which comprise a large search space containing various types of grid-based convolutions and pooling, permutations, transposed convolutions, certain kinds of graph convolutions, the Fourier Neural Operator (FNO) [Li et al., 2021c], and infinitely many more. This broad expressivity reflects the key insight of our work: that many of the most important neural operations in ML consist of multiple channels that apply weights w to inputs x by computing

$$\mathrm{K}\,\mathrm{diag}(\mathrm{L}w)\mathrm{M}x \tag{11.1}$$

where the matrices K, L, and M are *efficient* (to represent and apply) and shared across channels.

We leverage XD-operations to take critical steps towards a broader NAS that enables the discovery of good design patterns with limited human specification from data in under-

explored domains. To do so we develop a simple procedure which transforms any backbone convolutional neural network (CNN) into an architecture search space by replacing its operations with XD-operations. This space is then searched using a simple weight-sharing algorithm that needs only a small amount of tuning to find effective operations. As a simple first demonstration, we show that XD-operations yield models that are 15% more accurate than standard discrete search spaces on *permuted* CIFAR-10, highlighting the fragility of standard NAS operation spaces on new datasets, and thus the need for XD-operations.

As our main evaluation, we demonstrate the effectiveness of XD-operations in a series of applications showing that, starting from vanilla CNNs, they consistently outperform custom-designed operations.

- Learning to solve partial differential equations (PDEs): when substituted into a simple CNN backbone, XD-operations outperform convolutions and the dense prediction NAS method Auto-DeepLab [Liu et al., 2019a], and even achieve lower error than custom-designed, state-of-the-art operations (FNOs [Li et al., 2021c]) across three problems with different dimensionalities (Burgers' equation, Darcy Flow, and Navier-Stokes). Our method also maintains consistent performance across different resolutions, a major stated advantage of FNOs over previous methods.
- Protein folding: on the task of predicting residue distances in a polypeptide chain—a key component of the protein folding problem—we substitute XD-operations into vanilla ResNets and achieve lower error than cyclically-dilated ResNets adapted specifically for this setting [Adhikari, 2020a]. Furthermore, our ResNet-34 XD outperforms the reported error of the much deeper Dilated ResNet-258.
- Music modeling: on two next-note prediction tasks, we show that substituting XD-operations into an undilated CNN outperforms temporal convolutional networks (TCNs)—exponentially-dilated 1d CNNs that themselves outperform standard convolutional and recurrent networks [Bai et al., 2018].

Code to reproduce these results is available here: `https://github.com/nick11roberts/XD`. Software to apply XD-operations can be found here: `https://github.com/mkhodak/relax`.

Related Work. AutoML is a well-studied area, with most work focusing on fairly small hyperparameter spaces [Bergstra and Bengio, 2012, Li et al., 2018a] or on NAS [Elsken et al., 2019]. Most NAS operation spaces only contain a few operations such as convolutions [Liu

et al., 2019c, Mei et al., 2020, Zela et al., 2020, Dong and Yang, 2020], which may not be useful for domains where CNNs are ineffective. Applications of NAS outside vision largely follow the same pattern of combining human-designed operations [Nekrasov et al., 2019, Wang et al., 2020a]. On the other extreme, AutoML-Zero [Real et al., 2020b] demonstrates the possibility of evolving all aspects of ML from scratch. We seek to establish a middle ground with large and domain-agnostic search spaces that still allow the use of well-tested methods, e.g. stochastic gradient descent (SGD).

Several papers have generalized the DFT to replace layers in deep nets [Dao et al., 2019, Alizadeh vahid et al., 2020, Ailon et al., 2020, Dao et al., 2020] in order to speed up or add structure to models while *reducing* expressivity. In contrast, we can replace *convolutions* and other layers while *increasing* expressivity by extending their diagonalization via K-matrices. As discussed in Section 11.1, using K-matrices for this directly is inefficient for input dimension > 1.

11.1 The Expressive Diagonalization Relaxation

In this section we overview our main contribution: a large, general search space of neural operations. Formally, we view an architecture as a *parameterizable* object—a mapping from model weights to functions—described by a *labeled* directed acyclic graph (DAG) $\mathcal{G}(V, E)$. Each edge in E has the form $(u, v, \mathbf{Op})$, where $u, v \in V$ are nodes and **Op** is an operation that can be parameterized to define some transformation of the representation at node u; node v aggregates the outputs of its incoming edges into a new representation. For example, the popular ResNet architecture [He et al., 2016a] has many nodes with two incoming edges, one labeled by the convolution operation **Conv** and one by the identity (skip-connect) **Id**, whose outputs it sums and passes to outgoing edges with the same labels. Each architecture has a source node taking in input data and an output node returning a prediction.

Neural architecture search is the problem of automatically selecting an operation for each edge of $\mathcal{G}$ to optimize an objective.[1] For each edge $e \in E$ a NAS algorithm must pick one element of a *search space* $\mathcal{S} = \{\mathbf{Op}_a \,|\, a \in \mathcal{A}\}$ of operations specified by architecture parameters $a \in \mathcal{A}$ to assign to e; in past work, $\mathcal{A}$ usually indexes a small set of operations. As

[1] It is often defined as selecting both operations and a graph topology [Zoph et al., 2018], but if the set of operations contains the zero-operation **Zero** then the former subsumes the latter.

an example, we will refer to a variant[2] $\mathcal{S}_{\textbf{discrete}}$ of the DARTS search space with parameters $\mathcal{A}_{\textbf{discrete}} = \{1,\dots,8\}$ where each operation is one of **Zero**, **Id**, $\textbf{MaxPool}_{3\times3}$, $\textbf{AvgPool}_{3\times3}$, $\textbf{Conv}_{3\times3 \text{ or } 5\times5}$, or $\textbf{DilatedConv}_{3\times3,2 \text{ or } 5\times5,2}$ [Liu et al., 2019c].

Our main contribution is a novel family of operations that comprise a search space containing almost all these operations, in addition to many others that have been found useful on different types of data. The starting point of our construction of these XD-operations is the simple observation that all the operations $\textbf{Op} \in \mathcal{S}_{\textbf{discrete}}$ listed above except $\textbf{MaxPool}_{3\times3}$ are *linear*, i.e. for any model weights w there exists a matrix A_w such that for all inputs x we have $\textbf{Op}(w)(x) = A_w x$. More specifically, all seven of them return convolutions: to see this note that **Zero**, **Id**, and $\textbf{AvgPool}_{3\times3}$ each apply a convolution with filter $\mathbf{0}_{1\times1}$, $\mathbf{1}_{1\times1}$, and $\mathbf{1}_{3\times3}/9$, respectively. This means that most of the operations in the DARTS search space—which is representative of NAS operation spaces in computer vision—share the convolution's diagonalization by the discrete Fourier transform (DFT). Formally, if $A_w \in \mathbb{R}^{n^2\times n^2}$ is the matrix representing a 2d convolution with filter $w \in \mathbb{R}^k$ of kernel size $k \in [n]^2$, then for any 2d input $x \in \mathbb{R}^{n^2}$ we have

$$\textbf{Conv}(w)(x) = A_w x = F^{-1}\,\mathrm{diag}\,(F\underline{w})\,Fx \tag{11.2}$$

Here $[n] = \{1,\dots,n\}$, $\mathrm{diag}(z)$ denotes the diagonal matrix with entries z, $\underline{w} \in \mathbb{R}^{n^2}$ is an appropriate zero-padding of $w \in \mathbb{R}^k$, and $F \in \mathbb{C}^{n^2\times n^2}$ is the 2d DFT (a Kronecker product of two 1d DFTs).

This diagonalization explicates both the computational and representational efficiency of the DARTS operations, as the DFT and its inverse can be applied in time $\mathcal{O}(n \log n)$ and stored with $\mathcal{O}(n \log n)$ bits. It also suggests a natural way to dramatically expand the operation space while preserving these efficiencies: just replace matrices F and F^{-1} in (11.2) by any one of a general family of efficient matrices. Doing so yields the single-channel version of our *expressive diagonalization* (XD) operations:

$$\textbf{XD}^{\mathbf{1}}_{\alpha}(w)(x) = \mathrm{Real}\,(K\,\mathrm{diag}\,(L\underline{w})\,Mx) \tag{11.3}$$

Here architecture parameter $\alpha = (K, L, M)$ sets the matrices replacing F and F^{-1} in Equa-

[2]For memory-efficiency, all convolutions in the original DARTS search space are separable [Liu et al., 2019c].

tion 11.2.

The main remaining question is the family of efficient matrices to use, i.e. the domain of the architecture parameters K, L, and M. For this we turn to the Kaleidoscope matrices, or *K-matrices* [Dao et al., 2020], which generalize F and F^{-1} to include all computationally efficient linear transforms with short description length, including important examples such as sparse matrices and permutations. To obtain this general family, K-matrices allow the DFT's butterfly factors—matrices whose products yield its efficient implementation—to take on different values. While a detailed construction of K-matrices can be found in the original paper, we need only the following useful properties: they are as (asymptotically) efficient to apply as DFTs, are differentiable and can thus be updated using gradient-based methods, and can be composed (made "deeper") to make more expressive K-matrices.

Specifying that K, L, and M in Equation 11.3 are K-matrices largely completes our core contribution: a new search space $\mathcal{S}_{\mathbf{XD}}$ of XD-operations with K-matrix architecture parameters. We give a full multi-channel formalization in N dimensions, as well as an overview of its expressivity, in Section 11.2. First, we note some key aspects of this new search space:

- Complexity: $\mathbf{XD}^{\mathbf{1}}_{\alpha}(w)$ requires three K-matrices and $\mathcal{O}(1)$ filter weights to represent, i.e. description length $\mathcal{O}(n \log n)$; this is larger than a regular convolution (which has no architecture parameters) but is not quadratic in the input size like a linear layer. Applying $\mathbf{XD}^{\mathbf{1}}_{\alpha}$ requires multiplication by three K-matrices, yielding a theoretical per-channel time complexity of $\mathcal{O}(n \log n)$, matching the efficiency of convolutions. However, as XD-operations strictly generalize convolutions they are more expensive to apply in-practice; we detail these costs both in the application sections and as appendix table, and we view improving upon them as an important future direction.
- Initialization: a crucial advantage of XD-operations is that we can initialize or *warm-start* search using operations with known constructions. In particular, since we can recover convolutions (11.2) by setting architecture parameters $K = F^{-1}$, $L = F$, and $M = F$ in Equation 11.3, we can always start search with any CNN backbone. We use this extensively in experiments. $\times$
- K-matrices: as they contain all efficient linear transforms, K-matrices can represent all functions returned by XD-operations, including convolutions. However, for input dimension and filter size > 1 the only known way is to apply K-matrices directly to flattened inputs $x \in \mathbb{R}^{n^N}$, yielding much worse description length $\mathcal{O}(n^N \log n)$. In contrast,

as detailed in Section 11.2, our diagonalization approach uses Kronecker products to apply DFTs to each dimension separately, yielding description length $\mathcal{O}(n \log n)$. It is thus the first (and in some sense, "right") method to use such matrices to replace convolutions. Furthermore, diagonalization allows us to separate model weights w from architecture parameters α, letting the former vary across channels while fixing the latter.

Finally, we address the fact that the architecture parameters of $\mathcal{S}_{\mathbf{XD}}$ are continuous, not discrete, contrasting with much of the NAS literature. This can be viewed as a natural extension of the weight-sharing paradigm [Pham et al., 2018], in which continuous relaxation enables updating architecture parameters with gradient methods. For example, many algorithms traverse the relaxed DARTS search space

$$\tilde{\mathcal{S}}_{\mathbf{discrete}} = \left\{ \sum_{i=1}^{8} \lambda_i \, \mathbf{Op}_i \, | \lambda_i \geqslant 0, \sum_{i=1}^{8} \lambda_i = 1 \right\},$$

defined via DARTS operations $\mathbf{Op}_i \in \mathcal{S}_{\mathbf{discrete}}$ and architecture parameters λ_i in the 8-simplex; most search spaces then require discretizing after search via a rounding procedure that maps from the simplex to $\mathcal{A}_{\mathbf{discrete}}$. Note that the fully continuous nature of XD-operations means that we will only evaluate the final network returned by search. In particular, while some weight-sharing papers also report the correlation between true architecture performance and that indicated by the shared weights [Yang et al., 2020], there is no obvious way to define a ranking or sampling distribution over XD-operations in order to do so. This also means that our final architecture will not be more efficient than the supernet, unlike other weight-sharing methods that do discretize.

11.2 XD-Operations and Their Expressivity

Here we formalize XD-operations and show what operations they include. We first define operations:

Definition 11.1. *A* **parameterizable operation** *is a mapping* $\boldsymbol{Op} : \mathcal{W} \mapsto \mathcal{F}$ *from parameter space* $\mathcal{W}$ *to a space* $\mathcal{F} = \{\boldsymbol{Op}(w) : \mathcal{X} \mapsto \mathcal{Y} | w \in \mathcal{W}\}$ *of* **parameterized functions** *from input space* $\mathcal{X}$ *to output space* $\mathcal{Y}$*. A* **search space** *is a set of operations with the same* $\mathcal{W}$*,* $\mathcal{X}$*, and* $\mathcal{Y}$*.*

For example, if $\mathcal{X} = \mathcal{Y} = \mathbb{R}^n$ and $\mathcal{W} = \mathbb{R}^{n \times n}$ then each $W \in \mathcal{W}$ defines a parameterized linear layer that for each $x \in \mathcal{X}$ returns $\mathbf{Lin}(W)(x) = Wx$. Here **Lin** is the parameterizable

operation and for each W the linear map $\mathbf{Lin}(W)$ is the parameterized function.

From Definition 11.1, we say a search space can *express* a specific operation if it contains it. Crucially, the ability of a parameterizable operation $\mathbf{Op}_1$ to express a parameterized function $\mathbf{Op}_2(w)$ output from another operation $\mathbf{Op}_2$ given the right set of weights w does *not* imply that a search space containing $\mathbf{Op}_1$ can express $\mathbf{Op}_2$. For example, $\mathbf{Lin}(\times I_n) = \mathbf{Id}(W)\ \forall\, W \in \mathbb{R}^{n\times n}$ but $\mathbf{Lin}(W) \neq \mathbf{Id}(W)\ \forall\, W \neq I_n$, so a search space containing the linear operation **Lin** cannot express the skip-connection **Id**, despite the fact that **Lin** can be parameterized to compute the identity.

Formalizing Multi-Channel XD-Operations. Recall the single-channel XD-operation $\mathbf{XD}^{\mathbf{1}}_{\alpha}$ in Equation 11.3 specified by three-matrix architecture parameter $\alpha = (K, L, M)$. For input dimension $N \geqslant 1$, every matrix $B \in \alpha$ is a Kronecker product of N K-matrices of depth $d \in \mathbb{Z}^3_+$, i.e. $B = \bigotimes_{i=1}^{N} B_i$ for K-matrices $B_i \in \mathcal{C}^{n\times n}$ of depth $d_{[1]}$, $d_{[2]}$, or $d_{[3]}$ for $B = K$, L, or M, respectively.[3] Roughly speaking, $\mathbf{XD}^{\mathbf{1}}_{\alpha}$ can return any linear operation that is diagonalized by K-matrices and is thus efficient to compute and represent, e.g. any convolution (recall we recover the diagonalization of $\mathbf{Conv}(w)$ in Equation 11.2 by setting K, L, and M appropriately in Equation 11.3). However, $\mathbf{XD}^{\mathbf{1}}_{\alpha}$ cannot represent efficient *parameter-free* operations such as skip-connections and average-pooling, both common in NAS. In particular, the only way to always ignore the model weights w is to set one of the K-matrices to zero, producing the zero-operation. We avoid this by adding a bias $b \in \mathcal{C}^{n^N}$ as an architecture parameter, yielding the *biased* single-channel XD-operation:[4]

$$\mathbf{XD}^{\mathbf{1}}_{\alpha,b}(w)(x) = \mathrm{Real}\left(K\,\mathrm{diag}(L\underline{w} + b)M\underline{x}\right) \tag{11.4}$$

This lets us define skip-connections (set $K = M = I_{n^N}$, $L = \mathbf{0}_{n^N\times n^N}$, and $b = \mathbf{1}_{n^N}$) and average-pooling (set $K = F^{-1}$, $L = \mathbf{0}_{n^N\times n^N}$, $M = F$, and b to be F multiplied by a pooling filter).

Lastly, we use $\mathbf{XD}^{\mathbf{1}}_{\alpha,b}$ to construct multi-channel "layers" that pass multiple input features through multiple channels and re-combine them as multiple output features. This follows the primary way of using convolutions in deep nets. The key insight here is that we will

[3] A depth-d K-matrix is a product of d depth-1 K-matrices.

[4] Zero-padding x as well lets the input to be smaller than the output if needed, e.g. for transposed convolutions.

share the same parameterizable operation (specified by α and b) across all channels, just as in convolutional layers.

Definition 11.2. *Let $a = (\alpha, b, C)$ be an architecture parameter containing a triple $\alpha = (K, \times_L, M)$ of Kronecker products of N K-matrices with depths $d \in \mathbb{Z}_+^3$, a bias $b \in \mathcal{C}^{n^N}$, and channel gates $C \in \mathcal{C}^{c\times c}$.*[5] *Using "$\bigoplus$" to denote concatenation, the* **XD-operation** $\mathbf{XD}_a$ *of depth d specified by a is a parameterizable operation on parameter space $\mathcal{W} = \mathbb{R}^{c\times c\times k}$ consisting of c^2 filters of size $k \in [n]^N$ that outputs parameterized functions on $\mathcal{X} = \mathbb{R}^{c\times m^N}$ for $m \leqslant n$ mapping every $x \in \mathcal{X}$ to*

$$\mathbf{XD}_a(w)(x) = \bigoplus_{i=1}^{c} \sum_{j=1}^{c} C_{[i,j]} \mathbf{XD}^{\mathbf{1}}_{\alpha,b}(w_{[i,j]})(x_{[j]}) \tag{11.5}$$

The last architecture parameter C allows interpolation between all-to-all layers ($\times_C = \mathbf{1}_{c\times c}$), e.g. multi-channel convolutions, and layers where each channel is connected to one other channel ($C = I_c$), e.g. skip-connections and average-pooling. We note that we use $\mathcal{S}_{\mathbf{XD}}$ to describe the set of operations covered by Definition 11.2 and conclude our construction by discussing two properties:

- Kernel size: the weight-space available to an XD-operation is $\mathbb{R}^{c\times c\times n^N}$; however, since we will initialize search with existing CNNs, we will zero-pad to have the same weight-space $\mathbb{R}^{c\times c\times k^N}$ as the convolutions with filter size $k \leqslant n$ that they replace. This preserves the weight count but also means that if the backbone has 3×3 filters our search space will *not* contain 5×5 convolutions. Experimentally, we find that relaxing the constraint to allow this does not significantly affect results on image tasks, so we do not do so in subsequent applications to avoid increasing the weight count.
- Depth: an XD-operation's depth is a triple describing the depths of its K-matrices K, L, and M. Increasing it trades off efficiency for expressivity; for example, in the next section we describe operations that we can show are contained in $\mathcal{S}_{\mathbf{XD}}$ if L or M have depth > 1. By default we will set the depth to be the minimum needed to initialize search with the backbone operation.

[5] For simplicity we formalize the case where all N dimensions have the same input size and there is an identical number c of input and output channels; both are straightforward to extend.

Expressivity of XD-Operations. For many papers that replace deep net layers with efficient linear transforms [Moczulski et al., 2015, Dao et al., 2020], the question of expressivity comes down to the transform capacity. For example, layers with a K-matrix in every channel can represent a different transform in each, thus allowing the output to be any combination of efficient linear operations. Our case is less straightforward since we care about expressivity of the search space, not of parameterized functions, and our approach is less-expressive *by design* as all channels share K-matrices K, L, and M. The latter can be thought of as a useful inductive bias on NAS: the set of XD-operations is still much broader than the set of convolutions, but the way in which model weights are applied is the same across all channels.

Expressivity results are a way to see if this bias is useful or constraining. Here we summarize some important operations that are 1d XD-operations; proofs can be found in the appendix and are straightforward to extend to multi-dimensional inputs. Formally, there exists $d \in \mathbb{Z}_+^3$ such that the set of XD-operations of depth d over weights $\mathcal{W} = \mathbb{R}^{c \times c \times k}$ and inputs $\mathcal{X} = \mathbb{R}^m$ for $m \leqslant n$ contains

1. convolutions with filter size $\leqslant k$, dilation $\leqslant \lfloor \frac{n-1}{k-1} \rfloor$, stride $\leqslant n-1$, and arbitrary channel groups.
2. parameter-free operations **Id**, **Zero**, and $\textbf{AvgPool}_s$ for any kernel size $s \leqslant n$.
3. composing 1 or 2 with multiplication of all input or output channels by a bounded-depth K-matrix.

Note this does not account for *all* important XD-operations, e.g. we show in the appendix that they also express Fourier Neural Operators [Li et al., 2021c] with $\leqslant \lfloor k/2 \rfloor$ modes and any transposed convolutions whose stride equals the dilated kernel size.[6] Still, the first two items account for non-separable variants of most operations considered in past NAS work in computer vision, excluding the nonlinear **MaxPool** [Ying et al., 2019, Dong and Yang, 2020]. Note depthwise-separable convolutions *are* contained in the set of compositions of XD-operations. The third item implies that XD-operations can express the basic and diffusion graph convolutions over fixed graphs [Kipf and Welling, 2017a, Li et al., 2018c]: both are point-wise convolutions composed with sparse multiplication by a modified adjacency matrix, which K-matrices can represent efficiently.

[6] This restriction still includes transposed convolutions used in well-known architectures such as U-Net [Ronneberger et al., 2015].

As a concrete example, consider dilated convolutions, which for $k > 1$ and dilation factor $d \geqslant 1$ apply filters of effective size $(k-1)d+1$ with nonzero entries separated by $d-1$ zeros. One could hope to express the application of $\mathbf{DilatedConv}_{k,d}$ to an input $x \in \mathbb{R}^n$ in the single-channel setting as $F^{-1}\operatorname{diag}(F\operatorname{diag}(p_{k,d})\underline{w})Fx$, where $p_{k,d} \in \{0,1\}^n$ zeroes out appropriate entries of $\underline{w}$, but this requires filter size $(k-1)d+1 > k$, increasing the number of weights. Instead, we can use a permutation $P_{k,d} \in \{0,1\}^{n\times n}$ before the DFT to place the k entries of $\underline{w}$ into dilated positions:

$$\mathbf{DilatedConv}_{k,d}(w)(x) = F^{-1}\operatorname{diag}(FP_{k,d}\underline{w})Fx \tag{11.6}$$

As permutations are depth-2 K-matrices [Dao et al., 2020], we can express $\mathbf{DilatedConv}_{k,d}$ with an XD-operation of depth $(1,3,1)$, with $K = F^{-1}$, $L = FP_{k,d}$, and $M = F$.

11.3 Finding and Evaluating XD-Operations

This section outlines a simple procedure that we use to evaluate XD-operations. Recall that NAS methods specify architectures by assigning operations to each edge $(u, v, \mathbf{Op})$ of a computational graph. We aim to simultaneously find good operations and model weights, a goal distinct from the classic *two-stage* NAS formulation, which finds assignments in an initial search phase before training the resulting architecture from scratch [Ying et al., 2019]. However, the use of weight-sharing [Pham et al., 2018] extends NAS to *one-shot* objectives where weights and architectures are jointly optimized. Under weight-sharing, architecture parameters become weights in a larger "supernet," extending the hypothesis class [Li et al., 2021a].

To assess XD-operations directly we assume the user provides a starter network with existing edge labels $\mathbf{Op}_{u,v}$ as a backbone. We transform this into a weight-sharing supernet by reparameterizing each operation $\mathbf{Op}_{u,v}$ as an XD-operation $\mathbf{XD}_{a_{u,v}}$ with architecture parameter $a_{u,v}$. Then we simultaneously train both $a_{u,v}$ and the model weights $w_{u,v}$ associated with each edge as follows:

- Architecture parameters $a_{u,v}$ are initialized using the original operation used by the CNN backbone by setting $\mathbf{Op}_{u,v} = \mathbf{XD}_{a_{u,v}}$; $a_{u,v}$ is then updated via SGD or Adam [Kingma and Ba, 2015]. We tune step-size, momentum, and the number of "warmup" epochs: initial epochs during which only model weights $w_{u,v}$ are updated. This can be viewed as a

specialized step-size schedule.

- Model weights $w_{u,v}$ are initialized and updated using the routine provided with the backbone.

This approach allows us to use established topologies and optimizers while searching for new operations, thus aligning with the goal for Sections 11.4, 11.5, and 11.6: to improve upon the CNN backbones that practitioners often use as a first attempt. As a simple example, we start by applying the procedure to image classification. Since this is not the main objective of our work, we treat it as a warmup and consider two datasets: CIFAR-10 and a variant where the images' rows and columns are permuted. On CIFAR-10 we do *not* expect to see much improvement from XD-operations over the CNN backbone used to initialize search, as convolutions are already the "right" operation for images. On the other hand, the "right" operation on permuted data, at least in layer one, is an inverse permutation followed by convolution; as this is an XD-operation[7], here we do hope to see improvement.

Using LeNet [LeCun et al., 1999] and ResNet-20 [He et al., 2016a] as backbones, we compare applying our algorithm to XD-operations with two baselines: (1) using just the backbone CNN and (2) applying a similar method to the relaxed set $\tilde{\mathcal{S}}_{\textbf{discrete}}$ of DARTS operations from Section 11.1. To optimize over $\tilde{\mathcal{S}}_{\textbf{discrete}}$ we take an approach similar to DARTS: parameterize the simplex using a softmax and apply Adam. We experiment with both a uniform initialization and one biased towards the backbone's operation. While both $\mathcal{S}_{\textbf{XD}}$ and $\mathcal{S}_{\textbf{discrete}}$ contain LeNet's $\textbf{Conv}_{5\times5}$ and ResNet's $\textbf{Conv}_{3\times3}$ and **Id**, for LeNet's $\textbf{MaxPool}_{3\times3}$ layer we initialize with the closest operation. For direct comparison, both search spaces employ weights with maximum filter size 5×5 and for both we evaluate the shared weights rather than retraining, which we find hurts $\tilde{\mathcal{S}}_{\textbf{discrete}}$. We set the XD-operations' depth to $d = 3_3$ to express the dilated convolutions in $\mathcal{S}_{\textbf{discrete}}$ and convolutions composed with permutations.

In Table 11.1, we see that while both the relaxed discrete NAS operations and XD-operations perform comparably on regular images, XD-operations achieve around 15% better accuracy with both backbones when the images are permuted.[8] Note that even

[7] Recall $\mathcal{S}_{\textbf{XD}}$ includes compositions of convolutions with multiplication by a K-matrix, e.g. a permutation.

[8] Full accuracy can be recovered via an auxiliary loss encouraging permutation-like K-matrices [Dao et al., 2020].

Table 11.1: Search space comparison on CIFAR-10. Validation accuracies are averages of three trials. While we use small CNNs for exploration, XD-operations can also be used with high-performance backbones to obtain $> 95\%$ accuracy (c.f. the appendix).

Backbone search space	CIFAR-10	Permuted CIFAR-10*	Cost (hours†)
LeNet	75.5 ± 0.1	43.7 ± 0.5	0.3
$\tilde{\mathcal{S}}_{\textbf{discrete}}$	75.6 ± 3.4	47.7 ± 1.0	1.0
$\mathcal{S}_{\textbf{XD}}$	77.7 ± 0.7	63.0 ± 1.0	0.9
ResNet-20	91.7 ± 0.2	58.6 ± 0.7	0.6
$\tilde{\mathcal{S}}_{\textbf{discrete}}$	92.7 ± 0.2	58.0 ± 1.0	5.3
$\mathcal{S}_{\textbf{XD}}$	92.4 ± 0.2	73.5 ± 1.6	5.6

* No data augmentation used in the permuted case.

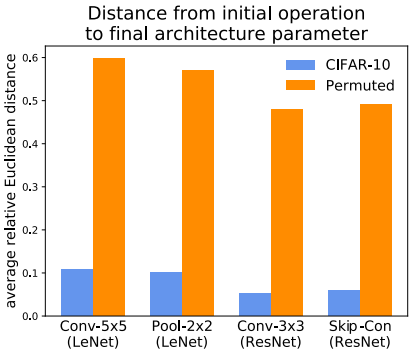


Figure 11.2: On permuted images, where convolutions are not the "right" operation, we find XD-operations that are farther away from the operations of the initial CNN backbone.

networks obtained by running state-of-the-art NAS procedures such as GAEA PC-DARTS [Li et al., 2021a] and DenseNAS [Fang et al., 2020] on permuted CIFAR-10 achieve only 66.3% and 61.6% accuracy, respectively, despite using millions more parameters than ResNet-20. While it is not straightforward to understand the recovered XD-operations that perform so well, we can use the relative Euclidean distance of their architecture parameters from initialization as a proxy for novelty; in Figure 11.2 we see that on regular images our procedure finds operations that are quite similar to convolutions, but on permuted data they are much further away. These results show that to enable NAS on diverse data, we will need a search space that contains truly novel operations, not just combinations of existing ones. In the remainder of this chapter, we study more diverse and realistic tasks that show further evidence that $\mathcal{S}_{\textbf{XD}}$ is a strong candidate for this.

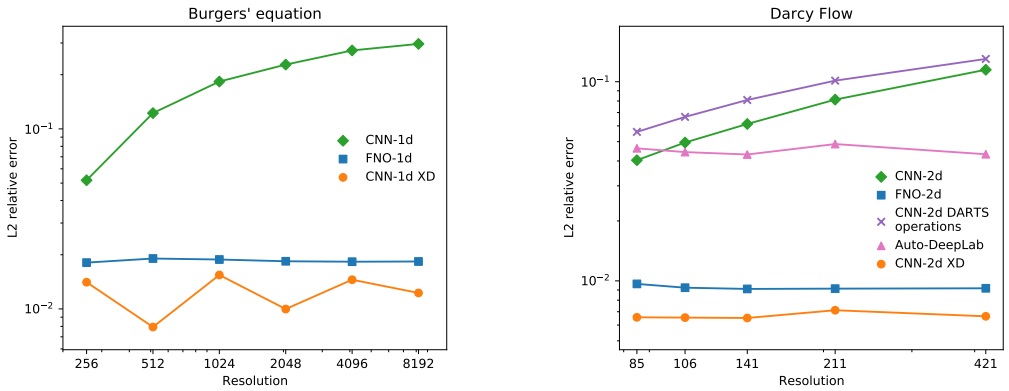


Figure 11.3: Relative error on Burgers' equation (left) and Darcy Flow (right) across different resolutions.

11.4 Application: Learning to Solve Partial Differential Equations

As our first non-vision application, we consider the task of solving PDEs, an important application area of ML in the natural sciences [Li et al., 2015, 2018d, Sirignano and Spiliopoulos, 2018]. In our setup, data generated by classical PDE solvers is used to learn functions from some initial condition or setting to the corresponding PDE solution, with the goal of replacing the solver by a deep net forward pass; the latter can be orders of magnitude faster. A recent state-of-the-art approach for this introduces Fourier Neural Operators [Li et al., 2021c], operations that significantly improve upon previous neural approaches across three different PDE settings. To evaluate the ability of XD-operations to compete with such custom-designed operations starting from simple CNN backbones, we will investigate the same three PDEs that they study: Burgers' equation, Darcy Flow, and the 2d Navier-Stokes equations, which involve 1d, 2d, and 3d data, respectively. The first two are studied across multiple resolutions, while the last one is studied at different viscosities.

As before, we start with a simple CNN backbone—the type a scientist might use in a first attempt at a solution—and replace all convolutions by XD-operations. We initially hope to do better than this backbone, but ambitiously also hope to compete with the custom-designed FNO. The specific CNN we use is simply the FNO architecture of the appropriate dimension N but with all N-dimensional FNOs replaced by N-dimensional convolutions; this performs similarly to their CNN baselines [Li et al., 2021c]. In all cases we compare mainly to the CNN backbone and our reproduction of the FNO results, as the latter exceeds all other neural

methods; a complete results table is provided in the appendix. Our reproduction of FNO is slightly worse than their reported numbers for Burgers' equation and slightly better in the other two settings. Note that on the Navier-Stokes equations we only compare to the 3d FNO on the two settings in which we were able to reproduce their approach; moreover, we do *not* compare to their use of a 2d FNO plus a recurrent net in time, but in-principle XD-operations can also be substituted there. In the 2d Darcy Flow case we also include comparisons to DARTS operations in the simple CNN backbone, as in Section 11.3, and to Auto-DeepLab (AutoDL) [Liu et al., 2019a], a well-known NAS method for dense prediction. For evaluating XD-operations we again follow the procedure in Section 11.3, in which we tune only the architecture optimizer; notably, we do this only at the lowest resolutions. At all dimensions we use XD-operations of depth $\mathrm{d} = \mathbf{1}_3$; in addition, in dimensions $\mathrm{N} > 1$ we fix the architecture biases b and channel gates C to $\mathbf{0}$ and $\mathbf{1}$, respectively, to conserve memory at higher resolutions. At lower ones we find that the performance difference is negligible.

We report our results for the Burger's equation and Darcy Flow in Figure 11.3; for 2d Navier-Stokes the results are in Table 11.2. In all cases we dramatically outperform the CNN backbone used to initialize XD-operations; furthermore, we also achieve better error than FNO, despite it being custom-made for this problem. In particular, we find that XD-operations have higher *training error* but generalize better (c.f. the appendix). Figure 11.3 also shows that XD-operations perform consistently well across resolutions, a major advantage of FNOs over previous methods, whose performance was tightly coupled to the discretization [Li et al., 2021c]. Notably, CNN performance worsens with higher resolution, unlike that of XD and FNO. Finally, we also substantially outperform DARTS operations and AutoDL in 2d, although the latter is at least consistent across resolutions. These results provide strong evidence that XD-operations are a useful search space for discovering neural operations, even in domains where the convolutions used to initialize them perform much worse than state-of-the-art. Note that these results do come at a cost of slower training and inference: XD-operations are roughly an order of magnitude slower than FNOs, despite having fewer parameters in 2d and 3d. This still yields solvers one-to-two orders of magnitude faster than classical solvers, maintaining usefulness for the problem.

Table 11.2: Relative test error on the 2d Navier-Stokes equations at different settings of the viscosity ν and time steps T. Best results in each setting are bolded.

	$\nu = 10^{-4}$, T = 30	$\nu = 10^{-5}$, T = 20
CNN-3d (our baseline)	0.325	0.278
FNO-3d (reproduced)	0.182	0.177
CNN-3d XD (ours)	**0.172**	**0.168**

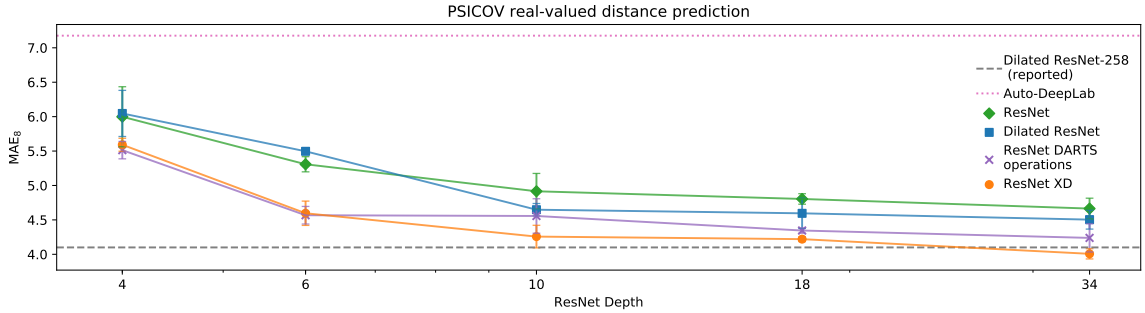


Figure 11.4: ResNet XD outperforms both baseline and dilated ResNets on PSICOV. At the highest depth we test we also outperform the reported MAE_8 of the much deeper Dilated ResNet-258 [Adhikari, 2020a].

11.5 Application: Real-Valued Distance Prediction for Protein Folding

As a second scientific application, we consider the task of inferring the 3d "folded" structure of a polypeptide chain, which yields important insights into the function of the resulting protein [Jones et al., 2011]. This problem is a high-priority challenge in biology and has recently seen significant ML-driven advances from deep learning methods such as AlphaFold [Jumper et al., 2021, 2020] and PDNET [Adhikari, 2020a]. These typically involve training a network to predict pairwise physical distances between residues in the chain. We work with the PDNET benchmark, which consists of a training set of 3,356 proteins, a validation set of 100 of proteins, and the PSICOV [Jones et al., 2011] test set of 150 proteins. PDNET is designed to be more accessible than datasets used by large-scale methods such as AlphaFold, which are not always publicly available and/or require massive compute [Jumper et al., 2021, 2020]. We follow the PDNET training procedure [Adhikari, 2020a] and evaluate

test set performance using their MAE_8 metric for assessing long-range distances.

As before we start with simple CNN backbones—in this case ResNets. We choose this to compare most directly to the custom-designed architecture used by PDNET, consisting of a Dilated ResNet characterized by its use of a cyclically increasing dilation rate across ResNet blocks [Adhikari, 2020a]. At a sufficient depth, the Dilated ResNet is shown to outperform a standard pre-activation ResNet adapted to this task [Adhikari, 2020a]. Our goal will be to see whether we can start with the vanilla ResNet and use XD to outperform both it and the specialized Dilated ResNet. We also aim to outperform the DARTS operations baseline from the previous two sections as well as the AutoDL NAS approach for dense prediction. We use XD-operations of depth $\mathrm{d} = \mathbf{1}_3$ and fix the architecture biases and channel gates as before to conserve memory. We evaluate architectures of different depths—4, 6, 10, 18, and 34—by varying the number of ResNet blocks used in the backbone architecture and baseline.

We report the results as averages across three trials for each depth in Figure 11.4. Notably, while Dilated ResNet slightly outperforms ResNet, ResNet XD outperforms both dilated and standard ResNets at all depths. This provides further evidence that XD-operations can outperform specialized operations for diverse domains, even when initialized naively as standard convolutions. XD also outperforms AutoDL, which does poorly, and DARTS operations, except at the two smaller depths where performance is similar. Moreover, our ResNet-34 XD's MAE_8 of 4.0 also improves upon PDNET's reported MAE_8 of 4.1 attained by the much deeper Dilated ResNet-258 [Adhikari, 2020a]; however, in our reproduction Dilated ResNet-258 achieved an MAE_8 of 3.5. Given the trend in Figure 11.4, where XD-operations consistently improve the backbone architecture of the same depth, we conjecture that ResNet-258 XD could further improve upon this result. We leave scaling XD-operations to such deeper networks to future work.

11.6 Application: Music Modeling

Our final application is to music modeling, i.e. learning to predict the next note from sheet music [Allan and Williams, 2005]. The dominant approaches for such tasks are recurrent nets [Hochreiter and Schmidhuber, 1997a] and Transformers [Vaswani et al., 2017b], but recent work has shown that specially-designed convolutional models can also be made competitive at similar model sizes [Bai et al., 2018, 2019]. We will consider the temporal convolutional network (TCN) [Bai et al., 2018], which improves upon a regular CNN by

Table 11.3: XD-operations compared to recent results in music modeling. We report average loss across three trials. The best result on each task is bolded.

Method (source)	JSB Chorales	Nottingham
Best recurrent [Bai et al., 2018]	8.43	3.29
TCN [Bai et al., 2018]	8.10	3.07
Transformer [Wang et al., 2020b]	-	3.34
R-Transformer [Wang et al., 2020b]	-	**2.37**
Undilated TCN (our baseline)	8.16 ± 0.04	3.23 ± 0.02
TCN (reproduced)	8.17 ± 0.01	2.97 ± 0.01
Undilated TCN XD (ours)	$\mathbf{8.07 \pm 0.01}$	2.84 ± 0.02

having the dilation factor grow exponentially across layers. The tasks we study are on the JSB Chorales and Nottingham corpora, used in the original evaluation of TCNs [Bai et al., 2018]. As the baseline we take the TCN and set all dilation factors to one (undilated); our goal will be to start with this undilated network and match or outperform the custom dilation design of the TCN.

The results presented in Table 11.3 show that we achieve this goal, as we outperform both the undilated baseline and the TCN on both tasks. While the simple undilated backbone that we initialize with turns out to already match the TCN on JSB Chorales, on Nottingham our approach demonstrates that XD-operations can be used to outperform hand-designed architectures starting from vanilla CNNs.[9] Where possible we also compare to other known results; XD-operations outperforms all of these except the R-Transformer [Wang et al., 2020b], a model combining recurrent nets and self-attention, on Nottingham.

Together with our results on PDEs and proteins, our study of music modeling provides further evidence that XD-operations can effectively find good operations using standard backbones on diverse tasks. One notable difficulty here is causality enforcement: making sure the input data does not contain the target when predicting the next entry. While TCNs can efficiently do so via temporal shifts, we do it in a brute-force manner by treating sequences of length n as $n-1$ data-points with masked targets. This is expensive and thus limits our evaluation to small music tasks. A fruitful direction for future work is thus to examine whether it is possibly to directly enforce causality in XD-operations, e.g. by forcing

[9] In the appendix we report similar improvements on two other tasks on which TCNs were evaluated—permuted MNIST and Penn TreeBank—that we do not discuss in detail as our focus is on under-explored tasks.

architecture parameters K and M to be lower triangular; since a product of lower triangular matrices is again lower triangular, the entire operation is then a multiplication of the input sequence by a lower triangular matrix, which suffices to prevent causality violations.

11.7 Conclusion

This work aims to transition NAS from combining existing operations designed for vision and text to finding novel and effective operations in many domains. To do so we introduced a new search space of XD-operations and demonstrated its effectiveness on diverse tasks. Combining XD-operations with standard topology-search NAS, warm-starting search from non-standard operations such as graph convolutions and FNOs,[10] improving the computational limitations described earlier, and constructing spaces containing missing operations such as BatchNorm [Ioffe and Szegedy, 2015] and self-attention [Vaswani et al., 2017b] are all promising future directions. Finally, note that our goal—lowering the barrier for applying ML—necessarily comes with the possibility of misuse. Mitigating this involves developing tools for application-specific concerns, e.g. privacy and fairness, that go beyond the error metrics we target.

[10] In this direction, we found that initializing XD with FNO did *worse* than initializing with convolutions on Burgers' equation and Darcy Flow, a surprising result given how much better FNO is than the baseline CNN. Similarly, initializing XD with convolutions dilated as in the original TCN did not lead to significant improvement, except in one setting, over undilated initialization. See the appendix for more details and results.

11.A Expressivity Results

Here we collect results on the expressivity of the set of XD-operations. For simplicity, our results will be in the following single-dimensional ($N = 1$) setting:

Setting 11.3. *We consider input spaces of form $\mathcal{X} = \mathbb{R}^{c \times m}$ for input size $m \in \mathcal{N}$ and channel count $c \in \mathcal{N}$ and parameter spaces $\mathcal{W} = \mathbb{R}^{c \times c \times k}$ for filter size $k \in [n]$, where output size $n \geqslant m$ is a power of 2.*

It is straightforward to extend the results to multiple dimensions using Kronecker products and to input sizes other than powers of two using padding. Note that all of our results will also assume a circular padded domain.

11.A.1 Convolutions

Definition 11.4. *A* **convolution** *in Setting 11.3 with filter size k, dilation $d \in [\lfloor \frac{n-1}{k-1} \rfloor]$, stride $s \in [n-1]$, and channel groups described by a matrix $B \in \{0,1\}^{n \times n}$ s.t. $B_{[i,j]} = 1$ if channels i and j are in the same group and 0 otherwise is a parameterizable operation that for any weight $w \in \mathcal{W}$ outputs a function mapping every $x \in \mathcal{X}$ to*

$$\frac{1}{n}\begin{pmatrix} \operatorname{diag}(\mathfrak{a}_s(\underbrace{\mathbf{1}_{\lceil \frac{n}{s} \rceil}})) \sum_{j=1}^{c} B_{[1,j]} F_n^{-1} \operatorname{diag}(F_n \mathfrak{a}_d(\underline{w_{[1,j]}})) F_n x_{[j]} \\ \vdots \\ \operatorname{diag}(\mathfrak{a}_s(\underbrace{\mathbf{1}_{\lceil \frac{n}{s} \rceil}})) \sum_{j=1}^{c} B_{[c,j]} F_n^{-1} \operatorname{diag}(F_n \mathfrak{a}_d(\underline{w_{[c,j]}})) F_n x_{[j]} \end{pmatrix} \tag{11.7}$$

where $F_n \in \mathcal{C}^{n \times n}$ is the $n \times n$ DFT and $\mathfrak{a}_d : \mathbb{R}^n \mapsto \mathbb{R}^n$ is an atrous permutation of a vector that is equivalent to multiplication by some permutation matrix $P_d \in \{0,1\}^{n \times n}$. We will use $\boldsymbol{Conv}_k$ to denote the case of $d = 1$, $s = 1$, and $B = \mathbf{1}_{c \times c}$.

Claim 11.5. *All multi-channel convolutions of the form given in Definition 11.4 are contained in the search space of XD-operations of depth $(1, 3, 1)$.*

Proof. Setting the architecture parameters to be $K = \operatorname{diag}(\mathfrak{a}_s(\underbrace{\mathbf{1}_{\lceil \frac{n}{s} \rceil}}))F_n^{-1}$, $L = F_n P_d$, $M = F_n$, $b = \mathbf{0}_n$, and $C = B$, and noting that (a) the DFT and its inverse are both depth 1 K-matrices, (b) multiplying a K-matrix by a diagonal matrix is another K-matrix of the same depth, and

(c) permutation matrices are K-matrices of depth 2 yields the result. These three facts can be found in the original paper [Dao et al., 2020]. □

Remark 11.6. *Note that for the case of dilation* $d = 1$ *the result in Claim 11.5 holds with depth* $\mathbf{1}_3$.

11.A.2 Parameter-Free Operations

Definition 11.7. *The* **skip-connection** *in Setting 11.3 is parameterizable operation that outputs a function mapping every* $x \in \mathcal{X}$ *to itself. The* **zero-operation** *in Setting 11.3 is parameterizable operation that outputs a function mapping every* $x \in \mathcal{X}$ *to* $\mathbf{0}_{c \times n}$.

Claim 11.8. *The skip-connection and zero-operation are both contained in the search space of XD-operations of depth* $\mathbf{1}_3$.

Proof. For both set the architecture parameters to be $K = F_n^{-1}$, $L = \mathbf{0}_{n \times n}$, $M = F_n$, and $C = I_c$. To obtain the skip-connection set $b = \mathbf{1}_n$; to obtain the zero-operation set $b = \mathbf{0}_n$. □

Definition 11.9. *An* **average pooling** *operation in Setting 11.3 with filter size* k, *dilation* $d \in [\lfloor \frac{n-1}{k-1} \rfloor]$, *and stride* $s \in [n-1]$ *is parameterizable operation outputs a function mapping every* $x \in \mathcal{X}$ *to the output of a convolution (as in Definition 11.4) with the same filter size, dilation, and stride, channel groups described by* $B = I_c$, *and filters* $w_{[j,j]} = \mathbf{1}_k / k \; \forall \, j \in [c]$.

Claim 11.10. *All average pooling operations are contained in the search space of XD-operations of depth* $\mathbf{1}_3$.

Proof. Setting the architecture parameters to be $K = \text{diag}(\mathfrak{a}_s(\underline{\mathbf{1}_{\lceil \frac{n}{s} \rceil}}))F_n^{-1}$, $L = \mathbf{0}_{n \times n}$, $M = F_n$, $b = \mathfrak{a}_d(\underline{\mathbf{1}_k/k})$, and $C = I_c$ and noting that (a) the DFT and its inverse are both depth 1 K-matrices and (b) multiplying a K-matrix by a diagonal matrix of the same depth is another K-matrix of the same depth yields the result. □

11.A.3 Compositions with Multiplication by a Fixed K-Matrix

Definition 11.11. *A* **fixed linear operation** $\boldsymbol{Lin}_A$ *in Setting 11.3 with fixed matrix* $A \in \mathbb{R}^{n \times n}$ *is a parameterizable operation that outputs a function mapping every* $x \in \mathcal{X}$ *to* $\boldsymbol{Lin}_A(w)(x) = \begin{pmatrix} Ax_{[1]} & \cdots & Ax_{[c]} \end{pmatrix}^T$. *For example,* $\boldsymbol{Lin}_{I_c} = \boldsymbol{Id}$.

Definition 11.12. *Let* $\boldsymbol{Op}_1$ *and* $\boldsymbol{Op}_2$ *be two parameterizable operations in Setting 11.3 with* $\mathcal{X}$. *Then for any weight* $w \in \mathcal{W}$ *their* **composition** $\boldsymbol{Op}_1 \circ \boldsymbol{Op}_2$ *outputs the parameterized function* $\boldsymbol{Op}_1(w) \circ \boldsymbol{Op}_2(w)$.

Claim 11.13. *Let* $\boldsymbol{Op}$ *be a parameterizable operation in Setting 11.3 that is contained in the set of XD-operations of some depth* $d \in \mathbb{N}^3$ *and let* A *be a K-matrix of depth* d'. *Then* $\boldsymbol{Op} \circ \boldsymbol{Lin}_A$ *is contained in the set of XD-operations of depth* $(d_{[1]}, d_{[2]}, d_{[3]} + d')$ *and* $\boldsymbol{Lin}_A \circ \boldsymbol{Op}$ *is contained in the set of XD-operations of depth* $(d_{[1]} + d', d_{[2]}, d_{[3]})$.

Proof. Let K and M be the first and last K-matrices of the representation of **Op** as an XD-operation, which thus have depth at most $d_{[1]}$ and $d_{[3]}$, respectively. Then the representation of $\mathbf{Op} \circ \mathbf{Lin}_A$ as an XD-operation is the same except with depth $d_{[3]} + d'$ K-matrix MA as the last K-matrix, and similarly the representation of $\mathbf{Lin}_A \circ \mathbf{Op}$ as an XD-operation is the same except with depth $d_{[1]} + d'$ K-matrix AK as the first K-matrix. □

11.A.4 Other Named Operations

Definition 11.14. *Suppose we have a fixed* n*-node graph with adjacency matrix* A *and degree matrix* D, *and let* $\hat{A}$ *and* $\hat{D}$ *be the adjacency and degree matrices, respectively, of the same graph but with added self-loops. Then regular* **graph convolution** *[Kipf and Welling, 2017a] in Setting 11.3 with* $k = 1$ *is a parameterizable operation that for any weight* $W \in \mathcal{W}$ *outputs*

a function mapping every $x \in \mathcal{X}$ *to* $\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}x^T w$ *and the* **diffusion graph convolution** *[Li et al., 2018c] in Setting 11.3 with* $k = 1$ *is a parameterizable operation that for any weight* $W \in \mathcal{W}$ *outputs a function mapping every* $x \in \mathcal{X}$ *to* $D^{-1}Ax^T w$.

Claim 11.15. *Suppose* A *and* $\hat{A}$ *can be represented by K-matrices of depth* d *and* $\hat{d}$*, respectively. Then the corresponding graph convolution is contained in the search space of XD-operations of depth* $(1, 1, \hat{d}+1)$ *and the corresponding diffusion graph convolution in that of depth* $(1, 1, d+1)$.

Proof. For any $G \in \mathbb{R}^{n\times n}$ we have $Gx^T w = \mathbf{Lin}_G(w)(x)w = \mathbf{Conv}_1(w)(\mathbf{Lin}_G(w)(x)) = (\mathbf{Conv}_1 \circ \mathbf{Lin}_G)(w)(x)$. The result follows by Claims 11.5 and 11.13, the fact that a K-matrix multiplied by a diagonal matrix is another K-matrix of the same depth, and by substituting $G = \hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$ (for graph convolution) or $G = D^{-1}A$ (for diffusion graph convolution). □

Remark 11.16. *Note that the above claim is meaningful because adjacency matrices of realistic graphs are usually sparse and sparse matrices can be efficiently represented as K-matrices [Dao et al., 2020].*

Definition 11.17. *A* **Fourier neural operator** *(FNO) [Li et al., 2021c] in Setting 11.3 with even* k *and thus* $k/2$ *modes is a parameterizable operation that for any weight* $w \in \mathcal{W}$ *outputs a function mapping every* $x \in \mathcal{X}$ *to*

$$\begin{pmatrix} \text{Real}\left(\sum_{j=1}^{c} F_n^{-1}\,\text{diag}(\begin{pmatrix} w_{[1,j,1:k/2]} + i w_{[1,j,k/2+1:k]} & \mathbf{0}_{n-k/2} \end{pmatrix}^T) F_n x_{[j]}\right) \\ \vdots \\ \text{Real}\left(\sum_{j=1}^{c} F_n^{-1}\,\text{diag}(\begin{pmatrix} w_{[c,j,1:k/2]} + i w_{[c,j,k/2+1:k]} & \mathbf{0}_{n-k/2} \end{pmatrix}^T) F_n x_{[j]}\right) \end{pmatrix} \tag{11.8}$$

Claim 11.18. *The FNO with* $k/2$ *modes is contained in the search space of XD-operations of depth* $(1, 4, 1)$.

Proof. Setting the architecture parameters to be $K = F_n^{-1}$, $L \in \mathbb{C}^{n\times n}$ the n-sparse matrix mapping $\underline{w}$ to $\begin{pmatrix} w_{[1,j,1:k/2]} + i w_{[1,j,k/2+1:k]} & \mathbf{0}_{n-k/2} \end{pmatrix}^T$, $M = F_n$, $b = \mathbf{0}_n$, and $C = \mathbf{1}_{c\times c}$, and noting that an n-sparse matrix is a depth-4 K-matrix [Dao et al., 2020] yields the result. □

Remark 11.19. *If we allow the parameter space in Setting 11.3 to be complex then the FNO with all* k *modes will be contained in the search space of XD-operations of depth* $\mathbf{1}_3$.

Definition 11.20. *Each channel of* **transposed convolution** *with stride* $\mathrm{d}(\mathrm{k}-1)+1$*, where* k *is the kernel size and* d *is the dilation rate, computes a feature map in which each input element is replaced by that element multiplied by the dilated filter of size* $\mathrm{d}(\mathrm{k}-1)+1$*. The multi-channel extension of this over parameter space* $\mathcal{W} = \mathbb{R}^{\mathrm{c}\times\mathrm{c}\times\mathrm{k}}$ *is similar to that for standard convolutions.*

Claim 11.21. *All transposed convolutions with stride equal to the dilated kernel size are contained in the search space of XD-operations of depth* $(1,3,3)$*.*

Proof. A transposed convolution is equivalent to a regular convolution with the same filter applied to the input after it has been zero-padded and then permuted to separate all entries by $\mathrm{d}(\mathrm{k}-1)$ zeros. Since permutations are K-matrices of depth 2 the result follows by Claims 11.5 and Claim 11.13. □

Definition 11.22. *A* **depthwise-separable convolution** *in Setting 11.3 with filter size* k *but with parameter space* $\mathcal{W} = \mathbb{R}^{\mathrm{c}\times\mathrm{k}} \times \mathbb{R}^{\mathrm{c}\times\mathrm{c}}$ *is a parameterizable operation that for any weight* $w \in \mathcal{W}$ *outputs* $\boldsymbol{Conv}_1(w_{[2]}) \circ \boldsymbol{Conv}_{\mathrm{k},\mathrm{I_c}}(w_{[1]})$*, where* $\boldsymbol{Conv}_{\mathrm{k},\mathrm{I_c}}$ *denotes the convolution in Definition 11.4 with* $\mathrm{B} = \mathrm{I_c}$*.*

Remark 11.23. *Since both* $\boldsymbol{Conv}_1$ *and* $\boldsymbol{Conv}_{\mathrm{k},\mathrm{I_c}}$ *are XD-operations, by definition depthwise-separable convolutions are contained in the search space of composed XD-operations, which by Claim 11.8 also contains all of the above operations.*

11.B Practical Complexity of XD-Operations

Table 11.4: Comparison of the computational and memory costs of XD-operations when substituted for convolutions. For simplicity, we consider cases with 2d inputs and where the channel and bias parameters are fixed.

	input	kernel	minutes / epoch		memory (Gb)		param. ($\times 10^6$)	
Task (backbone)	size	size	**Conv**	**XD**	**Conv**	**XD**	**Conv**	**XD**
CIFAR-10 (WRN-40-4)	32	3	1.4	4.3	3.73	15.6	8.96	9.08
Darcy Flow (Conv4*)	85	13	0.028	0.14	4.51	5.53	0.701	0.744
PSICOV (ResNet-18)	128	3	5.9	11	1.50	10.7	0.038	0.549

* Four-layer convolutional network with parameterized skip (shortcut) connections derived from the FNO network [Li et al., 2021c] as described in Section 11.4.

In this section we report a detailed comparison of computational costs of the XD-operation compared to a convolution; this is presented in Table 11.4. Due to their familiarity, we present results for tasks that have 2d inputs and thus use 2d convolutions in their default backbone. Note that since XD-operations are more general than convolutions, they must by definition be at least as expensive as convolutions in both computation and memory. While in this chapter our focus is on absolute performance using learning metrics (e.g. test error), we view finding a good tradeoff between the performance of XD-operations on certain tasks and convolutions, for example by restricting the expressivity of XD-operations, as important directions for future work.

11.C Experimental Details: CIFAR-10 and Permuted CIFAR-10

Table 11.5: Architecture optimizer settings on CIFAR-10 tasks. Note that the step-size is updated using the same schedule as the backbone.

search space	backbone	task	optimizer	initial step-size	warmup epochs	perturb
$\tilde{\mathcal{S}}_{\textbf{discrete}}$	LeNet	CIFAR-10	Adam	1E-1	0	0.1
		Permuted	Adam	1E-1	50	0.875
	ResNet-20	CIFAR-10	Adam	1E-3	0	0.1
		Permuted	Adam	1E-1	0	0.875
$\mathcal{S}_{\textbf{XD}}$	LeNet	CIFAR-10	Adam	1E-4	0	-
		Permuted	Adam	1E-3	0	-
	ResNet-20	CIFAR-10	Adam	1E-4	50	-
		Permuted	Adam	1E-3	0	-

For our experiments with image classification backbones we use the standard CIFAR-10 data [Krizhevksy, 2009] and a permuted version where all rows and columns are identically permuted. For unpermuted data we use standard data augmentation [He et al., 2016a] while for permuted data we do not use any data augmentation. As specified in Section 11.3, we keep the training routine of the model weights the same and tune only the architecture optimizer, the settings of which are specified in Table 11.5. Note that for the DARTS operation space we specify a "perturb" parameter that specifies how unbiased the initial architecture parameters are towards the backbone operation; specifically, we initialize architecture parameters so as to assign one minus this quantity as the weight to the backbone operation, so 0.875 means the initialization is uniform (since $|\tilde{\mathcal{S}}_{\textbf{discrete}}| = 8$) while 0.1 means the backbone operation is assigned 0.9 of the weight.

11.C.1 LeNet

The LeNet backbone we consider consists of two $\textbf{Conv}_{5\times5}$ layers, each followed by $\textbf{MaxPool}_{2\times2}$, and two fully connected layers. When warm-starting with XD-operations we use $\textbf{AvgPool}_{2\times2}$ instead of $\textbf{MaxPool}_{2\times2}$, while when warm-starting with the DARTS operations we use $\textbf{MaxPool}_{3\times3}$. For the baseline training routine we use 200 epochs of Momentum(0.9), with the first 100 at learning rate 0.01, the next 50 at 0.005, and the last 50 at 0.001.

11.C.2 ResNet-20

We use the implementation and training routine provided here: `https://github.com/akamaster/pytorch_resnet_cifar10`. When replacing operations in the backbone we substitute for both the $\mathbf{Conv}_{3\times3}$ operations and the skip-connections **Id**; some of the latter are downsampled, which XD-operations can handle as strides.

11.C.3 WideResNet-40-4

We use the same implementation as for ResNet-20 but adapt the original WRN training routine [Zagoruyko and Komodakis, 2016b], except with weight-decay set to 10^{-4} (as in ResNet-20); on the regular CIFAR-10 tasks this does not seem to affect performance. To conserve computation and memory, we do not tune the architecture optimizer parameters here and simply use the same ones used for ResNet-20; furthermore, we fix the channel and bias parameters of XD-operations and do not allow the kernel size to be larger the 3×3. Because of these modifications, we only use our evaluation here as a sanity check for large-network performance of XD-operations and do not include it in the main results.

11.C.4 DARTS Cell Search

To search the full DARTS search space, which is a standard NAS benchmark, we use GAEA PC-DARTS, a recent state-of-the-art method [Li et al., 2021a], using code made available by the authors here: `https://github.com/liamcli/gaea_release`. On CIFAR-10 we simply use their best reported cell but evaluate it using the “base" routine [Yang et al., 2020], i.e. without auxiliary losses or additional data augmentation; this is to obtain fair comparison with the other backbone models. Note that the model is still much larger and the training routine much more intensive. On permuted data we follow the standard three-stage pipeline in which we run search four times, train all four found cells and select the best one, and finally train that cell multiple times.

11.C.5 DenseNAS Search

We use the DenseNAS search and evaluation code released by the authors here: `https://github.com/JaminFong/DenseNAS`. While the search space is designed for ImageNet [Russakovsky et al., 2015], we adapt it to CIFAR-10 by taking the DenseNAS-R1 setting and downscale the input sizes to match 32x32 images used.

Table 11.6: Search space comparison on CIFAR-10. Validation accuracies are averages of three trials.

Backbone	Search Space	CIFAR-10	Permuted*	Cost (hours†)
LeNet	backbone	75.5 ± 0.1	43.7 ± 0.5	0.3
	$\tilde{\mathcal{S}}_{\textbf{discrete}}$	75.6 ± 3.4	47.7 ± 1.0	1.0
	$\mathcal{S}_{\textbf{XD}}$	77.7 ± 0.7	63.0 ± 1.0	0.9
ResNet-20	backbone	91.7 ± 0.2	58.6 ± 0.7	0.6
	$\tilde{\mathcal{S}}_{\textbf{discrete}}$	92.7 ± 0.2	58.0 ± 1.0	5.3
	$\mathcal{S}_{\textbf{XD}}$	92.4 ± 0.2	73.5 ± 1.6	5.6
WRN-40-4	backbone	95.2 ± 0.1	64.7 ± 0.9	4.6
	$\tilde{\mathcal{S}}_{\textbf{discrete}}$	95.2 ± 0.2	61.3 ± 1.3	19.9
	$\mathcal{S}_{\textbf{XD}}$	95.0 ± 0.1	72.9 ± 0.8	14.3
ResNet-18	DenseNAS	94.5 ± 0.3	61.6 ± 3.3	3.6
Cell	DARTS‡	96.0 ± 0.2	66.3 ± 0.5	28.6

* No data augmentation used in the permuted case.
† On a V100 GPU; time for DARTS Cell is training cost only.
‡ Search using GAEA PC-DARTS [Li et al., 2021a]; training using "base" routine [Yang et al., 2020].

11.D Experimental Details: Solving PDEs

For our PDE experiments, we use the FNO code and setup [Li et al., 2021c] provided here: `https://github.com/zongyi-li/fourier_neural_operator`. We use the same training routine and settings as the backbone architecture for each task and only tune the architecture optimizer. We consider the following hyperparameters for the architecture optimizer: Adam vs. SGD (with or without momentum), initial learning rate, and number of warmup epochs. The final hyperparameters for each task can be found in Table 11.7. Our CNN backbone is analogous to the FNO architecture used for each problem. In particular, the CNN backbone architecture used for each task is simply the FNO architecture where FNO layers of dimension N with m modes are replaced by N-dimensional convolutional layers with filters of size $(m+1)^N$ and circular padding to match the dimensionality of FNO. In Table 11.8 and Table 11.9 we present reported [Li et al., 2021c], reproduced, and our own results on the 1d Burgers' equation and 2d Darcy Flow.

For AutoDL we use the code and setup provided here: `https://github.com/NoamRosenberg/autodeeplab`. We only conduct search on the lowest resolution and use the

resulting architecture at higher resolutions. Search was conducted for 40 epochs, as in the original paper, and the search learning rate was tuned.

Table 11.7: Architecture optimizer settings on PDE tasks. Note that the step-size is updated using the same schedule as the backbone.

task	optimizer	initial step-size	warmup epochs
1d Burgers' equation	Adam	1E-3	0
1d Burgers' equation (FNO init)	Momentum(0.5)	1E-4	250
2d Darcy Flow	Momentum(0.5)	1E-1	0
2d Darcy Flow (FNO init)	Momentum(0.5)	1E-1	0
2d Navier Stokes ($\nu = 10^{-4}, T = 30$)	Momentum(0.5)	5E-3	0
2d Navier Stokes ($\nu = 10^{-5}, T = 20$)	Momentum(0.5)	1E-3	0

Table 11.8: Test relative errors on the 1d Burgers' equation. We were not able to match the FNO-1d results reported by the authors [Li et al., 2021c] using their published codebase, however, our proposed XD operations outperform our reproduction of their results at every resolution. Furthermore, we outperform their reported test relative errors on every resolution except $s = 4096$, where we roughly match their performance.

Method (source)	$s = 256$	$s = 512$	$s = 1024$	$s = 2048$	$s = 4096$	$s = 8192$
NN [Li et al., 2021c]	0.4714	0.4561	0.4803	0.4645	0.4779	0.4452
GCN [Li et al., 2021c]	0.3999	0.4138	0.4176	0.4157	0.4191	0.4198
FCN [Li et al., 2021c]	0.0958	0.1407	0.1877	0.2313	0.2855	0.3238
PCANN [Li et al., 2021c]	0.0398	0.0395	0.0391	0.0383	0.0392	0.0393
GNO [Li et al., 2021c]	0.0555	0.0594	0.0651	0.0663	0.0666	0.0699
LNO [Li et al., 2021c]	0.0212	0.0221	0.0217	0.0219	0.0200	0.0189
MGNO [Li et al., 2021c]	0.0243	0.0355	0.0374	0.0360	0.0364	0.0364
FNO-1d [Li et al., 2021c]	0.0149	0.0158	0.0160	0.0146	**0.0142**	0.0139
CNN (ours)	0.0518	0.1220	0.1830	0.2280	0.2730	0.2970
FNO-1d (reproduced)	0.0181	0.0191	0.0188	0.0184	0.0183	0.0183
CNN XD (ours)	**0.0141**	**0.0079**	**0.0154**	**0.0099**	0.0145	**0.0123**
FNO-1d XD (ours)	0.0153	0.0154	0.0154	0.0167	0.0160	0.0155

Table 11.9: Test relative errors on 2d Darcy Flow. Our reproduction of the FNO-2d results outperform those reported by the authors [Li et al., 2021c]. Nonetheless, our proposed XD operations outperform both our reproduction and the reported results at every resolution.

Method (source)	s = 85	s = 106	s = 141	s = 211	s = 421
NN [Li et al., 2021c]	0.1716	-	0.1716	0.1716	0.1716
GCN [Li et al., 2021c]	0.0253	-	0.0493	0.0727	0.1097
FCN [Li et al., 2021c]	0.0299	-	0.0298	0.0298	0.0299
PCANN [Li et al., 2021c]	0.0244	-	0.0251	0.0255	0.0259
GNO [Li et al., 2021c]	0.0346	-	0.0332	0.0342	0.0369
LNO [Li et al., 2021c]	0.0520	-	0.0461	0.0445	-
MGNO [Li et al., 2021c]	0.0416	-	0.0428	0.0428	0.0420
FNO-2d [Li et al., 2021c]	0.0108	-	0.0109	0.0109	0.0098
CNN (ours)	0.0404	0.0495	0.0613	0.0813	0.1150
FNO-2d (reproduced)	0.0096	0.0092	0.0091	0.0091	0.0091
CNN XD (ours)	**0.0065**	**0.0065**	**0.0065**	**0.0071**	**0.0066**
FNO-2d XD (ours)	0.0082	0.0079	0.0077	0.0076	0.0074

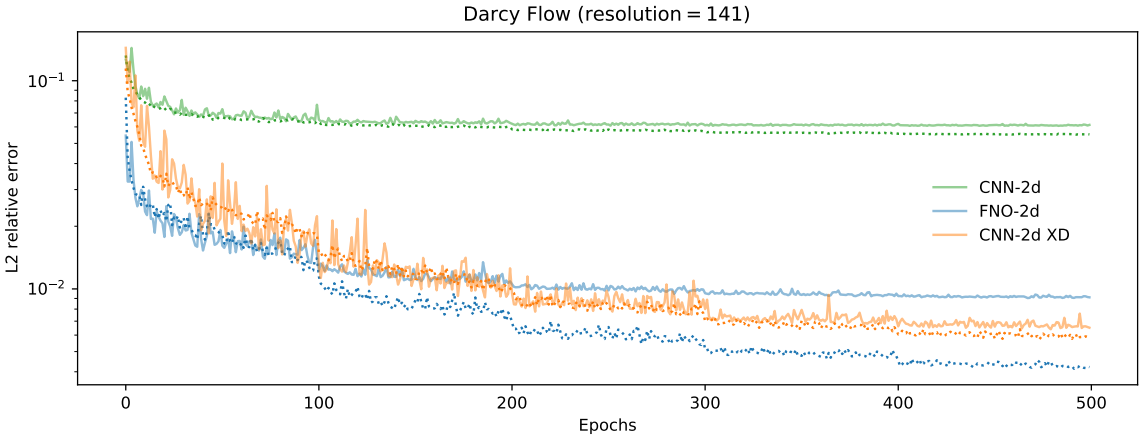


Figure 11.5: Training curves (dotted) and test curves (solid) on Darcy Flow at resolution 141, showing better generalization of XD-operations.

11.E Experimental Details: Protein Folding

For our protein folding experiments, our code is a PyTorch re-implementation of the PDNET code and setup [Adhikari, 2020a] provided here: `https://github.com/ba-lab/pdnet`. As before, we use the same training routine and settings as the Dilated ResNet architecture

Table 11.10: Architecture optimizer settings on for our protein folding experiments, across different ResNet depths. Note that the same step-size is used throughout since the backbone has no step-size schedule.

search space	optimizer	step-size	warmup epochs
ResNet-4 XD	Adam	1E-4	2
ResNet-6 XD	Momentum(0.99)	1E-4	2
ResNet-10 XD	Momentum(0.99)	1E-3	2
ResNet-18 XD	Momentum(0.9)	5E-4	2
ResNet-34 XD	Momentum(0.9)	5E-4	2

Table 11.11: Test MAE_8 of the Dilated ResNet of [Adhikari, 2020a], compared to a standard ResNet backbone and XD-operations applied to ResNet. Results are averaged over 3 trials.

Method	depth $= 4$	depth $= 6$	depth $= 10$	depth $= 18$	depth $= 34$
ResNet	5.99 ± 0.43	5.30 ± 0.11	4.91 ± 0.25	4.80 ± 0.07	4.66 ± 0.15
Dilated ResNet	6.04 ± 0.33	5.49 ± 0.02	4.64 ± 0.08	4.59 ± 0.22	4.50 ± 0.13
ResNet XD	$\mathbf{5.59 \pm 0.09}$	$\mathbf{4.59 \pm 0.17}$	$\mathbf{4.25 \pm 0.16}$	$\mathbf{4.22 \pm 0.03}$	$\mathbf{4.00 \pm 0.07}$

and only tune the architecture optimizer. We consider the following hyperparameters for the architecture optimizer: Adam vs. SGD (with or without momentum), learning rate, and number of warmup epochs. The final hyperparameters for each depth can be found in Table 11.10. Our ResNet backbone differs from Dilated ResNet in that its dilation rate is set to 1 in every convolutional layer. In Table 11.11, we present average MAE_8 on the PSICOV test set for each method at each depth.

11.F Experimental Details: Music Modeling and Sequence Modeling

For our sequence modeling experiments we use the TCN code [Bai et al., 2018] provided here: `https://github.com/locuslab/TCN`. As before we use the same settings and training routine as the backbone for all tasks, tuning only the architecture optimizer. The specific settings are provided in Table 11.12. For both the baselines and XD-operations we use the same optimizer settings for both the dilated and undilated TCN backbones. In Table 11.13 we present results for both music modeling and for two additional benchmarks—permuted

Table 11.12: Architecture optimizer settings on sequence modeling tasks. Note that the step-size is updated using the same schedule as the backbone.

task	optimizer	initial step-size	warmup epochs
Permuted MNIST	Adam	2E-4	0
JSB Chorales	Adam	2E-4	25
Nottingham	Adam	2E-3	0
Penn Treebank	Adam	2E-6	0

Table 11.13: XD-operations applied to TCNs compared to recent empirical results in sequence modeling. Our results are averages of three trials. Methods achieving within one deviation of the best performance are bolded.

Method (source)	Permuted MNIST* (error)	JSB Chorales (loss)	Nottingham (loss)	Penn Treebank (perplexity)
LSTM [Bai et al., 2018]	14.3	8.45	3.29	78.93
GRU [Bai et al., 2018]	12.7	8.43	3.46	92.48
RNN [Bai et al., 2018]	74.7	8.91	4.05	114.50
TCN backbone [Bai et al., 2018]	2.8	8.10	3.07	88.68
TrellisNet [Bai et al., 2019]	1.87	-	-	**54.19**
R-Transformer [Wang et al., 2020b]	-	-	**2.37**	84.38
HiPPO-LegS [Gu et al., 2020]	**1.7**	-	-	-
TCN backbone (reproduced)	2.89 ± 0.04	8.17 ± 0.01	2.97 ± 0.01	88.49 ± 0.31
TCN backbone XD (ours)	$\mathbf{1.75 \pm 0.11}$	$\mathbf{8.07 \pm 0.02}$	2.81 ± 0.05	84.11 ± 0.25
Undilated TCN (ours)	11.3 ± 2.1	8.16 ± 0.04	3.21 ± 0.02	94.30 ± 0.33
Undilated TCN XD (ours)	$\mathbf{1.77 \pm 0.10}$	$\mathbf{8.07 \pm 0.01}$	2.84 ± 0.02	85.04 ± 0.49

* We use depth $\mathrm{d} = (3,3,3)$ XD-operations for permuted MNIST experiments; elsewhere we use $(1,3,1)$. Results within a standard deviation of the best are bolded.

MNIST and Penn Treebank—on which we see a similar pattern of XD-operations being able to recover and even beat (dilated) TCN performance starting from an undilated network.

Chapter 12

Conclusion

This dissertation has pursued a science of foundation models—an effort to replace ad-hoc recipes and rules of thumb with systematic understanding of how these models should be designed, trained, and adapted. Across three parts, we have developed scaling laws that account for test-time compute and skill-dependent behavior, data selection and adaptation techniques that dramatically reduce data requirements, and benchmarks and neural primitives that bring machine learning to diverse scientific domains. In this closing chapter, we summarize the contributions of each part (§12.1) and offer closing remarks (§12.2).

12.1 Summary of Contributions

Part I: Scaling Recipes for Foundation Models. In Chapter 2, we introduced Train-to-Test (T^2) scaling laws that jointly optimize pretraining and inference-time compute. Our key finding is that when test-time scaling via repeated sampling is accounted for, the optimal pretraining strategy shifts dramatically into the overtraining regime: models should be substantially smaller and more overtrained than what Chinchilla prescribes. We validated this prediction by training models in the forecasted optimal region and confirming their superior performance, and showed that these findings persist after post-training.

In Chapter 3, we demonstrated that compute-optimal allocations are skill-dependent. Through experiments across nine compute scales and nineteen benchmarks, we found that knowledge-based tasks are capacity-hungry—favoring larger models—while code-based reasoning tasks are data-hungry—favoring more training tokens and smaller models.

We further showed that these differences are fundamental and not merely artifacts of data composition, with implications for how the proportion of skill-relevant data in the pretraining mix influences optimal model size.

In Chapter 4, we presented Manticore, a system that automates the design of pretrained hybrid architectures by combining components from different model families. By learning projectors between pretrained model representations and applying differentiable architecture search, Manticore constructs hybrids that are competitive with hand-designed architectures while reusing pretrained weights rather than training from scratch.

Part II: Data Efficient Learning. In Chapter 5, we formalized the notion of skills for language models and introduced Skill-it, an online data selection algorithm that exploits prerequisite dependencies between skills. Skill-it achieves up to 3× reduction in data requirements for continual pretraining by dynamically reweighting data sources based on the model's current proficiency at each skill.

In Chapter 6, we presented Loki, a training-free method for adapting pretrained models to predict classes they have never observed. By computing Fréchet means over metric spaces that relate observed and unobserved labels, Loki enables zero-shot prediction with theoretical guarantees on class coverage and practical improvements of up to 19.5% on vision-language models.

In Chapter 7, we introduced AutoWS-Bench-101, a benchmark for evaluating automated weak supervision across ten diverse datasets spanning images, text, time series, and tabular data. Our central finding is that automated WS methods must incorporate signal from foundation models if they are to outperform simple few-shot baselines, pointing to the need for tighter integration between these approaches.

Part III: Adaptation to Scientific Domains. In Chapter 8, we presented the AUP score—the Area Under the Performance Profile—a principled method for comparing methods across tasks with heterogeneous metrics. The AUP score, introduced in the AutoML Decathlon and since adopted in multiple competitions and benchmarks, avoids the pitfalls of naive score or rank averaging while providing a single number for comparison.

In Chapter 9, we described the AutoML Decathlon, a competition that evaluated AutoML methods on twenty diverse tasks spanning PDEs, audio, biological signals, and more. The results demonstrated that combining modern ML techniques—transfer learning, differentiable

NAS, and advanced hyperparameter optimization—is a promising direction for AutoML on diverse tasks.

In Chapter 10, we presented NAS-Bench-360, a benchmark of ten tasks for evaluating neural architecture search beyond standard image classification. We showed that state-of-the-art NAS methods perform inconsistently across diverse tasks, with many catastrophic failures, establishing the need for more robust evaluation.

In Chapter 11, we introduced XD-operations, a search space of neural operations that generalizes convolutions through learnable efficient linear transforms. XD-operations discover novel domain-tailored operations that outperform hand-designed architectures on PDE solving, protein folding, and music modeling, demonstrating that automated operation design can surpass expert knowledge in scientific domains.

12.2 Closing Remarks

The development of foundation models today resembles an earlier era of chemistry, when practitioners relied on recipes and intuition rather than systematic understanding. The transition from alchemy to chemistry required developing the right abstractions, measurement tools, and experimental methodology. This dissertation has contributed to an analogous transition for foundation models: from scaling laws that account for how models will actually be used at inference time, to data selection methods grounded in the structure of what models need to learn, to benchmarks and neural primitives designed for the full diversity of scientific tasks.

A unifying theme across this work is that *one size does not fit all.* Optimal compute allocation depends on what skills you are targeting. Optimal data selection depends on what the model has already learned. Optimal architecture depends on the domain. And optimal evaluation depends on the heterogeneity of the tasks at hand. Recognizing and exploiting this structure—rather than relying on uniform recipes—is the key to building foundation models that are both more efficient and more capable.

As foundation models increasingly serve as the substrate for scientific discovery, the need for principled understanding of their behavior will only grow. The tools developed in this dissertation—fine-grained scaling laws, skills-based data selection, geometric adaptation, and diverse-task benchmarks—provide a foundation for this endeavor. Our hope is that

by making these methods accessible and systematic, we can help bring the transformative potential of foundation models to the scientific domains where they are needed most.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Amro Abbas, Kushal Tirumala, Dániel Simig, Surya Ganguli, and Ari S. Morcos. Semdedup: Data-efficient learning at web-scale through semantic deduplication, 2023.

Mohamed S. Abdelfattah, Abhinav Mehrotra, Lukasz Dudziak, and Nicholas D. Lane. Zero-cost proxies for lightweight NAS. In *Procedings of the 9th International Conference on Learning Representations*, 2021.

Badri Adhikari. A fully open-source framework for deep learning protein real-valued distances. *Scientific Reports*, 10(1):13374, 2020a. doi: 10.1038/s41598-020-70181-0. URL `https://doi.org/10.1038/s41598-020-70181-0`.

Badri Adhikari. Deepcon: protein contact prediction using dilated convolutional neural networks with dropout. *Bioinformatics*, 36(2):470–477, 2020b.

Badri Adhikari. A fully open-source framework for deep learning protein real-valued distances. *Scientific reports*, 10(1):1–10, 2020c.

Nir Ailon, Omer Leibovich, and Vineet Nair. Sparse linear networks with a fixed butterfly structure: Theory and practice. arXiv, 2020.

Takuya Akiba, Makoto Shing, Yujin Tang, Qi Sun, and David Ha. Evolutionary optimization of model merging recipes, 2024.

Túlio C. Alberto, Johannes V. Lochter, and Tiago A. Almeida. Tubespam: Comment spam filtering on youtube. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 138–143, 2015. doi: 10.1109/ICMLA.2015.37.

Keivan Alizadeh vahid, Anish Prabhu, Ali Farhadi, and Mohammad Rastegari. Butterfly transform: An efficient FFT based neural architecture design. In *Proceedings of the IEEE Conference on Conference on Computer Vision and Pattern Recognition*, 2020.

Moray Allan and Christopher Williams. Harmonising chorales by probabilistic inference. In *Advances in Neural Information Processing Systems*, 2005.

Ido Amos, Jonathan Berant, and Ankit Gupta. Never train from scratch: Fair comparison of long-sequence models requires data-driven priors. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=PdaPky8MUn.

Hyrum S. Anderson and Phil Roth. Ember: An open dataset for training static pe malware machine learning models, 2018a. URL https://arxiv.org/abs/1804.04637.

Hyrum S. Anderson and Phil Roth. EMBER: an open dataset for training static PE malware machine learning models. *CoRR*, abs/1804.04637, 2018b. URL http://arxiv.org/abs/1804.04637.

Sanjeev Arora and Anirudh Goyal. A theory for emergence of complex skills in language models. *ArXiv*, abs/2307.15936, 2023. URL https://api.semanticscholar.org/CorpusID:260334352.

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, Dylan Zinsley, James Zou, Atri Rudra, and Christopher Ré. Simple linear attention language models balance the recall-throughput tradeoff. *arXiv:2402.18668*, 2024.

Ben Athiwaratkun, Sanjay Krishna Gouda, Zijian Wang, Xiaopeng Li, Yuchen Tian, Ming Tan, Wasi Uddin Ahmad, Shiqi Wang, Qing Sun, Mingyue Shang, Sujan Kumar Gonugondla, Hantian Ding, Varun Kumar, Nathan Fulton, Arash Farahani, Siddhartha Jain, Robert Giaquinto, Haifeng Qian, Murali Krishna Ramanathan, Ramesh Nallapati, Baishakhi Ray, Parminder Bhatia, Sudipta Sengupta, Dan Roth, and Bing Xiang. Multilingual evaluation of code generation models. 2022. doi: 10.48550/ARXIV.2210.14868. URL https://arxiv.org/abs/2210.14868.

Manfredo Atzori, Arjan Gijsberts, Simone Heynen, Anne-Gabrielle Mittaz Hager, Olivier Deriaz, Patrick Van Der Smagt, Claudio Castellini, Barbara Caputo, and Henning Müller. Building the ninapro database: A resource for the biorobotics community. In *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, pages 1258–1265. IEEE, 2012a.

Manfredo Atzori, Arjan Gijsberts, Simone Heynen, Anne-Gabrielle Mittaz Hager, Olivier Deriaz, Patrick van der Smagt, Claudio Castellini, Barbara Caputo, and Henning Müller. Building the ninapro database: A resource for the biorobotics community. In *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, pages 1258–1265, 2012b. doi: 10.1109/BioRob.2012.6290287.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Stephen H Bach, Daniel Rodriguez, Yintao Liu, Chong Luo, Haidong Shao, Cassandra Xia, Souvik Sen, Alex Ratner, Braden Hancock, Houman Alborzi, et al. Snorkel drybell: A case study in deploying weak supervision at industrial scale. In *Proceedings of the 2019 International Conference on Management of Data*, pages 362–375, 2019.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv, 2018.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Trellis networks for sequence modeling. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

Yuntao Bai, Andy Jones, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback, 2022.

Gökhan Bakir, Thomas Hofmann, Bernhard Schölkopf, Alexander J. Smola, Ben Taskar, and S.V.N Vishwanathan. *Predicting Structured Data*. The MIT Press, July 2007. ISBN 978-0-262-25569-1. doi: 10.7551/mitpress/7443.001.0001. URL `https://doi.org/10.7551/mitpress/7443.001.0001`.

Björn Barz and Joachim Denzler. Hierarchy-based image embeddings for semantic image retrieval. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 638–647. IEEE, 2019.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

Akshita Bhagia, Jiacheng Liu, Alexander Wettig, David Heineman, Oyvind Tafjord, Ananya Harsh Jha, Luca Soldaini, Noah A. Smith, Dirk Groeneveld, Pang Wei Koh, Jesse Dodge, and Hannaneh Hajishirzi. Establishing task scaling laws via compute-efficient model ladders, 2024a. URL https://arxiv.org/abs/2412.04403.

Akshita Bhagia, Jiacheng Liu, Alexander Wettig, David Heineman, Oyvind Tafjord, Ananya Harsh Jha, Luca Soldaini, Noah A Smith, Dirk Groeneveld, Pang Wei Koh, et al. Establishing task scaling laws via compute-efficient model ladders. *arXiv preprint arXiv:2412.04403*, 2024b.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. Pythia: A suite for analyzing large language models across training and scaling, 2023.

Louis J. Billera, Susan P. Holmes, and Karen Vogtmann. Geometry of the space of phylogenetic trees. *Advances in Applied Mathematics*, 27(4):733–767, 2001. ISSN 0196-8858. doi: https://doi.org/10.1006/aama.2001.0759. URL https://www.sciencedirect.com/science/article/pii/S0196885801907596.

Sid Black, Leo Gao, Phil Wang, Connor Leahy, and Stella Biderman. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow, March 2021. URL https://doi.org/10.5281/zenodo.5297715.

Benedikt Boecking, Willie Neiswanger, Eric Xing, and Artur Dubrawski. Interactive weak supervision: Learning useful heuristics for data labeling. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=IDFQI9OY6K.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, et al. On the opportunities and risks of foundation models. https://arxiv.org/abs/2108.07258, 2021a.

Rishi Bommasani, Percy Liang, et al. On the opportunities and risks of foundation models, 2021b.

Aleksandar Botev, Soham De, Samuel L Smith, Anushan Fernando, George-Cristian Muraru, Ruba Haroun, Leonard Berrada, Razvan Pascanu, Pier Giuseppe Sessa, Robert Dadashi, Léonard Hussenot, Johan Ferret, Sertan Girgin, Olivier Bachem, Alek Andreev, Kathleen Kenealy, Thomas Mesnard, Cassidy Hardin, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Armand Joulin, Noah Fiedel, Evan Senter, Yutian Chen, Srivatsan Srinivasan, Guillaume Desjardins, David Budden, Arnaud Doucet, Sharad Vikram, Adam Paszke, Trevor Gale, Sebastian Borgeaud, Charlie Chen, Andy Brock, Antonia Paterson, Jenny Brennan, Meg Risdal, Raj Gundluru, Nesh Devanathan, Paul Mooney, Nilay Chauhan, Phil Culliton, Luiz GUStavo Martins, Elisa Bandy, David Huntsperger, Glenn Cameron, Arthur Zucker, Tris Warkentin, Ludovic Peran, Minh Giang, Zoubin Ghahramani, Clément Farabet, Koray Kavukcuoglu, Demis Hassabis, Raia Hadsell, Yee Whye Teh, and Nando de Frietas. Recurrentgemma: Moving past transformers for efficient open language models, 2024.

Bradley Brown, Jordan Juravsky, Ryan Saul Ehrlich, Ronald Clark, Quoc V Le, Christopher Re, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling, 2025. URL `https://openreview.net/forum?id=0xUEBQV54B`.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020a.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020b.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. Multipl-e: A scalable and polyglot approach to benchmarking neural code generation. *IEEE Trans. Softw. Eng.*, 49(7):3675–3691, July 2023. ISSN 0098-5589. doi: 10.1109/TSE.2023.3267446. URL https://doi.org/10.1109/TSE.2023.3267446.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. 2021a.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021b.

Mayee Chen, Nicholas Roberts, Kush Bhatia, Jue Wang, Ce Zhang, Frederic Sala, and Christopher Ré. Skill-it! a data-driven skills framework for understanding and training language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 36000–36040. Curran Associates, Inc., 2023a. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/70b8505ac79e3e131756f793cd80eb8d-Paper-Conference.pdf.

Mayee F Chen, Nicholas Roberts, Kush Bhatia, Jue WANG, Ce Zhang, Frederic Sala, and Christopher Re. Skill-it! a data-driven skills framework for understanding and training language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b. URL https://openreview.net/forum?id=IoizwO1NLf.

Mayee F Chen, Michael Y. Hu, Nicholas Lourie, Kyunghyun Cho, and Christopher Re. Aioli: A unified optimization framework for language model data mixing. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=sZGZJhaNSe.

M.F. Chen, D. Fu, D. Adila, M. Zhang, F. Sala, K. Fatahalian, and C. Ré. Shoring up the foundations: Fusing model embeddings and weak supervision. In *Uncertainty in Artificial Intelligence (UAI)*, 2022.

Tianqi Chen and Carlos Guestrin. XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, aug 2016a. doi: 10.1145/2939672.2939785. URL https://doi.org/10.1145%2F2939672.2939785.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016b. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939785. URL https://doi.org/10.1145/2939672.2939785.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/chen20j.html.

Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four {gpu} hours: A theoretically inspired perspective. In *International Conference on Learning Representations*, 2021c. URL https://openreview.net/forum?id=Cnon5ezMHtu.

Yeshwanth Cherapanamjeri, Nicolas Flammarion, and Peter L. Bartlett. Fast mean estimation with sub-gaussian rates. In Alina Beygelzimer and Daniel Hsu, editors, *Proceedings of the Thirty-Second Conference on Learning Theory*, volume 99 of *Proceedings of Machine Learning Research*, pages 786–806. PMLR, 25–28 Jun 2019. URL https://proceedings.mlr.press/v99/cherapanamjeri19b.html.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023. URL https://lmsys.org/blog/2023-03-30-vicuna/.

Carlo Ciliberto, Lorenzo Rosasco, and Alessandro Rudi. A consistent regularization approach for structured prediction. In *Advances in Neural Information Processing Systems 30 (NIPS 2016)*, volume 30, 2016.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018.

Gari D Clifford, Chengyu Liu, Benjamin Moody, Li-wei H. Lehman, Ikaro Silva, Qiao Li, A E Johnson, and Roger G. Mark. Af classification from a short single lead ecg recording: The physionet/computing in cardiology challenge 2017. In *2017 Computing in Cardiology (CinC)*, pages 1–4, 2017a. doi: 10.22489/CinC.2017.065-469.

Gari D Clifford, Chengyu Liu, Benjamin Moody, Li-wei H. Lehman, Ikaro Silva, Qiao Li, A E Johnson, and Roger G. Mark. Af classification from a short single lead ecg recording: The physionet/computing in cardiology challenge 2017. In *2017 Computing in Cardiology (CinC)*, pages 1–4, 2017b. doi: 10.22489/CinC.2017.065-469.

Gari D Clifford, Chengyu Liu, Benjamin Moody, H Lehman Li-wei, Ikaro Silva, Qiao Li, AE Johnson, and Roger G Mark. Af classification from a short single lead ecg recording: The physionet/computing in cardiology challenge 2017. In *2017 Computing in Cardiology (CinC)*, pages 1–4. IEEE, 2017c.

Taco S. Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical cnns. In *International Conference on Learning Representations*, 2018a.

Taco S. Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical CNNs. In *International Conference on Learning Representations (ICLR)*, 2018b. URL `https://openreview.net/forum?id=Hkbd5xZRb`.

Cody Coleman, Christopher Yeh, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. Selection via proxy: Efficient data selection for deep learning, 2019.

ENCODE Project Consortium et al. The encode (encyclopedia of dna elements) project. *Science*, 306(5696):636–640, 2004.

Ulysse Côté-Allard, Cheikh Latyr Fall, Alexandre Drouin, Alexandre Campeau-Lecours, Clément Gosselin, Kyrre Glette, François Laviolette, and Benoit Gosselin. Deep learning for electromyographic hand gesture signal classification using transfer learning. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 27(4): 760–771, 2019.

Ulysse Côté-Allard, Cheikh Latyr Fall, Alexandre Drouin, Alexandre Campeau-Lecours, Clément Gosselin, Kyrre Glette, François Laviolette, and Benoit Gosselin. Deep learning for electromyographic hand gesture signal classification using transfer learning. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 27(4): 760–771, 2019. doi: 10.1109/TNSRE.2019.2896269.

George Dahl, Frank Schneider, Zachary Nado, Naman Agarwal, Chandramouli Shama Sastry, Philipp Hennig, Sourabh Medapati, Runa Eschenhagen, Priya Kasimbeg, Daniel Suo, Juhan Bae, Justin Gilmer, Abel Peirson, Bilal Khan, Rohan Anil, Mike Rabbat, Shankar Krishnan, Daniel Snider, Ehsan Amid, and Peter Mattson. Benchmarking neural network training algorithms, 06 2023.

Tri Dao, Albert Gu, Matthew Eichhorn, Atri Rudra, and Christopher Ré. Learning fast algorithms for linear transforms using butterfly factorizations. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Tri Dao, Nimit Sohoni, Albert Gu, Matthew Eichhorn, Amit Blonder, Megan Leszczynski, Atri Rudra, and Christopher Ré. Kaleidoscope: An efficient, learnable representation for all structured linear maps. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

Nilaksh Das, Sanya Chaba, Renzhi Wu, Sakshi Gandhi, Duen Horng Chau, and Xu Chu. Goggles: Automatic image labeling with affinity coding. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 1717–1732, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367356. doi: 10.1145/3318464.3380592. URL https://doi.org/10.1145/3318464.3380592.

Mohammad-Javad Davari and Eugene Belilovsky. Model breadcrumbs: Scaling multi-task model merging with sparse masks. *ArXiv*, abs/2312.06795, 2023. URL https://api.semanticscholar.org/CorpusID:266174505.

Alexander Philip Dawid and Allan M Skene. Maximum likelihood estimation of observer error-rates using the EM algorithm. *Applied statistics*, pages 20–28, 1979.

Soham De, Samuel L. Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, Guillaume Desjardins, Arnaud Doucet, David Budden, Yee Whye Teh, Razvan Pascanu, Nando De Freitas, and Caglar Gulcehre. Griffin: Mixing gated linear recurrences with local attention for efficient language models, 2024.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de las Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, Seb Noury, Federico Pesamosca, David Pfau, Olivier Sauter, Cristian Sommariva, Stefano Coda, Basil Duval, Ambrogio Fasoli, Pushmeet Kohli, Koray Kavukcuoglu, Demis Hassabis, and Martin Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022. doi: 10.1038/s41586-021-04301-9. URL https://doi.org/10.1038/s41586-021-04301-9.

Angus Dempster, François Petitjean, and Geoffrey I Webb. Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. doi: 10.1109/MSP.2012.2211477.

Yangruibo Ding, Zijian Wang, Wasi Uddin Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, and Bing Xiang. Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=wgDcbBMSfh.

Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002a.

Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, January 2002b. ISSN 1436-4646. doi: 10.1007/s101070100263. URL https://arxiv.org/abs/cs/0102001.

Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002c. doi: 10.1007/s101070100263. URL https://doi.org/10.1007/s101070100263.

Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. 2001. doi: 10.48550/ARXIV.CS/0102001. URL https://arxiv.org/abs/cs/0102001.

Xuanyi Dong and Yezhou Yang. One-shot neural architecture search via self-evaluated template network. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3680–3689, 2019a.

Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four GPU hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019b.

Xuanyi Dong and Yi Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

Yawen Duan, Xin Chen, Hang Xu, Zewei Chen, Xiaodan Liang, Tong Zhang, and Zhenguo Li. TransNAS-Bench-101: Improving transferability and generalizability of cross-task neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2021a.

Yawen Duan, Xin Chen, Hang Xu, Zewei Chen, Li Xiaodan, Tong Zhang, and Zhenguo Li. Transnas-bench-101: Improving transferability and generalizability of cross-task neural architecture search. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5247–5256, 2021b.

Jared A Dunnmon, Alexander J Ratner, Khaled Saab, Nishith Khandwala, Matthew Markert, Hersh Sagreiya, Roger Goldman, Christopher Lee-Messer, Matthew P Lungren, Daniel L Rubin, and Christopher Ré. Cross-modal data programming enables rapid medical machine learning. *Patterns (N Y)*, 1(2), May 2020.

Ryan Ehrlich, Bradley Brown, Jordan Juravsky, Ronald Clark, Christopher Ré, and Azalia Mirhoseini. Codemonkeys: Scaling test-time compute for software engineering. *arXiv preprint arXiv:2501.14723*, 2025.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. Autogluon-tabular: Robust and accurate automl for structured data, 2020. URL `https://arxiv.org/abs/2003.06505`.

ENCODE Project Consortium et al. 306(5696):636–640, 2004.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pages 1437–1446. PMLR, 2018.

Angela Fan, Yacine Jernite, Ethan Perez, David Grangier, Jason Weston, and Michael Auli. ELI5: Long form question answering. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3558–3567, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1346. URL `https://aclanthology.org/P19-1346`.

Jiemin Fang, Yuzhu Sun, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. Densely connected search space for more flexible neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

Ali Farhadi, Ian Endres, Derek Hoiem, and David Forsyth. Describing objects by their attributes. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1778–1785, 2009. doi: 10.1109/CVPR.2009.5206772.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis, and Pushmeet Kohli. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022. doi: 10.1038/s41586-022-05172-4. URL `https://doi.org/10.1038/s41586-022-05172-4`.

Eduardo Fonseca, Jordi Pons, Xavier Favory, Frederic Font, Dmitry Bogdanov, Andres Ferraro, Sergio Oramas, Alastair Porter, and Xavier Serra. Freesound datasets: A platform for the creation of open audio datasets. In Sally Jo Cunningham, Zhiyao Duan, Xiao Hu, and Douglas Turnbull, editors, *Proceedings of the 18th International Society for Music Information Retrieval Conference, ISMIR 2017, Suzhou, China, October 23-27, 2017*, pages 486–493, 2017a. URL `https://ismir2017.smcnus.org/wp-content/uploads/2017/10/161_Paper.pdf`.

Eduardo Fonseca, Jordi Pons Puig, Xavier Favory, Frederic Font Corbera, Dmitry Bogdanov, Andres Ferraro, Sergio Oramas, Alastair Porter, and Xavier Serra. Freesound datasets: a platform for the creation of open audio datasets. In *Hu X, Cunningham SJ, Turnbull D, Duan Z, editors. Proceedings of the 18th ISMIR Conference; 2017 oct 23-27; Suzhou, China.[Canada]: International Society for Music Information Retrieval; 2017. p. 486-93.* International Society for Music Information Retrieval (ISMIR), 2017b.

Eduardo Fonseca, Xavier Favory, Jordi Pons, Frederic Font, and Xavier Serra. Fsd50k: an open dataset of human-labeled sound events. *arXiv preprint arXiv:2010.00475*, 2020.

Maurice R. Fréchet. Les éléments aléatoires de nature quelconque dans un espace distancié. *Annales de l'institut Henri Poincaré*, 1948.

Jason A. Fries, Paroma Varma, Vincent S. Chen, Ke Xiao, Heliodoro Tejeda, Priyanka Saha, Jared Dunnmon, Henry Chubb, Shiraz Maskatia, Madalina Fiterau, Scott Delp, Euan Ashley, Christopher Ré, and James R. Priest. Weakly supervised classification of aortic valve malformations using unlabeled cardiac mri sequences. *Nature Communications*, 10(1):3111, 2019. doi: 10.1038/s41467-019-11012-3. URL `https://doi.org/10.1038/s41467-019-11012-3`.

Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Marc' Aurelio Ranzato, and Tomas Mikolov. Devise: A deep visual-semantic embedding model. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper/2013/file/7cce53cf90577442771720a370c3c723-Paper.pdf.

Daniel Y. Fu, Mayee F. Chen, Frederic Sala, Sarah M. Hooper, Kayvon Fatahalian, and Christopher Ré. Fast and three-rious: Speeding up weak supervision with triplet methods. In *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*, 2020.

Daniel Y. Fu, Tri Dao, Khaled K. Saab, Armin W. Thomas, Atri Rudra, and Christopher Ré. Hungry Hungry Hippos: Towards language modeling with state space models. In *International Conference on Learning Representations (ICLR)*, 2023.

Robert M Gagne. The acquisition of knowledge. *Psychological review*, 69(4):355, 1962.

Robert M Gagne and Noel E Paradise. Abilities and learning sets in knowledge acquisition. *Psychological Monographs: General and Applied*, 75(14):1, 1961.

Sainyam Galhotra, Behzad Golshan, and Wang-Chiew Tan. Adaptive rule discovery for labeling text data. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 2217–2225, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383431. doi: 10.1145/3448016.3457334. URL https://doi.org/10.1145/3448016.3457334.

Junyu Gao, Tianzhu Zhang, and Changsheng Xu. I Know the Relationships: Zero-Shot Action Recognition via Two-Stream Graph Convolutional Networks and Knowledge Graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):8303–8311, July 2019. doi: 10.1609/aaai.v33i01.33018303. URL https://ojs.aaai.org/index.php/AAAI/article/view/4843.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2020a. URL https://arxiv.org/abs/2101.00027.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020b.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, September 2021a. URL `https://doi.org/10.5281/zenodo.5371628`.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. SimCSE: Simple contrastive learning of sentence embeddings. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2021b.

John S Garofolo. Timit acoustic phonetic continuous speech corpus. *Linguistic Data Consortium, 1993*, 1993.

Jort F Gemmeke, Daniel PW Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 776–780. IEEE, 2017.

Xinyang Geng, Arnav Gudibande, Hao Liu, Eric Wallace, Pieter Abbeel, Sergey Levine, and Dawn Song. Koala: A dialogue model for academic research. Blog post, April 2023. URL `https://bair.berkeley.edu/blog/2023/04/03/koala/`.

Amirata Ghorbani and James Zou. Data shapley: Equitable valuation of data for machine learning. In *International Conference on Machine Learning*, pages 2242–2251. PMLR, 2019.

Charles Goddard, Shamane Siriwardhana, Malikeh Ehghaghi, Luke Meyers, Vlad Karpukhin, Brian Benedict, Mark McQuade, and Jacob Solawetz. Arcee's mergekit: A toolkit for merging large language models, 2024.

Google. Palm2 technical report. Technical report, 2023. URL `https://ai.google/static/documents/palm2techreport.pdf`.

Sachin Goyal, Pratyush Maini, Zachary C Lipton, Aditi Raghunathan, and J Zico Kolter. Scaling laws for data filtering–data curation cannot be compute agnostic. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22702–22711, 2024.

Colin Graber and Alexander Schwing. Graph structured prediction energy networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/ea6979872125d5acbac6068f186a0359-Paper.pdf`.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, et al. Bootstrap your own latent a new approach to self-supervised learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Dirk Groeneveld, Iz Beltagy, Evan Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, et al. Olmo: Accelerating the science of language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15789–15809, 2024.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. HiPPO: Recurrent memory with optimal polynomial projections. In *Advances in Neural Information Processing Systems*, 2020.

Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=uYLFoz1vlAC`.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1321–1330. PMLR, 06–11 Aug 2017. URL `https://proceedings.mlr.press/v70/guo17a.html`.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Anupam Gupta. Advanced algorithms: Notes for cmu 15-850 (fall 2020), 2020.

Sonal Gupta and Christopher Manning. Improved pattern learning for bootstrapped entity extraction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 98–108, 2014.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020. doi: 10.18653/v1/2020.acl-main.740. URL http://dx.doi.org/10.18653/v1/2020.acl-main.740.

Kelvin Guu, Albert Webson, Ellie Pavlick, Lucas Dixon, Ian Tenney, and Tolga Bolukbasi. Simfluence: Modeling the influence of individual training examples by simulating training runs, 2023.

Isabelle Guyon, Lisheng Sun-Hosoya, Marc Boullé, Hugo Jair Escalante, Sergio Escalera, Zhengying Liu, Damir Jajetic, Bisakha Ray, Mehreen Saeed, Michèle Sebag, Alexander Statnikov, Wei-Wei Tu, and Evelyne Viegas. *Analysis of the AutoML Challenge Series 2015–2018*, pages 177–219. Springer International Publishing, Cham, 2019. ISBN 978-3-030-05318-5. doi: 10.1007/978-3-030-05318-5_10. URL https://doi.org/10.1007/978-3-030-05318-5_10.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016b. doi: 10.1109/CVPR.2016.90.

Marti A Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics-Volume 2*, pages 539–545. Association for Computational Linguistics, 1992.

William Held, Bhargavi Paranjape, Punit Singh Koura, Mike Lewis, Frank Zhang, and Todor Mihaylov. Optimizing pretraining data mixtures with llm-estimated utility, 2025. URL https://arxiv.org/abs/2501.11747.

Peter Henderson*, Mark S. Krass*, Lucia Zheng, Neel Guha, Christopher D. Manning, Dan Jurafsky, and Daniel E. Ho. Pile of law: Learning responsible data filtering from the law and a 256gb open-source legal dataset, 2022. URL https://arxiv.org/abs/2207.00220.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021a.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021b.

Danny Hernandez, Tom Brown, Tom Conerly, Nova DasSarma, Dawn Drain, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Tom Henighan, Tristan Hume, Scott Johnston, Ben Mann, Chris Olah, Catherine Olsson, Dario Amodei, Nicholas Joseph, Jared Kaplan, and Sam McCandlish. Scaling laws and interpretability of learning from repeated data, 2022.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically, 2017. URL https://arxiv.org/abs/1712.00409.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997a.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997b. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, DDL Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 10, 2022a.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Thomas Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karén Simonyan, Erich Elsen, Oriol Vinyals, Jack Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 30016–30030. Curran Associates, Inc., 2022b. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/c1e2faff6f588870935f114ebe04a3e5-Paper-Conference.pdf.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022c. Curran Associates Inc. ISBN 9781713871088.

Shenda Hong, Yanbo Xu, Alind Khare, Satria Priambada, Kevin Maher, Alaa Aljiffry, Jimeng Sun, and Alexey Tumanov. Holmes: health online model ensemble serving for deep learning models in intensive care units. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1614–1624, 2020.

Sarah Hooper, Michael Wornow, Ying Hang Seah, Peter Kellman, Hui Xue, Frederic Sala, Curtis Langlotz, and Christopher Re. Cut out the annotator, keep the cutout: better segmentation with weak supervision. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=bjkX6Kzb5H`.

Cheng-Yu Hsieh, Jieyu Zhang, and Alexander Ratner. Nemo: Guiding and contextualizing weak supervision for interactive data programming, 2022. URL `https://arxiv.org/abs/2203.01382`.

Daniel Hsu and Sivan Sabato. Loss minimization and parameter estimation with heavy tails. *Journal of Machine Learning Research*, 17(18):1–40, 2016. URL `http://jmlr.org/papers/v17/14-273.html`.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=6t0Kwf8-jrj`.

Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Predicting predictions from training data, 2022.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

Berivan Isik, Natalia Ponomareva, Hussein Hazimeh, Dimitris Paparas, Sergei Vassilvitskii, and Sanmi Koyejo. Scaling laws for downstream task performance of large language models. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2024.

Berivan Isik, Natalia Ponomareva, Hussein Hazimeh, Dimitris Paparas, Sergei Vassilvitskii, and Sanmi Koyejo. Scaling laws for downstream task performance in machine translation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=vPOMTkmSiu.

Zohar Jackson, César Souza, Jason Flaks, Yuxin Pan, Hereman Nicolas, and Adhish Thite. Jakobovski/free-spoken-digit-dataset: v1.0.8, August 2018. URL https://doi.org/10.5281/zenodo.1342401.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, Olivier J Henaff, Matthew Botvinick, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver IO: A general architecture for structured inputs & outputs. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=fILj7WpI-g.

Dong-Hwan Jang, Sangdoo Yun, and Dongyoon Han. Model stock: All we need is just a few fine-tuned models. *ArXiv*, abs/2403.19522, 2024. URL https://api.semanticscholar.org/CorpusID:268733341.

Chao Jia, Yinfei Yang, Ye Xia, Yi-Ting Chen, Zarana Parekh, Hieu Pham, Quoc Le, Yun-Hsuan Sung, Zhen Li, and Tom Duerig. Scaling up visual and vision-language representation learning with noisy text supervision. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 4904–4916. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/jia21b.html.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Matt Gardner Johannes Welbl, Nelson F. Liu. Crowdsourcing multiple choice science questions. 2017.

David T. Jones, Daniel W. A. Buchan, Domenico Cozzetto, and Massimiliano Pontil. PSICOV: precise structural contact prediction using sparse inverse covariance estimation on large multiple sequence alignments. *Bioinformatics*, 28(2):184–190, 11 2011. ISSN 1367-4803. doi: 10.1093/bioinformatics/btr638. URL https://doi.org/10.1093/bioinformatics/btr638.

David Josephs, Carson Drake, Andy Heroy, and John Santerre. semg gesture recognition with a simple model of attention. In *Machine Learning for Health*, pages 126–138. PMLR, 2020.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147/.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Kathryn Tunyasuvunakool, Olaf Ronneberger, Russ Bates, Augustin Žídek, Alex Bridgland, Clemens Meyer, Simon A A Kohl, Anna Potapenko, Andrew J Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Martin Steinegger, Michalina Pacholska, David Silver, Oriol Vinyals, Andrew W Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. High accuracy protein structure prediction using deep learning. *In Fourteenth Critical Assessment of Techniques for Protein Structure Prediction (Abstract Book)*, 2020. URL https://predictioncenter.org/casp14/doc/CASP14_Abstracts.pdf.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021. doi: 10.1038/s41586-021-03819-2. URL https://doi.org/10.1038/s41586-021-03819-2.

Michael C. Kampffmeyer, Yinbo Chen, Xiaodan Liang, Hao Wang, Yujia Zhang, and Eric P. Xing. Rethinking knowledge graph propagation for zero-shot learning. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11479–11488, 2018.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020a.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020b. URL `https://arxiv.org/abs/2001.08361`.

David R Karger, Sewoong Oh, and Devavrat Shah. Iterative learning for reliable crowdsourcing systems. In *Advances in neural information processing systems*, pages 1953–1961, 2011.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention, 2020.

Joshua Kazdan, Rylan Schaeffer, Youssef Allouah, Colin Sullivan, Kyssen Yu, Noam Levi, and Sanmi Koyejo. Efficient prediction of pass@ k scaling in large language models. *arXiv preprint arXiv:2510.05197*, 2025.

Dahyun Kim, Chanjun Park, Sanghoon Kim, Wonsung Lee, Wonho Song, Yunsu Kim, Hyeonwoo Kim, Yungi Kim, Hyeonju Lee, Jihoo Kim, Changbae Ahn, Seonghoon Yang, Sukyung Lee, Hyunbyung Park, Gyoungjin Gim, Mikyoung Cha, Hwalsuk Lee, and Sunghun Kim. Solar 10.7b: Scaling large language models with simple yet effective depth up-scaling, 2023a.

Joongwon Kim, Akari Asai, Gabriel Ilharco, and Hannaneh Hajishirzi. Taskweb: Selecting better source tasks for multi-task nlp, 2023b.

Junkyung Kim, Drew Linsley, Kalpit Thakkar, and Thomas Serre. Disentangling neural mechanisms for perceptual grouping, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations*, 2017a.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017b. URL `https://openreview.net/forum?id=SJU4ayYgl`.

Hannah Rose Kirk, Yennie Jun, Filippo Volpin, Haider Iqbal, Elias Benussi, Frederic Dreyer, Aleksandar Shtedritski, and Yuki Asano. Bias out-of-the-box: An empirical analysis of intersectional occupational biases in popular generative language models. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 2611–2624. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/1531beb762df4029513ebf9295e0d34f-Paper.pdf.

Nikita Kitaev and Dan Klein. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1249. URL https://www.aclweb.org/anthology/P18-1249.

Nikita Kitaev, Steven Cao, and Dan Klein. Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3499–3505, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1340. URL https://www.aclweb.org/anthology/P19-1340.

Nikita Klyuchnikov, Ilya Trofimov, Ekaterina Artemova, Mikhail Salnikov, Maxim Fedorov, and Evgeny Burnaev. NAS-Bench-NLP: Neural architecture search benchmark for natural language processing. arXiv, 2020.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. The stack: 3 tb of permissively licensed source code. *Preprint*, 2022.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions, 2017.

Anna Korba, Alexandre Garcia, and Florence d'Alché-Buc. A structured prediction approach for label ranking. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/b3dd760eb02d2e669c604f6b2f1e803f-Paper.pdf.

Alex Krizhevksy. Learning multiple layers of features from tiny images. Technical report, 2009.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. URL https://api.semanticscholar.org/CorpusID:18268744.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Volodymyr Kuleshov and Percy S Liang. Calibrated structured prediction. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/52d2752b150f9c35ccb6869cbf074e48-Paper.pdf.

Po-Nien Kung, Sheng-Siang Yin, Yi-Cheng Chen, Tse-Hsuan Yang, and Yun-Nung Chen. Efficient multi-task auxiliary learning: Selecting auxiliary data by feature similarity. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 416–428, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.34. URL https://aclanthology.org/2021.emnlp-main.34.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026/.

Christoph H. Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 951–958, 2009. doi: 10.1109/CVPR.2009.5206594.

Christoph H. Lampert, Hannes Nickisch, and Stefan Harmeling. Attribute-based classification for zero-shot visual object categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(3):453–465, 2014. doi: 10.1109/TPAMI.2013.140.

Michael Langberg and Leonard J. Schulman. *Universal approximators for integrals*, pages 598–607. doi: 10.1137/1.9781611973075.50. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611973075.50.

Hugo Laurençon, Lucile Saulnier, et al. The bigscience roots corpus: A 1.6tb composite multilingual dataset, 2023.

Ya Le and Xuan S. Yang. Tiny imagenet visual recognition challenge. 2015.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*. 1999.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. Deduplicating training data makes language models better. *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2022. doi: 10.18653/v1/2022.acl-long.577. URL http://dx.doi.org/10.18653/v1/2022.acl-long.577.

M. Lerasle and R. I. Oliveira. Robust empirical mean estimators, 2011.

David D Lewis. A sequential algorithm for training text classifiers: Corrigendum and additional data. In *Acm Sigir Forum*, volume 29, pages 13–19. ACM New York, NY, USA, 1995.

Juan Li, Ruoxu Wang, Ningyu Zhang, Wen Zhang, Fan Yang, and Huajun Chen. Logic-guided semantic representation learning for zero-shot relation classification. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2967–2978, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.265. URL https://aclanthology.org/2020.coling-main.265.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2019.

Liam Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018a.

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. *arXiv preprint arXiv:1810.05934*, 2018b.

Liam Li, Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Geometry-aware gradient algorithms for neural architecture search. In *Proceedings of the 9th International Conference on Learning Representations*, 2021a.

Liam Li, Mikhail Khodak, Nina Balcan, and Ameet Talwalkar. Geometry-aware gradient algorithms for neural architecture search. In *International Conference on Learning Representations*, 2021b. URL https://openreview.net/forum?id=MuSYkd1hxRP.

Lisha Li, Kevin G Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *ICLR (Poster)*, 2017.

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *Proceedings of the 6th International Conference on Learning Representations*, 2018c.

Yingzhou Li, Haizhao Yang, Eileen R. Martin, Kenneth L. Ho, and Lexing Ying. Butterfly factorization. *Multiscale Modeling & Simulation*, 13(2):714–732, 2015.

Yingzhou Li, Haizhao Yang, and Lexing Ying. Multidimensional butterfly factorization. *Applied and Computational Harmonic Analysis*, 44(3):737–758, 2018d.

Yuchen Li, Yuanzhi Li, and Andrej Risteski. How do transformers learn topic structure: Towards a mechanistic understanding, 2023.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *Proceedings of the 9th International Conference on Learning Representations*, 2021c.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations (ICLR)*, 2021d. URL https://openreview.net/forum?id=c8P9NQVtmnO.

Paul Pu Liang, Chiyu Wu, Louis-Philippe Morency, and Ruslan Salakhutdinov. Towards understanding and mitigating social biases in language models. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 6565–6576. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/liang21a.html.

Drew Linsley, Junkyung Kim, Vijay Veerabadran, Charlie Windolf, and Thomas Serre. Learning long-range spatial dependencies with horizontal gated recurrent units. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 152–164, Red Hook, NY, USA, 2018. Curran Associates Inc.

Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan Yuille, and Li Fei-Fei. Auto-DeepLab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE Conference on Conference on Computer Vision and Pattern Recognition*, 2019a.

Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 82–92, 2019b.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019c.

Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=pPjZIOuQuF.

Zhengying Liu, Adrien Pavao, Zhen Xu, Sergio Escalera, Fabio Ferreira, Isabelle Guyon, Sirui Hong, Frank Hutter, Rongrong Ji, Julio C. S. Jacques Junior, Ge Li, Marius Lindauer, Zhipeng Luo, Meysam Madadi, Thomas Nierhoff, Kangning Niu, Chunguang Pan, Danny Stoll, Sebastien Treguer, Jin Wang, Peng Wang, Chenglin Wu, Youcheng Xiong, Arbër Zela, and Yang Zhang. Winning solutions and post-challenge analyses of the ChaLearn AutoDL challenge 2019. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 17, 2020.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Aaron Lou, Isay Katsman, Qingxuan Jiang, Serge Belongie, Ser-Nam Lim, and Christopher De Sa. Differentiating through the fréchet mean. In *Proceedings of the 37th International Conference on Machine Learning*, pages 6393–6403, 2020.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In Dekang Lin, Yuji Matsumoto, and Rada Mihalcea, editors, *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011a. Association for Computational Linguistics. URL https://aclanthology.org/P11-1015.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011b. Association for Computational Linguistics. URL https://aclanthology.org/P11-1015.

Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. Language models of code are few-shot commonsense learners, 2022.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19 (2):313–330, 1993a. URL https://aclanthology.org/J93-2004.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19 (2):313–330, 1993b. URL https://aclanthology.org/J93-2004.

Abhinav Mehrotra, Alberto Gil, C. P. Ramos, Sourav Bhattacharya, Łukasz Dudziak, Ravichander Vipperla, Thomas Chau, Samin Ishtiaq, Mohamed S. Abdelfattah, and Nicholas D. Lane. NAS-Bench-ASR: Reproducible neural architecture search for speech recognition. In *Proceedings of the 8th International Conference on Learning Representations*, 2021.

Jieru Mei, Yingwei Li, Xiaochen Lian, Xiaojie Jin, Linjie Yang, Alan Yuille, and Jianchao Yang. AtomNAS: Fine-grained end-to-end neural architecture search. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

Joseph Mellor, Jack Turner, Amos Storkey, and Elliot J. Crowley. Neural architecture search without training. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

Eric J. Michaud, Ziming Liu, Uzay Girit, and Max Tegmark. The quantization model of neural scaling, 2023.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018.

Sören Mindermann, Muhammed Razzak, Winnie Xu, Andreas Kirsch, Mrinank Sharma, Adrien Morisot, Aidan N. Gomez, Sebastian Farquhar, Jan Brauner, and Yarin Gal. Prioritized training on points that are learnable, worth learning, and not yet learned (workshop version), 2021.

Stanislav Minsker. Geometric median and robust estimation in Banach spaces. *Bernoulli*, 21(4):2308 – 2335, 2015. doi: 10.3150/14-BEJ645. URL https://doi.org/10.3150/14-BEJ645.

Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1003–1011. Association for Computational Linguistics, 2009.

Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models, 2019.

Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. Cross-task generalization via natural language crowdsourcing instructions. In *ACL*, 2022.

Marcin Moczulski, Misha Denil, Jeremy Appleyard, and Nando de Freitas. ACDC: A structured efficient linear layer. arXiv, 2015.

Robert C. Moore and William Lewis. Intelligent selection of language model training data. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 220–224, Uppsala, Sweden, July 2010. Association for Computational Linguistics. URL https://aclanthology.org/P10-2041.

MosaicML. Introducing mpt-7b: A new standard for open-source, commercially usable llms, 2023. URL https://www.mosaicml.com/blog/mpt-7b.

Moin Nadeem, Anna Bethke, and Siva Reddy. Stereoset: Measuring stereotypical bias in pretrained language models. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2021. doi: 10.18653/v1/2021.acl-long.416. URL http://dx.doi.org/10.18653/v1/2021.acl-long.416.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=9XFSbDPmdW.

Nikita Nangia and Samuel R. Bowman. Listops: A diagnostic dataset for latent tree learning, 2018.

Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Amar Budhiraja, Despoina Magka, Vladislav Vorotilov, Gaurav Chaurasia, Dieuwke Hupkes, Ricardo Silveira Cabral, Tatiana Shavrina, Jakob Foerster, Yoram Bachrach, William Yang Wang, and Roberta Raileanu. Mlgym: A new framework and benchmark for advancing ai research agents. *arXiv preprint arXiv:2502.14499*, 2025.

Vladimir Nekrasov, Hao Chen, Chunhua Shen, and Ian Reid. Fast neural architecture search of compact semantic segmentation models via auxiliary cells. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

Arkadij Semenovič Nemirovskij and David Borisovich Yudin. Problem complexity and method efficiency in optimization. 1983.

Mohammad Norouzi, Tomas Mikolov, Samy Bengio, Yoram Singer, Jonathon Shlens, Andrea Frome, Greg Corrado, and Jeffrey Dean. Zero-shot learning by convex combination of semantic embeddings. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2014. URL http://dblp.uni-trier.de/db/conf/iclr/iclr2014.html#NorouziMBSSFCD13.

Adamantios Ntakaris, Martin Magris, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods. http://urn.fi/urn:nbn:fi:csc-kata20170601153214969115. N/A.

Tom M W Nye, Xiaoxian Tang, Grady Weyenberg, and Ruriko Yoshida. Principal component analysis and the locus of the fréchet mean in the space of phylogenetic trees. *Biometrika*, 104(4):901–922, Dec 2017. ISSN 0006-3444 (Print); 1464-3510 (Electronic); 0006-3444 (Linking). doi: 10.1093/biomet/asx047.

Gabriel Orlanski, Nicholas Roberts, Aws Albarghouthi, and Frederic Sala. Reward models enable scalable code verification by trading accuracy for throughput. *arXiv preprint arXiv:2506.10056*, 2025.

Isabel Papadimitriou and Dan Jurafsky. Learning Music Helps You Read: Using transfer to study linguistic structure in language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6829–6839, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.554. URL https://aclanthology.org/2020.emnlp-main.554.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset, Aug 2016.

Jongho Park, Jaeseung Park, Zheyang Xiong, Nayoung Lee, Jaewoong Cho, Samet Oymak, Kangwook Lee, and Dimitris Papailiopoulos. Can mamba learn how to learn? a comparative study on in-context learning tasks, 2024.

Ioannis Partalas, Aris Kosmopoulos, Nicolas Baskiotis, Thierry Artières, Georgios Paliouras, Éric Gaussier, Ion Androutsopoulos, Massih-Reza Amini, and Patrick Gallinari. Lshtc: A benchmark for large-scale text classification. *ArXiv*, abs/1503.08581, 2015.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Curran Associates Inc., Red Hook, NY, USA, 2019.

Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. Deep learning on a data diet: Finding important examples early in training, 2021.

Adrien Pavao, Isabelle Guyon, Anne-Catherine Letournel, Xavier Baró, Hugo Escalante, Sergio Escalera, Tyler Thomas, and Zhen Xu. Codalab competitions: An open source platform to organize scientific challenges. *Technical report*, 2022. URL `https://hal.inria.fr/hal-03629462v1`.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011a.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011b.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Hamza Alobeidli, Alessandro Cappelli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The refinedweb dataset for falcon LLM: Outperforming curated corpora with web data only. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL `https://openreview.net/forum?id=kM5eGcdCzq`.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024a. URL `https://openreview.net/forum?id=n6SCkn2QaG`.

Guilherme Penedo, Hynek Kydlíček, Leandro von Werra, and Thomas Wolf. Fineweb, 2024b. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Leon Derczynski, Xingjian Du, Matteo Grella, Kranthi Gv, Xuzheng He, Haowen Hou, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartłomiej Koptyra, Hayden Lau, Jiaju Lin, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Guangyu Song, Xiangru Tang, Johan Wind, Stanisław Woźniak, Zhenyuan Zhang, Qinghua Zhou, Jian Zhu, and Rui-Jie Zhu. RWKV: Reinventing RNNs for the transformer era. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14048–14077, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.936. URL https://aclanthology.org/2023.findings-emnlp.936.

Alexander Petersen and Hans-Georg Muller. Fréchet regression for random objects with euclidean predictors. *The Annals of Statistics*, 2016.

François Petitjean, Jordi Inglada, and Pierre Gancarski. Satellite image time series analysis under time warping. *IEEE Transactions on Geoscience and Remote Sensing*, 50 (8):3081–3095, 2012. doi: 10.1109/TGRS.2011.2179050.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Jeff M. Phillips. Coresets and sketches, 2016.

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y. Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: towards larger convolutional language models. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023a.

Michael Poli, Jue Wang, Stefano Massaroli, Jeffrey Quesnelle, Ryan Carlow, Eric Nguyen, and Armin Thomas. StripedHyena: Moving Beyond Transformers with Hybrid Signal Processing Models, 12 2023b. URL https://github.com/togethercomputer/stripedhyena.

Michael Poli, Armin W Thomas, Eric Nguyen, Pragaash Ponnusamy, Björn Deiseroth, Kristian Kersting, Taiji Suzuki, Brian Hie, Stefano Ermon, Christopher Ré, et al. Mechanistic design and scaling of hybrid architectures. *arXiv preprint arXiv:2403.17844*, 2024.

Felipe Maia Polo, Seamus Somerstep, Leshem Choshen, Yuekai Sun, and Mikhail Yurochkin. Sloth: scaling laws for llm skills to predict multi-benchmark performance across families, 2025. URL https://arxiv.org/abs/2412.06540.

Tomer Porian, Mitchell Wortsman, Jenia Jitsev, Ludwig Schmidt, and Yair Carmon. Resolving discrepancies in compute-optimal scaling of language models. *Advances in Neural Information Processing Systems*, 37:100535–100570, 2024.

Ben Prystawski and Noah D. Goodman. Why think step-by-step? reasoning emerges from the locality of experience, 2023.

Reid Pryzant, Ziyi Yang, Yichong Xu, Chenguang Zhu, and Michael Zeng. Automatic rule induction for efficient semi-supervised learning, 2022. URL https://arxiv.org/abs/2205.09067.

A Yang Qwen, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengpeng Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint*, 2024.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In Jian Su, Kevin Duh, and Xavier Carreras, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264/.

Nikitha Rao, Kush Jain, Uri Alon, Claire Le Goues, and Vincent J. Hellendoorn. Cat-lm training language models on aligned code and tests. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering*, ASE '23, page 409–420. IEEE Press, 2024. ISBN 9798350329964. doi: 10.1109/ASE56229.2023.00193. URL https://doi.org/10.1109/ASE56229.2023.00193.

A. J. Ratner, B. Hancock, J. Dunnmon, F. Sala, S. Pandey, and C. Ré. Training complex models with multi-task weak supervision. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Honolulu, Hawaii, 2019.

Alexander Ratner, Christopher De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. Data programming: Creating large training sets, quickly. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 3574–3582, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Alexander J. Ratner, Stephen H. Bach, Henry R. Ehrenberg, Jason Alan Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, 11 3:269–282, 2017.

Christopher Ré, Feng Niu, Pallavi Gudipati, and Charles Srisuwananukorn. Overton: A data system for monitoring and improving machine-learned products. In *Proceedings of the 10th Annual Conference on Innovative Data Systems Research*, 2020.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, 2019.

Esteban Real, Chen Liang, David R. So, and Quoc V. Le. Automl-zero: Evolving machine learning algorithms from scratch. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020a.

Esteban Real, Chen Liang, David R. So, and Quoc V. Le. AutoML-Zero: Evolving machine learning algorithms from scratch. In *Proceedings of the 37th International Conference on Machine Learning*, 2020b.

Siddharth Reddy, Igor Labutov, and Thorsten Joachims. Latent skill embedding for personalized lesson sequence recommendation, 2016.

Wendi Ren, Yinghao Li, Hanting Su, David Kartchner, Cassie Mitchell, and Chao Zhang. Denoising multi-source weak supervision for neural text classification. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3739–3754, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.334. URL https://aclanthology.org/2020.findings-emnlp.334.

Nicholas Roberts, Mikhail Khodak, Tri Dao, Liam Li, Chris Ré, and Ameet Talwalkar. Rethinking neural operations for diverse tasks. arXiv, 2021a.

Nicholas Roberts, Mikhail Khodak, Tri Dao, Liam Li, Christopher Ré, and Ameet Talwalkar. Rethinking neural operations for diverse tasks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 15855–15869. Curran Associates, Inc., 2021b. URL https://proceedings.neurips.cc/paper/2021/file/84fdbc3ac902561c00871c9b0c226756-Paper.pdf.

Nicholas Roberts, Samuel Guo, Cong Xu, Ameet Talwalkar, David Lander, Lvfang Tao, Linhang Cai, Shuaicheng Niu, Jianyu Heng, Hongyang Qin, Minwen Deng, Johannes Hog, Alexander Pfefferle, Sushil Ammanaghatta Shivakumar, Arjun Krishnakumar, Yubo Wang, Rhea Sukthanker, Frank Hutter, Euxhen Hasanaj, Tien-Dung Le, Mikhail Khodak, Yuriy Nevmyvaka, Kashif Rasul, Frederic Sala, Anderson Schneider, Junhong Shen, and Evan Sparks. Automl decathlon: Diverse tasks, modern methods, and efficiency at scale. In Marco Ciccone, Gustavo Stolovitzky, and Jacob Albrecht, editors, *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pages 151–170. PMLR, 28 Nov–09 Dec 2022a. URL https://proceedings.mlr.press/v220/roberts23a.html.

Nicholas Roberts, Xintong Li, Tzu-Heng Huang, Dyah Adila, Spencer Schoenberg, Cheng-Yu Liu, Lauren Pick, Haotian Ma, Aws Albarghouthi, and Frederic Sala. AutoWS-bench-101: Benchmarking automated weak supervision with 100 labels. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022b. URL https://openreview.net/forum?id=nQZHEunntbJ.

Nicholas Roberts, Xintong Li, Tzu-Heng Huang, Dyah Adila, Spencer Schoenberg, Cheng-Yu Liu, Lauren Pick, Haotian Ma, Aws Albarghouthi, and Frederic Sala. AutoWS-bench-101: Benchmarking automated weak supervision with 100 labels. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022c. URL https://openreview.net/forum?id=nQZHEunntbJ.

Nicholas Roberts, Niladri S Chatterji, Sharan Narang, Mike Lewis, and Dieuwke Hupkes. Compute optimal scaling of skills: Knowledge vs reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 13295–13316, 2025.

Nicholas Carl Roberts, Mikhail Khodak, Tri Dao, Liam Li, Christopher Re, and Ameet Talwalkar. Rethinking neural operations for diverse tasks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021c. URL https://openreview.net/forum?id=je4ymjfb5LC.

Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1119–1129, Denver, Colorado, May–June 2015. Association for Computational Linguistics. doi: 10.3115/v1/N15-1118. URL https://aclanthology.org/N15-1118.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention*, 2015.

Jonathan S. Rosenfeld, Amir Rosenfeld, Yonatan Belinkov, and Nir Shavit. A constructive prediction of the generalization error across scales. *ArXiv*, abs/1909.12673, 2019. URL https://api.semanticscholar.org/CorpusID:203592013.

Abhinaba Roy, Deepanway Ghosal, Erik Cambria, Navonil Majumder, Rada Mihalcea, and Soujanya Poria. Improving Zero-Shot Learning Baselines with Commonsense Knowledge. *Cognitive Computation*, 14(6):2212–2222, November 2022. ISSN 1866-9964. doi: 10.1007/s12559-022-10044-0. URL https://doi.org/10.1007/s12559-022-10044-0.

Sebastian Ruder and Barbara Plank. Learning to select data for transfer learning with bayesian optimization. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017. doi: 10.18653/v1/d17-1038. URL http://dx.doi.org/10.18653/v1/D17-1038.

Sebastian Ruder, Parsa Ghaffari, and John G. Breslin. Data selection strategies for multi-domain sentiment analysis, 2017.

Alessandro Rudi, Carlo Ciliberto, GianMaria Marconi, and Lorenzo Rosasco. Manifold structured prediction. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018a. URL https://proceedings.neurips.cc/paper/2018/file/f6185f0ef02dcaec414a3171cd01c697-Paper.pdf.

Alessandro Rudi, Carlo Ciliberto, GianMaria Marconi, and Lorenzo Rosasco. Manifold structured prediction. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2018)*, volume 32, 2018b.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Jon Saad-Falcon, E Kelly Buchanan, Mayee F Chen, Tzu-Heng Huang, Brendan McLaughlin, Tanvir Bhathal, Shang Zhu, Ben Athiwaratkun, Frederic Sala, Scott Linderman, et al. Shrinking the generation-verification gap with weak verifiers. *arXiv preprint arXiv:2506.18203*, 2025.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Stella Biderman, Leo Gao, Tali Bers, Thomas Wolf, and Alexander M. Rush. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations (ICLR)*, 2022.

Nikhil Sardana, Sasha Doubov, and Jonathan Frankle. Beyond chinchilla-optimal: Accounting for inference in language model scaling laws. In *International Conference on Machine Learning*, 2023. URL `https://api.semanticscholar.org/CorpusID:266693796`.

Shreyas Saxena, Oncel Tuzel, and Dennis DeCoste. Data parameters: A new family of parameters for learning a differentiable curriculum. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper_files/paper/2019/file/926ffc0ca56636b9e73c565cf994ea5a-Paper.pdf`.

Rylan Schaeffer, Joshua Kazdan, John Hughes, Jordan Juravsky, Sara Price, Aengus Lynch, Erik Jones, Robert Kirk, Azalia Mirhoseini, and Sanmi Koyejo. How do large language monkeys get their power (laws)? In *Forty-second International Conference on Machine Learning*, 2025. URL `https://openreview.net/forum?id=QqVZ28qems`.

Rylan Schaeffer, Noam Itzhak Levi, Brando Miranda, and Sanmi Koyejo. Pretraining scaling laws for generative evaluations of language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL `https://openreview.net/forum?id=Ym33xJYINV`.

Junhong Shen, Mikhail Khodak, and Ameet Talwalkar. Efficient architecture search for diverse tasks, 2022.

Junhong Shen, Liam Li, Lucio M. Dery, Corey Staten, Mikhail Khodak, Graham Neubig, and Ameet Talwalkar. Cross-modal fine-tuning: Align then refine, 2023. URL `https://arxiv.org/abs/2302.05738`.

Changho Shin, Winfred Li, Harit Vishwakarma, Nicholas Carl Roberts, and Frederic Sala. Universalizing weak supervision. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=YpPiNigTzMT`.

Mustafa Shukor, Enrico Fini, Victor Guilherme Turrisi da Costa, Matthieu Cord, Joshua Susskind, and Alaaeldin El-Nouby. Scaling laws for native multimodal models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12–23, 2025.

Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375: 1339–1364, 2018.

Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE, 2017.

Ryan Smith, Jason A. Fries, Braden Hancock, and Stephen H. Bach. Language models in the loop: Incorporating prompting into weak supervision. 2022. URL `https://arxiv.org/pdf/2205.02318.pdf`.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Richard Socher, Milind Ganjoo, Christopher D Manning, and Andrew Ng. Zero-shot learning through cross-modal transfer. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL `https://proceedings.neurips.cc/paper/2013/file/2d6cc4b2d139a53512fb8cbb3086ae2e-Paper.pdf`.

Ben Sorscher, Robert Geirhos, Shashank Shekhar, Surya Ganguli, and Ari S. Morcos. Beyond neural scaling laws: beating power law scaling via data pruning, 2022.

Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. Curriculum learning: A survey. *International Journal of Computer Vision*, 130(6):1526–1565, Apr 2022. ISSN 1573-1405. doi: 10.1007/s11263-022-01611-x. URL `http://dx.doi.org/10.1007/s11263-022-01611-x`.

Jacob Mitchell Springer, Sachin Goyal, Kaiyue Wen, Tanishq Kumar, Xiang Yue, Sadhika Malladi, Graham Neubig, and Aditi Raghunathan. Overtrained language models are harder to fine-tune. *arXiv preprint arXiv:2503.19206*, 2025.

Claire Stevenson, Iris Smal, Matthijs Baas, Raoul Grasman, and Han van der Maas. Putting gpt-3's creativity to the (alternative uses) test, 2022.

Russell Stewart and Stefano Ermon. Label-free supervision of neural networks with physics and domain knowledge. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Cathie Sudlow, John Gallacher, Naomi Allen, Valerie Beral, Paul Burton, John Danesh, Paul Downey, Paul Elliott, Jane Green, Martin Landray, Bette Liu, Paul Matthews, Giok Ong, Jill Pell, Alan Silman, Alan Young, Tim Sprosen, Tim Peakman, and Rory Collins. Uk biobank: an open access resource for identifying the causes of a wide range of complex diseases of middle and old age. *PLoS Med*, 12(3):e1001779, Mar 2015. ISSN 1549-1676 (Electronic); 1549-1277 (Print); 1549-1277 (Linking). doi: 10.1371/journal.pmed.1001779.

Devin P Sullivan, Casper F Winsnes, Lovisa Åkesson, Martin Hjelmare, Mikaela Wiking, Rutger Schutten, Linzi Campbell, Hjalti Leifsson, Scott Rhodes, Andie Nordgren, Kevin Smith, Bernard Revaz, Bergur Finnbogason, Attila Szantner, and Emma Lundberg. Deep learning is combined with massive-scale citizen science to improve large-scale image classification. *Nature Biotechnology*, 36(9):820–828, 2018. doi: 10.1038/nbt.4225. URL https://doi.org/10.1038/nbt.4225.

Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Dan MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. PDEBench: An Extensive Benchmark for Scientific Machine Learning. In *36th Conference on Neural Information Processing Systems (NeurIPS 2022) Track on Datasets and Benchmarks*, 2022. URL https://arxiv.org/abs/2210.07182.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena : A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=qVyeW-grC2k.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Together. Redpajama-data: An open source recipe to reproduce llama training dataset, 2023. URL https://github.com/togethercomputer/RedPajama-Data.

Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J. Gordon. An empirical study of example forgetting during deep neural network learning, 2018.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023a.

Hugo Touvron, Louis Martin, Kevin R. Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Daniel M. Bikel, Lukas Blecher, Cristian Cantón Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony S. Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel M. Kloumann, A. V. Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, R. Subramanian, Xia Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zhengxu Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *ArXiv*, abs/2307.09288, 2023b. URL `https://api.semanticscholar.org/CorpusID:259950998`.

Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Utku Evci, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, and Hugo Larochelle. Meta-dataset: A dataset of datasets for learning to learn from few examples. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

Renbo Tu, Nicholas Roberts, Mikhail Khodak, Junhong Shen, Frederic Sala, and Ameet Talwalkar. NAS-bench-360: Benchmarking neural architecture search on diverse tasks. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022a. URL `https://openreview.net/forum?id=xUXTbq6gWsB`.

Renbo Tu, Nicholas Roberts, Mikhail Khodak, Junhong Shen, Frederic Sala, and Ameet Talwalkar. NAS-bench-360: Benchmarking neural architecture search on diverse tasks. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022b. URL `https://openreview.net/forum?id=xUXTbq6gWsB`.

Amir Vajdi, Mohammad Reza Zaghian, Nazli Rafei Dehkordi, Elham Rastegari, Kian Maroofi, Saman Farahmand, Shaohua Jia, Marc Pomplun, Nurit Haspel, and Akram Bayat. Human gait database for normal walk collected by smartphone accelerometer, 2019. URL https://arxiv.org/abs/1905.03109.

Paroma Varma and Christopher Ré. Snuba: Automating weak supervision to label training data. *Proc. VLDB Endow.*, 12(3):223–236, nov 2018. ISSN 2150-8097. doi: 10.14778/3291264.3291268. URL https://doi.org/10.14778/3291264.3291268.

Neeraj Varshney, Swaroop Mishra, and Chitta Baral. Let the model decide its curriculum for multitask learning, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017a. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017b.

Roman Vershynin. Introduction to the non-asymptotic analysis of random matrices. In *Compressed Sensing*, 2010. URL https://api.semanticscholar.org/CorpusID:133440.

Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. Will we run out of data? limits of llm scaling based on human-generated data, 2024. URL https://arxiv.org/abs/2211.04325.

Harit Vishwakarma and Frederic Sala. Lifting weak supervision to structured prediction. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=Cntmos_Ndf0.

Xingchen Wan, Binxin Ru, Pedro M Esperança, and Zhenguo Li. On redundancy and diversity in cell-based neural architecture search. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=rFJWoYoxrDB.

Fei Wang and Changshui Zhang. Label propagation through linear neighborhoods. *IEEE Transactions on Knowledge and Data Engineering*, 20(1):55–67, 2008. doi: 10.1109/TKDE.2007.190672.

X. Wang, Y. Ye, and A. Gupta. Zero-shot recognition via semantic embeddings and knowledge graphs. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6857–6866, Los Alamitos, CA, USA, jun 2018. IEEE Computer Society. doi: 10.1109/CVPR.2018.00717. URL https://doi.ieeecomputersociety.org/10.1109/CVPR.2018.00717.

Xiao Wang, Weikang Zhou, Qi Zhang, Jie Zhou, Songyang Gao, Junzhe Wang, Menghan Zhang, Xiang Gao, Yunwen Chen, and Tao Gui. Farewell to aimless large-scale pretraining: Influential subset selection for language model, 2023.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language model with self generated instructions, 2022a.

Yizhong Wang, Swaroop Mishra, et al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks, 2022b.

Yujing Wang, Yaming Yang, Yiren Chen, Jing Bai, Ce Zhang, Guinan Su, Xiaoyu Kou, Yunhai Tong, Mao Yang, and Lidong Zhou. Textnas: A neural architecture search space tailored for text representation. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, 2020a.

Zhiwei Wang, Yao Ma, Zitao Liu, and Jiliang Tang. R-Transformer: Recurrent neural network enhanced transformer. In *Proceedings of the 8th International Conference on Learning Representations*, 2020b.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners, 2021.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Colin White, Willie Neiswanger, and Yash Savani. BANANAS: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, 2021.

Colin White, Mikhail Khodak, Renbo Tu, Shital Shah, Sébastien Bubeck, and Debadeepta Dey. A deeper look at zero-cost proxies for lightweight nas. In *ICLR Blog Track*, 2022. URL https://iclr-blog-track.github.io/2022/03/25/zero-cost-proxies/. https://iclr-blog-track.github.io/2022/03/25/zero-cost-proxies/.

Richard T White. Research into learning hierarchies. *Review of Educational Research*, 43(3):361–375, 1973.

Richard T. White and Robert M. Gagné. Past and future research on learning hierarchies. *Educational Psychologist*, 11(1):19–28, 1974. doi: 10.1080/00461527409529119. URL https://doi.org/10.1080/00461527409529119.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. *ArXiv*, abs/2203.05482, 2022. URL https://api.semanticscholar.org/CorpusID:247362886.

Xiaoxia Wu, Ethan Dyer, and Behnam Neyshabur. When do curricula work?, 2020.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. LESS: Selecting influential data for targeted instruction tuning. In *International Conference on Machine Learning (ICML)*, 2024.

Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy Liang, Quoc V Le, Tengyu Ma, and Adams Wei Yu. Doremi: Optimizing data mixtures speeds up language model pretraining. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023a. URL https://openreview.net/forum?id=lXuByUeHhd.

Sang Michael Xie, Shibani Santurkar, Tengyu Ma, and Percy Liang. Data selection for language models via importance resampling, 2023b.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. PC-DARTS: Partial channel connections for memory-efficient architecture search. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. TIES-merging: Resolving interference when merging models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=xtaX3WyCj1.

Antoine Yang, Pedro M. Esperança, and Fabio M. Carlucci. NAS evaluation is frustratingly hard. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training, 2024.

Jiasheng Ye, Peiju Liu, Tianxiang Sun, Jun Zhan, Yunhua Zhou, and Xipeng Qiu. Data mixing laws: Optimizing data mixtures by predicting language modeling performance. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=jjCB27TMK3.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Dingli Yu, Simran Kaur, Arushi Gupta, Jonah Brown-Cohen, Anirudh Goyal, and Sanjeev Arora. SKILL-MIX: a flexible and expandable family of evaluations for AI models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Jf5gplvglq.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. *CoRR*, abs/2311.03099, 2023. URL https://doi.org/10.48550/arXiv.2311.03099.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference 2016*. British Machine Vision Association, 2016a.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference*, 2016b.

Amir R. Zamir, Alexander Sax, William Shen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun 2018. doi: 10.1109/cvpr.2018.00391. URL http://dx.doi.org/10.1109/CVPR.2018.00391.

Arber Zela, Julien Siems, and Frank Hutter. NAS-Bench-1Shot1: Benchmarking and dissecting one-shot neural architecture search. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

Jieyu Zhang, Yue Yu, Yinghao Li, Yujing Wang, Yaming Yang, Mao Yang, and Alexander Ratner. WRENCH: A comprehensive benchmark for weak supervision. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021a. URL https://openreview.net/forum?id=Q9SKS5k8io.

Keming Zhang and Joshua S. Bloom. deepCR: Cosmic ray rejection with deep learning. *The Astrophysical Journal*, 889(1):24, jan 2020a. doi: 10.3847/1538-4357/ab3fa6.

Keming Zhang and Joshua S Bloom. deepcr: Cosmic ray rejection with deep learning. *The Astrophysical Journal*, 889(1):24, 2020b.

Michael Zhang, Kush Bhatia, Hermann Kumbong, and Christopher Ré. The hedgehog & the porcupine: Expressive linear attentions with softmax mimicry. *arXiv preprint arXiv:2402.04347*, 2024.

Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Wenyue Hua, Haolun Wu, Zhihan Guo, Yufei Wang, Niklas Muennighoff, et al. A survey on test-time scaling in large language models: What, how, where, and how well? *arXiv preprint arXiv:2503.24235*, 2025.

Rongzhi Zhang, Yue Yu, Pranav Shetty, Le Song, and Chao Zhang. Prompt-based rule discovery and boosting for interactive weakly-supervised learning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 745–758, Dublin, Ireland, May 2022a. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.55. URL https://aclanthology.org/2022.acl-long.55.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022b.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/250cf8b51c773f3f8dc8b4be867a9a02-Paper.pdf.

Yi Zhang, Arturs Backurs, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, and Tal Wagner. Unveiling transformers with lego: a synthetic reasoning task, 2022c.

Zijun Zhang, Evan M Cofer, and Olga G Troyanskaya. Ambient: accelerated convolutional neural network architecture search for regulatory genomics. *bioRxiv*, 2021b.

Zijun Zhang, Christopher Y Park, Chandra L Theesfeld, and Olga G Troyanskaya. An automated framework for efficiently designing deep convolutional neural networks in genomics. *Nature Machine Intelligence*, 3(5):392–400, 2021c.

Jian Zhou and Olga G Troyanskaya. Predicting effects of noncoding variants with deep learning–based sequence model. *Nature methods*, 12(10):931–934, 2015.

Terry Yue Zhuo, Vu Minh Chien, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen GONG, James Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, David Lo, Binyuan Hui, Niklas Muennighoff, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=YrycTjllL0`.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.