Programming Language Principles for AI Systems: Interpretation, Duality and Incremental Computation

by

Zi Wang

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

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2025

Date of final oral examination: 12/22/2025

For those I have been, and those I have yet to become.

Preface

Pursuing a PhD is often described as a long and solitary journey. While this characterization holds some truth, it fails to capture the richness and complexity of the path, marked not only by intellectual rigor, but also by transformation, self discovery, and personal growth. Over the course of my doctoral studies, I evolved from a theoretically inclined student into a more well rounded researcher capable of tackling real world computational challenges. Confronting the ambiguity and intricacy of practical problems taught me to abstract away physical and semantic fuzziness to reveal their essential structure. This shift in perspective profoundly shaped my approach to research and to life, fostering a greater comfort with uncertainty and a deeper appreciation for empirical grounding.

Curiosity has always been central to my intellectual identity. As an undergraduate, I was driven by a desire to understand the world at its most fundamental level. This led me to explore a diverse range of disciplines, including biology, psychology, philosophy, mathematics, and computer science. My natural inclination toward reductionism motivated me to search for unifying principles that connect seemingly disparate fields. Over time, language emerged as a common thread. As both a medium for thought and a carrier of knowledge, it became a focal point in my academic exploration. This realization led me to study linguistics, analytic philosophy, and computer science in depth, particularly to understand the expressive and computational power of formal systems. This path ultimately brought me to programming languages research, though my work continues to reflect the theoretical computer science foundation I received during my training.

The past decade has witnessed tremendous progress in artificial intelligence, first through the success of convolutional neural networks in computer vision, and more recently with the emergence of transformer based language models. My initial research in artificial intelligence involved deep neural networks, and one of my first substantial projects focused on estimating Lipschitz constants of neural networks using semidefinite programming. While theoretical in nature, the project was motivated by empirical observations, and taught me a lesson echoed in the writings of many scientific pioneers: meaningful theory is often guided by practice. That realization became one of the most memorable eureka moments of my research journey.

In more recent years, the field has shifted dramatically toward large language models, which have exhibited remarkable general capabilities across domains. This marks a paradigm shift in the landscape of computer science research. As a programming language researcher, I find the linguistic behaviors of these models both exciting and deeply thought provoking. While earlier neural networks presented unresolved theoretical challenges, language models introduce new questions about structure, generalization, and meaning. I have become especially interested in how we can cultivate their capabilities and what their increasing fluency and precision imply about intelligence, interpretation, and the future of humanity.

As someone who studies language formally, I have also come to use language as a tool to interpret the world around me. At the beginning of my PhD, research felt like a collection of static symbols, mathematical expressions, theoretical constructs, and code, awaiting meaning. Over time, as I engaged more deeply, these symbols became dynamic and saturated with personal experience. The semantics of research, the lived reality of discovery, gradually filled in the gaps left by syntax alone. This transformation extended beyond academia. Words and symbols in everyday life have acquired more nuance, and I have become increasingly attuned to the fluid and contextual nature of meaning. As the title of this dissertation suggests, I believe the core of research is interpretation. It is the process of understanding how dynamic semantic structures evolve and interact. This perspective has allowed me to study systems that solve problems not in binary terms, but incrementally and flexibly, across a spectrum of difficulty.

I have come to see research as an activity that demands the willingness to *try everything*, knowing that failure is often part of the process. Progress is rarely linear. It is shaped by getting things wrong, by returning to the same questions, and by learning through missteps rather than immediate success. What matters is not avoiding failure, but refusing to give up or give in, and recognizing that reaching an apparent end is often the moment to begin again with deeper understanding.

These reflections would not have been possible without the people who supported and shaped my growth. I am deeply grateful to my advisor, Somesh Jha, for granting me the intellectual freedom to explore widely and for sharing conceptual insights that shaped my thinking beyond any specific project. I also owe a great deal to my former advisor, Benjamin Liblit, whose enthusiasm for empirical research, sense of humor, and intellectual playfulness profoundly influenced my early development. One of his memorable projects, a C program that hijacks compiler constructors and destructors, remains one of the most ingenious pieces of code I have encountered.

I would also like to thank Thomas Reps, Aws Albarghouthi, Yudong Chen, and other faculty at UW-Madison for their mentorship and encouragement. Whether through rigorous critique, quiet support, or a willingness to entertain speculative ideas, each contributed meaningfully to my growth as a researcher.

I am grateful to my collaborators and academic friends: David Brown, Jordan Henkel, Goutham Ramakrishnan, Earlence Fernandes, Thomas Kobber Panum, Gautam Prakriya, Zifan Wang, Vijay Bhattiprolu, Jihye Choi, Ke Wang, Bin Hu, Aaron Havens, Alexandre Araujo, Yang Zheng, Divyam Anshumaan, and Ashish Hooda. From each, I have learned immensely, both technically and personally.

I also thank my peers in the MadPL group: Zhicheng Cai, Jinman Zhao, Calvin Smith, Jason Breck, Samuel Drews, Qinheping Hu, John Cyphert, Anvay Grover, Yuhao Zhang, Jialu Bao, Anna Meyer, Amanda Xu, and Abtin Molavi. Their companionship made the long years of graduate school not only stimulating but joyful.

I reserve a special thanks for my close friend and long time collaborator, Shiwei Weng. Our friendship, which began in high school and has spanned countless discussions about programming languages, linguistics, literature, sociology, and life, has been a constant source of inspiration and support.

Finally, I owe everything to my parents. As I have grown older, my understanding of our relationship has deepened. What was once defined by authority has become one rooted in empathy, complexity, and gratitude. They have always given me the freedom to pursue my intellectual passions without pressure or demand, and for that I am profoundly thankful.

I am also grateful to the many past versions of myself, the ones who stayed curious, persisted through frustration, and continued forward even when the path felt uncertain. Their resilience made this dissertation possible.

To all of you, thank you.

Abstract

This dissertation develops a unified programming–language perspective on modern AI systems through three principles: *interpretation*, *duality*, and *incremental computation*. These principles provide semantic and structural tools for understanding how neural networks and large language models behave, how their behaviors can be represented, and how difficult optimization problems surrounding them can be made tractable.

The first contribution establishes an Interval Universal Approximation (IUA) theorem: networks with squashable activations can approximate interval semantics over input sets, extending classical pointwise approximation. We further show that high-precision interval range approximation is Δ_2–intermediate, revealing fundamental limits for interval-based abstractions.

The second contribution develops an algebraic interpretation of neural networks by encoding their computations as polynomial constraint systems. This representation supports geometric and optimization-based analyses, including semidefinite relaxations for global Lipschitz bounds derived via Lagrangian duality, clarifying the expressive boundary of quadratic encodings.

The final contribution introduces incremental methods for solving difficult optimization problems through structured trajectories of easier ones. We present a first-order incremental solver that refines spectral-product estimates into SDP-quality Lipschitz certificates, and a *functional homotopy* method for discrete input optimization—such as jailbreak prompt synthesis—that smooths the search landscape via a sequence of weakened model checkpoints. Both methods demonstrate how incremental computation bridges semantic complexity and practical algorithm design.

Together, these contributions show how semantic structure, dual formulations, and incremental computation form a coherent framework for analyzing and shaping the behavior of neural networks and language models.

Contents

Abstract **v**

Contents **vi**

List of Tables **x**

List of Figures **xi**

I Introduction **1**

1 Introduction **2**

1.1 Motivation: Reasoning About AI Systems 2

1.2 Interpretation: From Pointwise to Set-Based and Symbolic Semantics . . . 2

1.3 Duality: Logical, Functional, and Optimization Perspectives 3

1.4 Incremental Computation: Approximation as a Trajectory 4

2 Notation and Mathematical Preliminaries **8**

2.1 Sets, Logic, and Linear Algebra 8

2.2 Topology 9

2.3 Functional Analysis 9

2.4 Differentiation and Gradients 10

2.5 Algebraic Preliminaries 10

2.6 Optimization: Objectives and Constraints 11

3 Neural Networks as Programs: Testing and Verification **12**

3.1 Testing and Verification: From Software to AI Systems 12

3.2 Neural Networks as Parameterized Programs 13

3.3 Adversarial Robustness and Jailbreak Attacks 14

3.4 Formal Structures of Neural Network Programs 15
3.5 Optimization Formulations 17

4 Major Technical Tools 18
4.1 Descriptive Power and Complexity of Logical Forms 18
4.2 Abstract Interpretation 21
4.3 Linear Approximation and Lipschitz Regularity 22
4.4 Algebraic Formulations of Decision Problems 24
4.5 Lagrangian Duality and Shor's Relaxation 26

II Interval Universal Approximation 27

5 The Interval Universal Approximation: Introduction 30
5.1 Motivation 30
5.2 Abstract Interpretation and Interval Abstraction 31
5.3 Squashable and Step Functions 32
5.4 Interval Universal Approximation: Definition 36
5.5 Positive Result: Existence of Interval Approximators 37
5.6 Negative Result: Hardness of Interval Approximation 38

6 Interval Universal Approximation: Constructive Proof 40
6.1 Proof of the IUA Theorem 40
6.2 Approximating Indicator Functions 41
6.3 Complete Proof of the IUA Theorem 46

7 Hardness of Interval Approximation 49
7.1 Hardness of Range Approximation 51

IIIAlgebraic Interpretation of Neural Networks 56

8 Symbolic and Algebraic Interpretation of Neural Networks 59
8.1 Algebraic Unfolding of Feedforward Networks 59
8.2 Classification Semantics as Algebraic Relations 61
8.3 What Fits into Quadratic Form: Problems and Constructions 62
8.4 Input Constraints: A Quadratic Toolkit 63
8.5 Activation Expressivity and Semialgebraic Limits 63

8.6 Decidability via the First-Order Theory of the Reals 64
8.7 Summary and Scope 64

9 Semidefinite Relaxation and Quantitative Guarantees 66
9.1 From Algebraic Encodings to a Single SDP Formulation 66
9.2 Quantitative Guarantees via Banach Geometry 67
9.3 Beyond Two Layers: Practical Relaxations and Duality 69
9.4 Evaluation 70

IV Incremental Computation for Neural Networks and Language Models 71

10 First-Order Algorithms for LipSDP via Exact-Penalty Eigenvalue Reformulation 74
10.1 Motivation and Transition from Algebraic SDP Bounds 74
10.2 LipSDP: Dual and Primal (Shor) Forms 75
10.3 Exact-Penalty Eigenvalue Reformulation (Single Layer) 76
10.4 Multi-Layer Networks and Vector Outputs 77
10.5 First-Order Implementation: Eigenvalue-Oriented Structure 79
10.6 Evaluation 80
10.7 Summary 82

11 Homotopy Optimization as Incremental Computation: From Spectral Relaxation to Jailbreak Synthesis 83
11.1 Introduction: From Semidefinite Relaxations to Functional Homotopy . . . 83
11.2 Introduction: Incremental Computation in Optimization 84
11.3 Method 87
11.4 Evaluation 93
11.5 Summary 94

12 Conclusion 96

V Appendices 99

A Elided Proofs 100
A.1 Hardness of LLM Input Generation 100

A.2 Linear Approximation of GCG 102

B Empirical Evaluations **104**

B.1 Evaluation of GeoLIP 104

B.2 Evaluation of LipDiff 108

B.3 Evaluation of Functional Homotopy 111

Bibliography **117**

List of Tables

B.1 ℓ_∞-FGL estimation for two-layer networks (MNIST). 106
B.2 Average runtime (seconds) for ℓ_∞-FGL estimation on two-layer networks. . . . 106
B.3 ℓ_∞-FGL estimation for multi-layer networks. 107
B.4 Average runtime (seconds) of GeoLIP for multi-layer networks. 107
B.5 ℓ_2-FGL estimation for two-layer networks. 107
B.6 Average runtime (seconds) for ℓ_2-FGL estimation on two-layer networks. . . . 108
B.7 Neural network models used in evaluation. 109
B.8 Comparison of LipDiff variants with LipSDP and the norm-product baseline. LipDiff-Ex reproduces LipSDP's value closely on MNIST-DNN and substantially reduces estimation error where LipSDP fails to scale. 110
B.9 RBO (0–1) between ground-truth token rankings and alternative rankings. Higher is better. Token-gradient rankings offer only marginal improvement over random. (For adversarial images, analogous RBO values typically exceed 0.90 [54].) 113
B.10 ASR after 500 and 1000 iterations. Mistral and Vicuna saturate by 500 iterations. Although iteration counts are equal, per-iteration cost differs: a GCG iteration (with gradients) is $\approx 85\%$ slower than a GR iteration. For reference, Andriushchenko et al. [3] allow up to 10000 random iterations; we cap at 1000. 114

List of Figures

4.1 A diagram of the polynomial hierarchy, where arrows denote the inclusion relationship . . . 19

5.1 Layerwise representation of a typical feed-forward neural network consisting of several hidden layers. Adjacent layers are connected through affine transformations (matrix multiplications and bias additions), while activation layers apply non-linear functions, enabling the network to model complex relationships. . 33

5.2 Example activation functions. Smooth ReLU (smoothReLU_a) is parameterized by $a > 0$ ($a = 1$ is plotted). . . . 34

5.3 Illustration of δ-Interval Approximation. . . . 37

6.1 A grid illustration . . . 42

6.2 Precision loss θ when using a squashable activation to approximate a step. . . 43

6.3 **Step 2:** Neural construction of $\mathcal{N}_G$. . . . 44

6.4 Slicing $f(x) = sin(2x) + 1$ with approximation tolerance $\delta = 1.2$. . . . 47

7.1 The neural network encoding for $(X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee X_2 \vee X_4)$ . . . 54

11.1 An illustration of the pipeline for the FH application in jailbreak attacks. Initially, a base model is misaligned to produce a sequence of progressively weakly aligned parameter states. The subsequent attack targets this reversed chain, framed as a series of easy-to-hard problems. In this example, the attack begins with twenty "!" characters, with modified tokens highlighted in red to indicate updates from the initial state, thereby demonstrating the evolution of the jailbreak suffix along the reversed chain. . . . 87

11.2 An example of homotopy from $g(x)$ to $f(x)$. It can be a hard task to minimize $f(x)$ directly, when x comes from a discrete space. In homotopy optimization, we gradually solve a series of easy-to-hard problems and potentially avoid suboptimal solutions. Pink balls are the optimal solution to each problem. The path marked by the arrows illustrates the homotopy path over time. 90

11.3 Conceptual illustration of homotopy methods for suffix optimization. (a) Left: Greedy local search heuristic. The red region denotes successful suffixes. The search initiates from a starting point (black solid) and iteratively moves to the optimal neighboring input (dashed circle) based on loss values, potentially leading to local optima entrapment due to non-convexity. (b) Right: Homotopy approach. A series of progressively challenging optimization problems is constructed, with easier problems having larger solution spaces. The solution set gradually converges to that of the original problem. Adjacent problems in this continuum have proximal solutions, facilitating effective neighborhood search. Despite the underlying non-convexity, initiating from a near-optimal point simplifies each problem-solving step. 92

B.1 Optimization trajectories for LipDiff (**blue**) and LipDiff-Rand (**orange**) on MNIST-CNN (a) and CIFAR10-CNN (b). The y-axis (log scale) shows the estimated Lipschitz constant versus iteration. LipDiff converges substantially faster and to a lower value, confirming the effectiveness of analytical initialization. 110

B.2 Iterations to success by method. Bars summarize counts in ≈ 50-iteration bins. FH-GR reaches success more quickly and on more inputs. Note that a GCG iteration is costlier than an FH-GR iteration. 114

B.3 Loss comparison of GCG and FH-GR on “hard” samples. The red FH-GR curve shows the homotopy loss. Homotopy starts with substantially smaller loss due to misalignment, and as FH-GR iterates and successfully jailbreaks intermediate models, it replaces them (as in Algorithm 2) until reaching the base model by iteration 1000. FH-GR converges more quickly than GCG. . . . 115

B.4 Base-model loss comparison of GCG and FH-GR on “hard” samples. Unlike Figure B.3, we evaluate the usefulness of adversarial strings found by FH-GR by applying them to the base model and computing the base loss F_0. The red FH-GR curve indicates the base loss induced by FH-generated inputs; the loss decreases more consistently and converges to a lower value, leading to successful jailbreaks where GCG fails. 115

B.5 Loss comparison of GCG and FH-GR initialized from different checkpoints. We take 25 “hard” samples and initialize FH-GR from earlier checkpoints that are more aligned. Among cases where starting from an earlier checkpoint succeeds (loss in green), FH-GR still converges to a lower loss than GCG. GCG fails on all these cases, whereas FH-GR (ckpt-500→base) succeeds on all, and FH-GR starting from earlier (stronger) checkpoints succeeds on 13 cases for Llama-2 and 6 for Llama-3. 116
B.6 Transferability of successful attacks on the base model to its fine-tuned parameter states. Adversarial strings do not transfer uniformly across models; transfer seems to depend on checkpoint distance and alignment training. 116

Part I

Introduction

Chapter 1

Introduction

1.1 Motivation: Reasoning About AI Systems

Over the past decade, machine learning with neural networks has transformed the landscape of computation. From computer vision [32] to natural language processing [42], and even to program analysis tasks [45], neural networks now serve as the computational backbone for a vast range of applications. Yet, while these systems have demonstrated remarkable empirical performance, our ability to formally analyze, reason about, and predict their behavior remains limited.

This dissertation seeks to bring the lens of *programming language theory*—in particular, its principles of **interpretation**, **duality**, and **incremental computation**—to bear on this challenge. The core thesis is that neural networks and modern AI models can be understood not merely as statistical artifacts but as programmable, compositional, and algebraic structures. Doing so allows us to reinterpret verification, robustness, and adversarial optimization as forms of program reasoning.

1.2 Interpretation: From Pointwise to Set-Based and Symbolic Semantics

A foundational result in neural network theory is the *universal approximation theorem* [24, 14], which states that a neural network with a single hidden layer can approximate any continuous function to arbitrary precision. While this theorem is pointwise in nature, real-world applications often require reasoning about *sets of inputs*, not just individual points.

This motivates a shift from standard input-output semantics to *interval semantics*—a form of **abstract interpretation** [12] that enables reasoning about robustness to perturbations. In this setting, each input feature is modeled as an interval, and the network is interpreted over the space of such intervals. This yields provable robustness properties, provided the network's interval semantics are sufficiently precise [20, 50].

The first part of this dissertation establishes a generalization of the universal approximation theorem to interval semantics: the **Interval Universal Approximation Theorem** (IUA). We show that for a wide class of activation functions (which we call *squashable functions*), neural networks can approximate the set-valued (or collecting) semantics of any continuous function over compact domains to arbitrary precision. This deepens the theoretical foundation of interval-based verification and answers an open question about the expressive power of interval arithmetic in neural networks [5].

Beyond interval-based interpretation, this dissertation also introduces a second, algebraic mode of interpretation. In contrast to the interval perspective, this view treats the internal operations of neural networks—linear transformations, activation functions, and compositions—as symbolic algebraic objects. This **algebraic interpretation framework** allows verification problems to be encoded as algebraic expressions over symbolic variables. These expressions can then be relaxed and solved using tools from optimization, such as quadratically constrained quadratic programs (QCQPs) and SDPs. This symbolic–algebraic encoding plays a central role in the measurement and certification of global network properties, such as Lipschitz continuity.

1.3 Duality: Logical, Functional, and Optimization Perspectives

Across multiple chapters, this dissertation makes essential use of **duality** as a unifying conceptual device.

In the IUA theorem, we encounter *quantifier duality*, where the alternation between existential and universal quantifiers governs the complexity of range approximation. By negating each quantifier, we transition between falsification and certification—naturally leading to min–max formulations and connections with the polynomial hierarchy. We prove that approximating the range of a neural network to high precision is a Δ_2-intermediate problem, situated above NP and coNP.

In the analysis of Lipschitz constants, duality arises through optimization theory. After encoding verification as an algebraic problem, we relax it using convex duality and interpret the resulting semidefinite program as a geometric bound on network behavior. The Lagrangian dual perspective reveals tight connections between symbolic verification and spectral geometry. Moreover, our SDP-based relaxations are closely related to Shor's classical relaxation for non-convex QCQPs [46], highlighting the importance of dual views even in approximate analysis.

We also draw on **functional duality**—a correspondence between inputs and functions—to address adversarial prompting in large language models. In this setting, we study how the space of model parameters influences the distribution of vulnerable inputs. By formulating adversarial objectives as functions over both inputs and parameters, we construct joint functionals that allow us to exploit duality for more tractable optimization.

1.4 Incremental Computation: Approximation as a Trajectory

The third foundational principle of this dissertation is **incremental computation**: the idea that complex computational problems can be solved through a structured progression of simpler, approximate stages. Rather than solving a target problem in its full complexity upfront, incremental computation solves a sequence of related problems, each slightly closer to the final objective. These intermediate problems are designed to be easier to solve and to serve as stepping stones, enabling convergence through refinement.

This principle manifests throughout the dissertation as a strategy of constructing problem trajectories—paths in problem or parameter space—such that the solution to one instance naturally informs the next. Formally, the trajectory is often realized through continuous transformations (e.g., homotopies) or monotonic refinements (e.g., in constraint relaxation or spectral approximation). The key insight is that while the target problem may be computationally intractable, intermediate approximations can be both tractable and informative, enabling efficient progress toward a final solution.

In the context of Lipschitz constant estimation, we apply this principle by reformulating semidefinite programs (SDPs) as nonsmooth eigenvalue optimization problems. This reformulation enables the use of first-order subgradient algorithms that incrementally refine coarse spectral norm estimates into tight SDP-based bounds. The trajectory here moves from efficient but loose heuristics to increasingly accurate relaxations, ultimately achieving certified estimation at a scale suitable for deep models such as ResNet or ImageNet classifiers.

We adopt the same incremental principle in the design of the **functional homotopy** method for adversarial input generation in language models. Here, the core challenge is combinatorial: optimizing discrete token inputs to induce undesired model behavior. Rather than optimizing over discrete inputs directly—a problem that is NP-hard—we instead define a continuous trajectory in the space of model parameters. By perturbing parameters to produce misaligned intermediate models, we induce a sequence of increasingly difficult optimization problems over the input space. The adversarial input is adapted incrementally, starting from a weakly aligned model where attack success is easy, and transferring step by step toward the original well-aligned model. This design can be seen as a form of curriculum-based adversarial optimization, in which both the objective function and the attack trajectory are learned simultaneously through continuous approximation.

1.4.1 Dissertation Contributions and Organization

This dissertation develops a unified programming–language perspective on modern AI systems, organized around three foundational principles: *interpretation*, *duality*, and *incremental computation*. These principles recur across the technical chapters and connect four primary research threads, each based on a published paper.

Interpretation: Interval and Algebraic Semantics. The first strand of contributions develops semantic lenses for neural networks and language models.

- **Interval interpretation and constructive approximation (Part II, Chapters 5 to 7** Building on classical abstract interpretation, we introduce an *interval semantics* for feedforward networks and prove an *Interval Universal Approximation (IUA)* theorem: any continuous target function in the universal approximation class admits a network whose *interval* semantics δ–approximates the target's collecting semantics. We construct such networks explicitly using squashable activations and precisely quantify approximation error. We then show that high–precision interval range approximation is Δ_2–intermediate, strictly harder than both NP and coNP under standard assumptions, clarifying the inherent complexity gap between pointwise and interval–level guarantees. These results are based on our POPL 2022 paper on interval universal approximation and range hardness.

- **Symbolic and algebraic interpretation (Part III, Chapters 8 and 9).** We move from intervals to a precise *algebraic* semantics by symbolically unfolding networks into systems of quadratic constraints. Verification and robustness questions become first–order properties over the reals, yielding a uniform encoding framework for adversarial reachability, global sensitivity, and Lipschitz–style properties. This symbolic-algebraic view underpins the semidefinite relaxations and approximation guarantees developed in Part III.

Duality: Logical, Functional, and Optimization Views. The second strand exploits duality as a unifying conceptual tool.

- **Logical and range dualities (Part II, Chapter 7).** The IUA development exposes a quantifier duality between falsification and certification tasks, leading naturally to min–max formulations and a descriptive–complexity characterization of range approximation as a Δ_2–intermediate problem. This clarifies why the constructive interval approximators exist, yet can be computationally hard to synthesize or certify at fine precision.
- **Optimization and geometric duality (Part III, Chapter 9).** After encoding verification tasks algebraically, we apply convex duality to derive semidefinite relaxations for global Lipschitz constants. These SDPs admit constant–factor guarantees via Grothendieck–type inequalities and have a clear primal–dual interpretation: Shor's relaxation corresponds to Lagrange-multiplier LMIs over slope–restricted activations.
- **Functional duality for adversarial prompting (Part IV, Chapter 11).** For jailbreak attacks on LLMs, we introduce a *functional duality* viewpoint in which both model parameters and token sequences are treated as arguments of a joint functional. This leads to a parameter–space homotopy that trades a single hard discrete search problem for a sequence of easier related problems over progressively weaker models.

Incremental Computation: Trajectories in Problem Space. The third strand develops incremental methods that follow structured trajectories from coarse to precise analyses.

- **Incremental refinement of spectral bound (Part III, Chapter 10).** We present *LipDiff*, a first–order algorithm for semidefinite relaxations of global Lipschitz constants. LipDiff starts from an analytical spectral initialization and incrementally refines bounds via nonsmooth eigenvalue updates, preserving certificate validity at each iterate. This realizes an incremental computation over SDPs: intermediate solutions remain sound while monotonically improving precision.

- **Homotopy optimization for jailbreak synthesis (Part IV, Chapter 11).** We extend incremental computation from convex relaxations to discrete prompt optimization. The *functional homotopy* (FH) method constructs a trajectory in parameter space by "de–robust–training" an aligned model into a sequence of weaker checkpoints, then solving a chain of prompt-optimization subproblems along this path. We show the NP–hardness of model-agnostic input generation, analyze the limited value of token–gradient heuristics in the discrete regime, and demonstrate that FH yields more effective and efficient jailbreaks than GCG, GR, and AutoDAN on several open-source LLMs.

Organization of the Dissertation. The remainder of the dissertation is organized as follows.

- **Part I (Chapters 1 to 4).** Chapter 1 motivates the programming–language viewpoint on neural networks and LLMs and introduces the three principles of interpretation, duality, and incremental computation. Chapter 4 surveys the major technical tools used throughout: descriptive complexity, abstract interpretation, symbolic execution, universal approximation theorems, Lagrangian duality, and semidefinite relaxations.

- **Part II (Chapters 5 to 7).** Chapter 5 develops interval semantics for neural networks and states the IUA theorem. Chapter 6 proves the constructive approximation theorem. Chapter 7 establishes that precise interval range approximation is Δ_2–intermediate.

- **Part III (Chapters 8 and 9).** Chapter 8 introduces algebraic encodings of neural networks. Chapter 9 develops semidefinite relaxations and constant–factor approximation guarantees for global Lipschitz bounds.

- **Part IV (Chapters 10 and 11).** Chapter 10 presents LipDiff as an incremental semidefinite solver for global Lipschitz bounds. Chapter 11 presents functional homotopy as an incremental path method for LLM jailbreak synthesis.

  Together, these two chapters instantiate incremental computation in both discrete and continuous settings.

Together, these parts instantiate the three principles in distinct but connected settings, illustrating how programming–language ideas can be used to interpret, analyze, and incrementally approximate the behavior of modern AI systems.

Chapter 2

Notation and Mathematical Preliminaries

This chapter summarizes the mathematical conventions and foundational tools used throughout the dissertation. The results in later chapters draw on ideas from logic, topology, functional analysis, algebraic geometry, and convex optimization, and the purpose of this chapter is to establish a consistent and self-contained notation system that will support the remainder of the technical development. Although the material is standard, the organization reflects the needs of subsequent chapters, particularly those emphasizing interval semantics, algebraic encodings, and incremental optimization.

2.1 Sets, Logic, and Linear Algebra

- $\forall x \in S$ and $\exists x \in S$ denote universal and existential quantification, respectively. Logical negation is written $\neg P$.
- $\min_x f(x)$ denotes the minimum value of f, and $\arg\min_x f(x)$ the set of minimizers. Analogous notation applies to max and $\arg\max$.
- $[n] = \{1, 2, \ldots, n\}$ is the set of the first n positive integers. We work over the real numbers $\mathbb{R}$ by default.
- $\mathbb{R}_+ = [0, \infty)$ denotes nonnegative reals, $\mathbb{Z}_+$ positive integers, and $\mathbb{N} = \{0\} \cup \mathbb{Z}_+$ the nonnegative integers.
- A matrix $A \in \mathbb{R}^{m\times n}$ has entries A_{ij}; symmetric matrices $A \in \mathbb{S}^n$ satisfy $A^\top = A$.

- For $v \in \mathbb{R}^n$, $\operatorname{diag}(v)$ denotes the diagonal matrix whose diagonal entries are those of v. The identity matrix is $I_n = \operatorname{diag}(e_n)$ where $e_n = (1, \dots, 1)$.

These conventions support the algebraic network encodings and semidefinite formulations developed in Parts III and IV.

2.2 Topology

- Inputs are vectors $x \in \mathbb{R}^n$; functions are maps $f : \mathbb{R}^n \to \mathbb{R}^m$.
- A metric space is a pair (X, d) with $X \subseteq \mathbb{R}^n$ and $d : X \times X \to \mathbb{R}$ satisfying nonnegativity, symmetry, and the triangle inequality.
- Continuity of $f : X \to Y$ requires that small changes in x lead to small changes in $f(x)$, formalized via ϵ–δ definitions.
- A homotopy between functions f_0 and f_1 is a continuous deformation $H : X \times [0, 1] \to Y$ with $H(\cdot, 0) = f_0$ and $H(\cdot, 1) = f_1$. Homotopies later inform the construction of functional trajectories in Chapter 11.
- A map f is Lipschitz continuous with constant K if $d_Y(f(x_2), f(x_1)) \leq K\, d_X(x_2, x_1)$. Lipschitz constants play a central role in robustness analysis and in the incremental semidefinite method of Chapter 10.

2.3 Functional Analysis

- A *functional* is a map $\phi : \mathbb{R}^n \to \mathbb{R}$. The dual space $(\mathbb{R}^n)^*$ is the set of all linear functionals. Evaluating a function at a point is itself a functional, providing a basic instance of the functional dualities discussed in Chapter 11.
- Vector norms $\|v\|_p = (\sum_i |v_i|^p)^{1/p}$ induce metrics and support notions of continuity and approximation. The ℓ_2 norm is Euclidean; ℓ_∞ is the maximum norm.
- The operator norm $\|A\|_{p \to r} = \max_{\|x\|_p = 1} \|Ax\|_r$ quantifies how a matrix distorts space and underlies Lipschitz constants of neural networks.
- The inner product $\langle u, v \rangle = \sum_i u_i v_i$ and the Frobenius inner product $\langle A, B \rangle_F = \operatorname{tr}(A^\top B)$ are used extensively in the SDP relaxations in Chapter 9.

- A matrix A is positive semidefinite (PSD), written $A \succeq 0$, if $x^\top Ax \geq 0$ for all x. PSD matrices form the feasible region of SDPs.
- $B_p(x, \epsilon) = \{y \mid \|y - x\|_p < \epsilon\}$ denotes an ℓ_p ball. When p is omitted, context determines the default.
- For $p \geq 1$, its Hölder conjugate q satisfies $1/p + 1/q = 1$.

2.4 Differentiation and Gradients

- For $f : \mathbb{R} \to \mathbb{R}$, the derivative is
$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$
- For $f : \mathbb{R}^n \to \mathbb{R}$, the partial derivative with respect to x_i is
$$\frac{\partial f}{\partial x_i}(x) = \lim_{h \to 0} \frac{f(x + he_i) - f(x)}{h}.$$
- The gradient is the vector of partial derivatives:
$$\nabla f(x) = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right]^\top.$$
- For $f : \mathbb{R}^n \to \mathbb{R}^m$, the Jacobian $J_f(x)$ collects partial derivatives $\frac{\partial f_i}{\partial x_j}$. For scalar-valued f, the Hessian $H_f(x)$ is the matrix of second derivatives. These derivatives are central to first- and second-order optimization, including the gradient-based methods contrasted with functional homotopy.

2.5 Algebraic Preliminaries

- Variables represent unknown quantities. A monomial is a product of variables with nonnegative integer exponents.
- A polynomial is a finite linear combination of monomials with real coefficients. A polynomial is quadratic if each monomial has degree at most two.
- Quadratic functions can be written as $x^\top Qx + 2b^\top x + c$ with Q symmetric. Such expressions form the backbone of the quadratic encodings of neural networks in Chapter 8.

- A set is semialgebraic if it is a finite Boolean combination of polynomial equalities and inequalities. Many neural-network components—ReLU, max pooling, GroupSort—admit semialgebraic descriptions.

2.6 Optimization: Objectives and Constraints

We consider constrained optimization problems of the form

$$\min_{x \in \mathsf{X}} f(x) \quad \text{s.t.} \quad g_i(x) \leq 0 \ (i = 1, \ldots, m), \qquad h_j(x) = 0 \ (j = 1, \ldots, k),$$

where f is the objective and g_i, h_j are inequality and equality constraints.

Quadratically Constrained Quadratic Programs (QCQPs). These optimize a quadratic objective subject to quadratic constraints:

$$\min_x \ x^\top A_0 x + 2b_0^\top x + c_0 \quad \text{s.t.} \quad x^\top A_i x + 2b_i^\top x + c_i \leq 0.$$

Quadratic formulations arise naturally from the algebraic encodings in Part III.

Semidefinite Programs (SDPs). These are convex problems over PSD matrices:

$$\min_{X \succeq 0} \langle C, X \rangle_F \quad \text{s.t.} \quad \langle A_i, X \rangle_F = b_i.$$

SDPs play a central role in the Lipschitz analysis of neural networks and motivate the incremental first-order method developed in Chapter 10.

Chapter 3

Neural Networks as Programs: Testing and Verification

This chapter explores the perspective of treating neural networks and language models as programs and studies their behavior under the lens of program testing and verification. It introduces foundational testing and verification concepts, discusses neural networks and language models from both informal and formal perspectives, and highlights adversarial robustness and jailbreak attacks. We conclude by formalizing the associated verification and attack problems as optimization problems.

3.1 Testing and Verification: From Software to AI Systems

Program testing aims to identify concrete inputs that cause a system to exhibit undesirable, incorrect, or unintended behaviors—commonly referred to as bugs. This process is empirical in nature: it provides evidence of failure by demonstrating counterexamples but cannot guarantee their absence. Verification, by contrast, seeks to provide formal guarantees that such inputs do not exist under a given specification. That is, it attempts to prove that a program or system behaves correctly for *all* inputs within a specified domain. These two approaches—testing and verification—are fundamental in classical software engineering, where they serve as complementary tools for ensuring correctness, reliability, and robustness.

This distinction extends naturally to modern AI systems, especially those based on deep learning. *Testing* of AI models involves identifying specific adversarial inputs, distributional failures, or decision inconsistencies that violate expected behavior. In adversarial learning, for example, testing takes the form of crafting perturbations or prompts that induce misclassification or policy failure. *Verification*, on the other hand, involves proving properties of models—such as robustness, fairness, or bounded outputs—under all admissible inputs, often through formal analysis or relaxation techniques. These guarantees are particularly important in safety-critical settings such as autonomous driving, medical diagnosis, or legal reasoning, where empirical performance alone is insufficient.

Quantifier Duality.. At the heart of the distinction between testing and verification lies a fundamental logical duality:

$$\text{Testing:} \quad \exists x \in \mathsf{X} \text{ such that } P(x) \text{ fails}, \qquad \text{Verification:} \quad \forall x \in \mathsf{X},\ P(x) \text{ holds}.$$

This quantifier duality captures the tension between finding counterexamples (existential) and establishing universal correctness. In practice, many AI verification problems reduce to the negation of a testing claim: proving that no adversarial inputs exist within some bounded domain amounts to discharging a universal quantifier. Conversely, every failed verification attempt typically yields a test case—a constructive witness to the failure. This interplay highlights a deeper duality not just in methodology, but in the logical structure of reasoning about computation. One may view verification as generalized testing over infinite domains, and testing as failed attempts to falsify a verification claim. Much of this dissertation operates within this logical tension, exploiting structural properties to reduce verification to tractable testing instances, or vice versa.

3.2 Neural Networks as Parameterized Programs

Deep learning systems, particularly neural networks and large-scale language models, have transformed numerous application domains including computer vision, natural language processing, robotics, and scientific computing. These models achieve remarkable performance by learning complex, high-dimensional functions from data, often surpassing traditional hand-engineered solutions. Despite their empirical success, their internal behavior and generalization properties remain difficult to characterize analytically.

From a formal perspective, neural networks can be interpreted as parameterized numerical programs whose semantics arise from their structure (e.g., layers, activations, attention blocks) and from optimization-based training procedures. Although they are not constructed via explicit code in the traditional sense, their behavior can still be analyzed using tools from programming languages, logic, and formal methods. This abstraction allows us to apply concepts such as interpretation, compositionality, symbolic execution, and approximation to reason about neural computations systematically.

Mathematically, a neural network defines a function $f_p : \mathbb{R}^n \to \mathbb{R}^m$, where $p \in \mathbb{R}^d$ denotes the parameter vector (e.g., weights and biases) and $x \in \mathbb{R}^n$ is the input. The output $f_p(x)$ may represent a predicted class, a regression output, or—for more complex tasks—a structured object such as an image or sequence. The network's semantics are thus determined not only by its architecture but also by the learned parameters p, which encode data-dependent inductive biases shaped through training.

Language models, particularly autoregressive transformers, extend this functional view by incorporating sequential structure and probabilistic inference. These models can be formalized as stochastic or deterministic programs that map finite sequences of tokens to distributions over the next token. Formally, given a token sequence $x = (x_1, \ldots, x_t)$, a language model defines a conditional probability distribution $P_p(x_{t+1} \mid x_{1:t})$ computed via a forward pass through the model parameterized by p. Generation proceeds iteratively: each new token is sampled (or decoded deterministically) based on prior context and updated memory states. This iteration defines a stateful computation with both functional and generative semantics.

Crucially, viewing neural networks and language models as parameterized programs opens the door to formal reasoning—verification, interpretation, and abstraction—previously reserved for traditional code. Throughout this dissertation, we leverage this perspective to apply principles from programming language theory to the analysis, approximation, and verification of learned systems.

3.3 Adversarial Robustness and Jailbreak Attacks

Neural networks and language models are known to exhibit vulnerabilities to carefully constructed inputs that cause them to produce erroneous, unexpected, or even harmful outputs. In the image domain, such vulnerabilities manifest as *adversarial examples*—inputs that are visually or semantically indistinguishable from valid data but lead to incorrect classifications or decisions. These perturbations, often constrained by an ℓ_p norm to ensure imperceptibility, exploit the local linearity or high sensitivity of the model's decision boundary.

In the context of language models, related concerns emerge in the form of *jailbreak attacks*. These are specially crafted prompts designed to circumvent alignment constraints or safety filters, prompting the model to generate outputs that would otherwise be prohibited, such as hate speech, misinformation, or instructions for illegal activities. Unlike adversarial attacks in continuous domains, jailbreak attacks operate over discrete token sequences, making the optimization problem combinatorially harder and often requiring more sophisticated or indirect methods.

These phenomena pose a serious challenge to the integrity, safety, and trustworthiness of modern AI systems. They highlight the gap between training-time objectives and deployment-time behaviors, revealing a lack of robustness in the face of adversarial or out-of-distribution inputs. Moreover, they expose the limitations of current alignment and safety mechanisms, particularly in large-scale language models deployed in open-ended environments.

Understanding and formalizing adversarial vulnerabilities within the framework of programming languages and formal methods provides a principled lens for analysis. From this viewpoint, such attacks correspond to finding inputs that drive a program (the model) into failure modes or undesirable execution paths—analogous to test cases that expose bugs in traditional software. By adopting this perspective, we can leverage tools such as symbolic reasoning, optimization, verification, and abstraction to analyze and strengthen the behavior of AI systems under adversarial conditions.

3.4 Formal Structures of Neural Network Programs

Neural networks and language models can be viewed as parametric computational programs, with architectures composed of layers and semantics derived from forward execution over learned parameters. This structural interpretation forms the foundation for formal reasoning, verification, and adversarial analysis.

3.4.1 Neural Network Architecture

A neural network defines a parameterized function $f_p : \mathbb{R}^n \to \mathbb{R}^m$ as a composition of layers:

$$f_p(x) = L_k(\cdots L_2(L_1(x; p_1); p_2) \cdots ; p_k),$$

where each L_i is typically an affine transformation followed by a nonlinearity, and p_i denotes its trainable parameters (weights and biases). The total parameter vector $p = (p_1, \ldots, p_k)$ defines the learned function over the input domain.

Common activation functions include:

- **ReLU**: $\mathrm{ReLU}(x) = \max(0, x)$,

- **Sigmoid**: $\sigma(x) = \frac{1}{1+e^{-x}}$,
- **Softmax**: $\mathrm{softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$.

Modern architectures incorporate structural enhancements such as residual connections, attention mechanisms, normalization layers, and convolutional modules. These design patterns enhance expressiveness and trainability across domains such as vision, language, and control.

3.4.2 Classification Models

In classification tasks, the neural network outputs a vector of logits $f_p(x) \in \mathbb{R}^m$, interpreted as unnormalized scores for each class. The predicted label is then computed as:

$$\hat{y}(x) = \arg\max_i f_p(x)_i.$$

During training, the logits are often normalized via softmax, and the network is optimized to minimize cross-entropy loss against the true class labels. This decision rule induces a piecewise-linear decision boundary, whose geometric and algorithmic properties are central to many verification and adversarial robustness problems.

3.4.3 Autoregressive Language Models

Autoregressive models, such as transformer-based language models, generate text one token at a time by modeling conditional distributions. Given a prefix $x = (x_1, \ldots, x_t)$ from a vocabulary $\mathcal{V}$, the model outputs:

$$P(x_{t+1} \mid x_1, \ldots, x_t) = \mathrm{softmax}(f_p(x)),$$

where $f_p(x)$ returns logits over $\mathcal{V}$, and generation proceeds via:

$$x_{t+1} = \arg\max_{v \in \mathcal{V}} f_p(x)_v.$$

This process continues autoregressively, with the output at each step appended to the input sequence. The language model thus behaves as a stateful probabilistic program, where transitions are governed by learned parameters and prefix context. These structural and probabilistic features are crucial for modeling alignment and designing jailbreak attacks.

3.5 Optimization Formulations

The robustness and security of neural systems can be analyzed through the lens of formal optimization. A central concern in classification is the existence of *adversarial examples*: perturbed inputs $x' = x + \delta$ satisfying

$$\|\delta\| \leq \epsilon \quad \text{and} \quad \hat{y}(x') \neq \hat{y}(x),$$

where $\hat{y}(x) = \arg\max_i f_p(x)_i$. These inputs expose the model's vulnerability by inducing incorrect outputs despite being close to natural examples under some norm. The underlying issue lies in the fragility of the learned decision boundaries in high-dimensional spaces.

More broadly, many verification, testing, and alignment problems can be formulated as optimization tasks. These include:

- **Adversarial robustness testing:**

  $$\exists x' \in B(x, \epsilon) : \hat{y}(x') \neq \hat{y}(x),$$

  where the goal is to find a perturbation that causes misclassification.

- **Formal verification:**

  $$\forall x' \in B(x, \epsilon) : \hat{y}(x') = \hat{y}(x).$$

  This universally quantified condition asserts that the prediction is invariant under norm-bounded perturbations.

- **Jailbreak prompt synthesis:**

  $$\min_{x \in \mathcal{X}_{\text{prompt}}} f_p(x),$$

  where $f_p(x)$ is an alignment or safety score evaluated over a space of syntactically valid prompts. Here, the objective is to generate inputs that induce harmful or misaligned completions in language models.

These formulations unify the landscape of AI robustness under a common optimization-based framework, spanning both continuous and discrete domains. They highlight the interplay between logic (via quantifiers), geometry (via norm constraints), and semantics (via output behavior). Addressing these problems often requires a combination of relaxation techniques, abstraction layers, and incremental optimization strategies—core themes explored throughout this dissertation.

Chapter 4

Major Technical Tools

This chapter surveys the fundamental theoretical and computational tools that underpin our approach. We organize the discussion into five key sections.

4.1 Descriptive Power and Complexity of Logical Forms

A central theme in theoretical computer science is the connection between logic and computation. Logical formulas—built from quantifiers, Boolean connectives, and arithmetic predicates—provide a powerful framework for specifying computational problems. The complexity of a problem often correlates with the structure of the logical formula that defines it, particularly the number and alternation of quantifiers.

4.1.1 Descriptive Complexity and the Polynomial Hierarchy

Descriptive complexity theory formalizes this connection by associating computational complexity classes with classes of logical formulas. Specifically:

- Problems definable by *existential formulas* of the form $\exists x_1, \ldots, x_k \ \phi(x_1, \ldots, x_k)$, where ϕ is computable in polynomial time, correspond to NP.
- Problems definable by *universal formulas* of the form $\forall x_1, \ldots, x_k \ \phi(x_1, \ldots, x_k)$, where ϕ is computable in polynomial time, correspond to coNP.

This relationship extends naturally to alternating quantifiers:

$$\phi(x) = \exists y_1 \forall y_2 \exists y_3 \cdots Q_n y_n \ \psi(x, y_1, \ldots, y_n),$$

$$\begin{tikzcd}
\Sigma_n & \vdots & \Pi_n \\
& \Delta_n \arrow[ul] \arrow[ur] & \\
\Sigma_2 \arrow[uu, dashed] \arrow[ur, dashed] & & \Pi_2 \arrow[uu, dashed] \arrow[ul, dashed] \\
& \Delta_2 \arrow[ul] \arrow[ur] & \\
\Sigma_1 = \mathsf{NP} \arrow[uu] \arrow[ur] & & \mathsf{coNP} = \Pi_1 \arrow[uu] \arrow[ul] \\
& \Sigma_0 = \mathsf{P} = \Pi_0 \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Figure 4.1: A diagram of the polynomial hierarchy, where arrows denote the inclusion relationship

where $Q_n = \exists$ if n is odd and $Q_n = \forall$ if n is even, and ψ is polynomial-time decidable.

Bounding each quantified variable to be polynomial in the input size $|x|$ yields the **polynomial hierarchy** (PH), which generalizes NP and coNP. Each level of PH corresponds to a family of polynomial-time Turing machines with alternating quantifiers.

Definition 4.1.1 (The Σ_1 and Π_1 classes)**.** A language L is in Σ_1 (NP) if there exist a polynomial-time Turing machine M and a polynomial q such that

$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} : M(x, u_1) = 1.$$

Similarly, L is in Π_1 (coNP) if there exist a polynomial-time Turing machine M and a polynomial q such that

$$x \in L \iff \forall u_1 \in \{0,1\}^{q(|x|)} : M(x, u_1) = 1.$$

■

The hierarchy continues with alternating quantifiers, defining Σ_k and Π_k families as follows:

$$\Sigma_k^P = \{\exists u_1 \forall u_2 \cdots Q_k u_k : M(x, u_1, \ldots, u_k) = 1 \text{ for some poly-time } M\},$$
$$\Pi_k^P = \{\forall u_1 \exists u_2 \cdots Q_k u_k : M(x, u_1, \ldots, u_k) = 1 \text{ for some poly-time } M\}.$$

The hierarchy is believed to be strict: $\Sigma_k^P \subsetneq \Sigma_{k+1}^P$ unless it collapses at some level.

We focus on the first two levels of this hierarchy. Σ_2 and Π_2 are defined formally below.

Definition 4.1.2 (The Σ_2 class)**.** A language L is in Σ_2 if there exist a polynomial-time Turing machine M and a polynomial q such that

$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)}\ \forall u_2 \in \{0,1\}^{q(|x|)} :\ M(x, u_1, u_2) = 1.$$

■

Definition 4.1.3 (The Π_2 class)**.** A language L is in Π_2 if there exist a polynomial-time Turing machine M and a polynomial q such that

$$x \in L \iff \forall u_1 \in \{0,1\}^{q(|x|)}\ \exists u_2 \in \{0,1\}^{q(|x|)} :\ M(x, u_1, u_2) = 1.$$

■

Definition 4.1.4 (The Δ_2 class)**.** $\Delta_2 = \Sigma_2 \cap \Pi_2$. ■

Note that $\mathsf{NP}, \mathsf{coNP} \subseteq \Delta_2$, since one can substitute an empty string for u_1 or u_2 in Definitions 4.1.2 and 4.1.3. The polynomial hierarchy is the union of all Σ_n languages.

Definition 4.1.5 (Δ_2-intermediate language)**.** A set of languages $\mathbb{L}$ is Δ_2-*intermediate* if $\mathsf{NP} \cup \mathsf{coNP} \subseteq \mathbb{L}$ and $\mathbb{L} \subseteq \Delta_2$. ■

Remark. By definition, $\Delta_2 = \Pi_2 \cap \Sigma_2$, and $\mathsf{NP} \cup \mathsf{coNP} \subseteq \Delta_2$. It is unknown whether $\mathsf{NP} \cup \mathsf{coNP} = \Delta_2$. However, if $\mathsf{coNP} \not\subseteq \mathsf{NP}$ as commonly believed, then $\mathsf{NP} \subsetneq \mathbb{L}$ whenever $\mathbb{L}$ is Δ_2-intermediate.

4.1.2 From Quantifiers to Optimization

Quantifiers can be interpreted in terms of extremal optimization:

- $\exists y\ \phi(x, y)$ corresponds to $\max_y \phi(x, y)$ when ϕ is Boolean-valued.
- $\forall y\ \phi(x, y)$ corresponds to $\min_y \phi(x, y)$.

Thus, logical formulas with alternating quantifiers naturally correspond to multi-level optimization problems, such as:

$$\max_{y_1} \min_{y_2} \phi(x, y_1, y_2),$$

where ϕ evaluates to 0 or 1 depending on whether a constraint is satisfied. This mirrors strategic games and verification problems with adversarial or uncertain elements.

Notes. The polynomial hierarchy can be viewed as a resource-bounded analogue of the *arithmetical hierarchy* from computability theory, which classifies decision problems based on unbounded quantifier alternation over natural numbers. However, the arithmetical hierarchy involves semantic definitions over infinite structures (e.g., Peano arithmetic), whereas the polynomial hierarchy restricts attention to finite strings with polynomially bounded quantifiers.

Because the logical formulas used in PH describe finite structures and are bounded in size, they are more amenable to algorithmic interpretation and verification. The concepts from both hierarchies form the logical and computational backbone for characterizing expressiveness and tractability in program analysis and AI verification.

4.2 Abstract Interpretation

Abstract interpretation is a general framework for soundly and approximately reasoning about program semantics [12]. Instead of tracking exact program executions, it uses *abstract domains* to represent sets of states in a tractable way. This allows scalable static analysis while ensuring that results overapproximate real behaviors.

4.2.1 Concrete and Abstract Domains

Let C denote the *concrete domain* of program states (e.g., $\mathcal{P}(\mathbb{R}^n)$) and A an *abstract domain* representing summaries (e.g., intervals, boxes, polyhedra). Abstract interpretation connects them via:

- **Abstraction** $\alpha : C \to A$, mapping precise sets to safe abstract summaries.
- **Concretization** $\gamma : A \to C$, mapping abstract summaries to the sets they describe.

These maps define a *Galois connection* $(C, \sqsubseteq_C) \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} (A, \sqsubseteq_A)$ if:

$$\alpha(c) \sqsubseteq_A a \quad \text{iff} \quad c \sqsubseteq_C \gamma(a).$$

This ensures that all behaviors of c are contained in the interpretation of any abstract element a above $\alpha(c)$—guaranteeing soundness.

4.2.2 Soundness and Abstract Semantics

Soundness requires that all abstract operations safely overapproximate their concrete counterparts. For example, an abstract transformer $F^{\#} : A \to A$ corresponding to a concrete semantics $F : C \to C$ must satisfy:

$$F(c) \subseteq \gamma(F^{\#}(\alpha(c))).$$

This ensures that if the abstract semantics prove a property, it holds for all concrete executions.

Notes. In general settings involving recursion or loops, program invariants are computed using fixpoints of transfer functions, often accelerated via *widening* operators. These techniques are essential for ensuring termination of abstract analysis over infinite ascending chains. However, in this dissertation, we do not consider recurrent structures. Therefore, fixpoint iteration and widening are not required in our applications.

4.2.3 Applications in Neural Network Verification

Abstract interpretation has been adapted to certify properties of neural networks—such as output bounds under input perturbation—by propagating abstract input sets through the layers of the network [20]. When using interval or zonotope abstractions, each layer is transformed into an abstract transformer that overapproximates the reachable output space.

In later chapters, we instantiate this framework using interval-based abstract domains to verify the robustness of neural networks. Abstract interpretation in this setting serves as a structured form of symbolic execution that trades precision for scalability while preserving provable safety.

4.3 Linear Approximation and Lipschitz Regularity

A central question in adversarial robustness is whether a model's output remains stable under small perturbations of its input. That is, given an input x, does the model produce nearly identical predictions for inputs $x' \approx x$? This question centers on the *local stability* of the learned function f, which can be studied using tools from analysis—beginning with linear approximation.

4.3.1 Local Linear Approximation

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function. Around a point $a \in \mathbb{R}^n$, the first-order Taylor expansion yields the linear approximation:

$$f(a + \Delta x) \approx f(a) + \nabla f(a)^\top \Delta x, \tag{4.1}$$

where $\nabla f(a)$ is the gradient of f at a and Δx is a small displacement vector. This expression becomes exact when f is affine and serves as a local surrogate for general smooth functions.

The approximation quality depends on two factors:

- The magnitude of the perturbation $\|\Delta x\|$;
- The curvature or nonlinearity of f in the neighborhood of a.

When Δx is small and f does not change rapidly, the linear term dominates, and the approximation is accurate. This forms the basis for gradient-based analysis of local robustness.

4.3.2 Bounding Variation via Gradient Norms

Linear approximation suggests that the output deviation $|f(a + \Delta x) - f(a)|$ is governed by the inner product $\nabla f(a)^\top \Delta x$. By Hölder's inequality, this deviation can be bounded:

$$|f(a + \Delta x) - f(a)| \leq \|\nabla f(a)\|_q \cdot \|\Delta x\|_p,$$

where p and q are dual norms ($1/p + 1/q = 1$). This observation leads directly to a quantitative notion of stability—Lipschitz continuity.

4.3.3 Lipschitz Continuity and Rademacher's Theorem

Recall that a function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *Lipschitz continuous* with respect to the ℓ_p norm if there exists a constant L_f such that

$$|f(x) - f(x')| \leq L_f \cdot \|x - x'\|_p \quad \text{for all } x, x' \in \mathbb{R}^n.$$

Lipschitz continuity implies global control over the function's sensitivity to input perturbations. Crucially, Rademacher's theorem justifies using gradients to characterize this behavior:

Theorem 4.3.1 (Rademacher's Theorem)**.** *If $f : \mathbb{R}^n \to \mathbb{R}$ is Lipschitz with respect to ℓ_p, then f is differentiable almost everywhere, and the Lipschitz constant satisfies:*

$$L_f = \sup_x \|\nabla f(x)\|_q,$$

where q is the Hölder conjugate of p.

Thus, in regions where f is differentiable, the gradient norm directly bounds how much f can change. This insight allows us to transform robustness verification into a gradient-based certification problem.

4.3.4 Implications for Robustness Certification

For adversarial robustness, the key goal is to certify that the output remains stable within a bounded perturbation region $B(x, \epsilon) = \{x' : \|x' - x\|_p \leq \epsilon\}$. If one can verify that:

$$\sup_{x' \in B(x,\epsilon)} \|\nabla f(x')\|_q \leq \tau,$$

then it follows that for all $x' \in B(x, \epsilon)$:

$$|f(x') - f(x)| \leq \tau \cdot \epsilon.$$

In classification, this bound can be used to certify that the predicted label does not change within $B(x, \epsilon)$, provided the margin between the top class and others is greater than $\tau \cdot \epsilon$. Hence, Lipschitz bounds and gradient norms offer both theoretical and practical tools for certifying robustness in machine learning systems.

4.4 Algebraic Formulations of Decision Problems

Many verification and synthesis problems in AI can be naturally encoded as questions about real-valued variables subject to polynomial equalities and inequalities. These lead to a rich class of sets known as *semialgebraic*, which form the algebraic foundation of many decision problems.

4.4.1 Semialgebraic Sets

A *semialgebraic set* in $\mathbb{R}^n$ is a subset defined by a finite Boolean combination (using $\wedge$, $\vee$, and $\neg$) of polynomial equations and inequalities. Examples include:

- $\{x \in \mathbb{R}^n \mid P(x) = 0\}$,
- $\{x \in \mathbb{R}^n \mid Q(x) > 0\}$,
- Finite unions and intersections of the above.

Proposition 4.4.1 ([11]). *Every semialgebraic set is a finite union of basic semialgebraic sets of the form:*

$$\{x \in \mathbb{R}^n \mid P(x) = 0, Q_1(x) > 0, \ldots, Q_\ell(x) > 0\},$$

where P, Q_i are polynomials with real coefficients.

Notes. A function is semialgebraic if its graph is a semialgebraic set. This includes many activation functions used in neural networks, such as ReLU and piecewise linear splines.

4.4.2 Model-Theoretic View and Decidability

From the perspective of logic and model theory, semialgebraic sets are precisely those definable in the first-order theory of real closed fields, using formulas over the signature $\mathcal{L}_{\text{RCF}} = \{+, \cdot, <, 0, 1\}$. This allows semantic reasoning about sets defined via polynomial inequalities.

Theorem 4.4.2 (Tarski–Seidenberg Quantifier Elimination [47]). *The first-order theory of real closed fields admits quantifier elimination and is decidable. Consequently, any definable set in this theory corresponds exactly to a semialgebraic set.*

This foundational result implies that any logical formula over the reals involving polynomial terms and quantifiers (e.g., $\exists x\ \forall y\ P(x, y) > 0$) can be algorithmically reduced to an equivalent quantifier-free form. This underpins the decidability of many problems in real algebraic geometry and formal verification.

Notes. Semialgebraic sets belong to the broader framework of *tame topology*, which studies topological spaces that avoid pathological behaviors and exhibit finite geometric complexity. These sets are closed under projection, possess finitely many connected components, and behave predictably under logical operations. Their well-behaved structure makes them ideal for formal reasoning, enabling powerful results from real algebraic geometry to be applied in verification, optimization, and symbolic computation.

In later chapters, we will use these properties to justify the expressiveness and tractability of formulations such as QCQPs, which define semialgebraic feasible regions encoding properties of neural networks and their verification objectives.

4.5 Lagrangian Duality and Shor's Relaxation

Lagrangian duality. Given a constrained problem:

$$\min f(x) \quad \text{s.t. } g_i(x) \le 0,\ h_j(x) = 0,$$

we define the Lagrangian:

$$\mathcal{L}(x, \lambda, \nu) = f(x) + \sum \lambda_i g_i(x) + \sum \nu_j h_j(x),$$

and the dual problem is:

$$\max_{\lambda \ge 0, \nu} \inf_x \mathcal{L}(x, \lambda, \nu).$$

Under suitable regularity (e.g., Slater's condition), strong duality holds and the Karush–Kuhn–Tucker (KKT) conditions characterize optimality:

- Primal feasibility: $g_i(x^*) \le 0$, $h_j(x^*) = 0$;
- Dual feasibility: $\lambda_i \ge 0$;
- Complementary slackness: $\lambda_i g_i(x^*) = 0$;
- Stationarity: $\nabla f(x^*) + \sum \lambda_i \nabla g_i(x^*) + \sum \nu_j \nabla h_j(x^*) = 0$.

Semidefinite programming (SDP). An SDP has the form:

$$\min \langle C, X \rangle \ \text{ s.t. } \langle A_i, X \rangle = b_i,\ X \succeq 0,$$

with dual:

$$\max_v \sum v_i b_i \ \text{ s.t. } C - \sum v_i A_i \succeq 0.$$

These convex programs are solvable in polynomial time via interior point methods [6].

Shor's relaxation of QPs. Let $x^T A x + 2b^T x + c$ be a QP. Define:

$$X(x) = \begin{pmatrix} 1 \\ x \end{pmatrix} \begin{pmatrix} 1 \\ x \end{pmatrix}^T, \quad \bar{A}_i = \begin{pmatrix} c_i & b_i^T \\ b_i & A_i \end{pmatrix}.$$

Then the relaxed SDP is:

$$\min_X \left\{ \langle \bar{A}_0, X \rangle : \langle \bar{A}_i, X \rangle \le 0,\ X \succeq 0,\ X_{11} = 1 \right\}.$$

This lifting drops the nonconvex rank-1 constraint, enabling polynomial-time approximation of NP-hard problems such as MAXCUT [21].

Connections to geometry and approximation. SDP relaxations link to Banach space theory and have been used in approximability, hardness of approximation, and cut norms [7, 30, 44, 1]. They now underpin many modern optimization and verification techniques.

Part II

Interval Universal Approximation

This part develops the first major theme of the dissertation: *interpretation*, focusing on how neural networks behave under *set-based* rather than pointwise semantics. While classical universal approximation theorems articulate the expressiveness of neural networks at individual inputs, many robustness, verification, and safety questions require reasoning about *regions* of inputs—entire boxes of possible perturbations, uncertainty sets, or abstract domains. This shift from pointwise semantics to *interval semantics* brings both new expressive possibilities and fundamental computational challenges.

Interval semantics arises from abstract interpretation: instead of evaluating a network on a concrete input x, one propagates an axis-aligned box B through the layers of the network to obtain an overapproximation of all outputs reachable from any input in B. This perspective underlies a wide range of robustness and verification techniques, especially for ℓ_∞ perturbations, where the uncertainty region around an input is itself a box. Despite its practical appeal, little was previously understood about the theoretical limits of interval semantics: What functions can neural networks represent under interval abstraction? How does interval expressiveness compare with classical pointwise approximation? And, crucially, what is the computational complexity of synthesizing or analyzing interval-approximating networks?

Part II answers these questions through the *Interval Universal Approximation (IUA) Theorem*, which generalizes classical universal approximation to set semantics. We show that for a broad class of *squashable* activation functions, neural networks can approximate the *collecting semantics* of any continuous function over all input boxes to arbitrary precision. The proof requires constructing networks whose *abstract* behavior matches that of the target function—a substantially stronger requirement than matching pointwise values.

The second half of the part investigates the computational limits of such interval approximators. Leveraging the duality between $\exists$ and $\forall$ quantifiers, we show that the fundamental range-approximation subproblem is Δ_2-intermediate and therefore strictly harder than both NP and coNP unless the polynomial hierarchy collapses. This hardness persists even for very small approximation tolerances ($\delta < 1/2$) and even when the target function is itself a neural network. These results explain why interval-based verification methods often face precision bottlenecks: although interval approximators always exist, computing them to high fidelity or even deciding their range is provably difficult.

Structure of Part II. Chapter 5 introduces interval semantics and formalizes the IUA theorem. It defines squashable activations, set semantics, and interval abstraction, and states the positive universal approximation result. Chapter 6 provides a constructive proof of the IUA theorem, building networks from indicator-like components and carefully controlling approximation error under interval semantics. Chapter 7 establishes the computational hardness of interval approximation via reductions from SAT and its dual forms, showing that range approximation is Δ_2-intermediate.

Together, these chapters develop interval semantics from first principles, clarify the expressive power of interval abstraction, and delineate the inherent computational limits of interval-based reasoning. They form the interpretive foundation for the subsequent parts of the dissertation.

Chapter 5

The Interval Universal Approximation: Introduction

5.1 Motivation

A central notion in adversarial robustness is robustness to ℓ_∞ perturbations: for a given input $x \in \mathbb{R}^n$, a classifier f is said to be ℓ_∞-robust at x if, for all $x' \in \mathbb{R}^n$ satisfying $\|x' - x\|_\infty \leq \epsilon$, the predicted class remains unchanged:

$$\arg\max_i f(x')_i = \arg\max_i f(x)_i.$$

The ℓ_∞ ball $B_\infty(x, \epsilon) = \{x' \in \mathbb{R}^n \mid \|x' - x\|_\infty \leq \epsilon\}$ defines an axis-aligned box centered at x with side length 2ϵ. Thus, verifying ℓ_∞ robustness can be recast as bounding the network's output behavior over such a box.

Interval abstraction naturally aligns with this geometric structure. In particular, interval domains have been widely adopted to overapproximate neural network outputs under ℓ_∞ perturbations [20]. Within the framework of abstract interpretation, one propagates abstract input sets—represented as intervals or zonotopes—layer by layer through the network. Each layer is equipped with an *abstract transformer* that conservatively overapproximates its effect on the input set.

In this chapter, we investigate the theoretical power and limits of interval-based abstract interpretation in this setting. Specifically, we introduce and analyze the *interval universal approximation theorem*, which formalizes the representational capabilities of interval abstractions in neural-network verification.

5.2 Abstract Interpretation and Interval Abstraction

Abstract interpretation is a general methodology for reasoning about programs (or, in our case, functions like neural networks) by computing over approximations of sets of values. Rather than evaluating a function on a point, one evaluates it on an abstract set—such as a box in $\mathbb{R}^n$—and overapproximates its image. We begin by formalizing this with the notions of set semantics and interval domains.

Set Semantics. Let $f : \mathbb{R}^m \to \mathbb{R}$ be a function and $S \subseteq \mathbb{R}^m$ a set. The *set semantics* (or *collecting semantics*) of f is:

$$f(S) = \{f(\mathbf{x}) \mid \mathbf{x} \in S\}.$$

Computing $f(S)$ exactly is often intractable. The goal of abstract interpretation is to compute an overapproximation $N^{\#}(S^{\#})$ that is guaranteed to contain $f(S)$ while being efficiently computable.

The Interval Abstract Domain. The interval domain approximates arbitrary sets in $\mathbb{R}^m$ by axis-aligned boxes. A box is defined as:

$$B = \langle [l_1, u_1], \ldots, [l_m, u_m] \rangle,$$

where each interval $[l_i, u_i]$ bounds the projection of S onto the ith coordinate:

$$\alpha(S) = \langle [\inf S_i, \sup S_i] \rangle_{i=1}^{m}, \quad S_i = \{x_i \mid \mathbf{x} \in S\}.$$

The concretization function γ interprets an abstract box back into a concrete set:

$$\gamma(B) = \{\mathbf{x} \in \mathbb{R}^m \mid l_i \leq x_i \leq u_i \text{ for all } i\}.$$

Abstract Transformers. Given a neural network $\mathcal{N}$ composed of affine and nonlinear layers, we define corresponding abstract transformers that operate on interval inputs. The goal is to propagate abstract boxes through each layer to obtain an overapproximation of the network's output.

Definition 5.2.1 (Arithmetic Abstract Transformers)**.** Let $[l_1, u_1], [l_2, u_2]$ be one-dimensional intervals and $c \in \mathbb{R}$ a constant. Define:

$$\begin{aligned} c^{\#} &= [c, c] \\ x_i^{\#} &= [l_i, u_i] \\ [l_1, u_1] +^{\#} [l_2, u_2] &= [l_1 + l_2, u_1 + u_2] \\ [c, c] *^{\#} [l, u] &= [\min(cl, cu), \max(cl, cu)]. \end{aligned}$$

■

Definition 5.2.2 (Abstract Transformer for Activation [20])**.** Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a *monotonic* activation function and $[l, u]$ a one-dimensional interval. Then:

$$\sigma^{\#}([l, u]) = [\sigma(l), \sigma(u)].$$

■

These definitions can be lifted to vector-valued functions by applying operations elementwise.

Theorem 5.2.3 (Soundness of Interval Abstraction)**.** *Let B be a box in $\mathbb{R}^m$ such that $\gamma(B) \subseteq dom(\mathcal{N})$. Then:*

$$\mathcal{N}(\gamma(B)) \subseteq \gamma(N^{\#}(B)).$$

This theorem guarantees that abstract transformers yield safe overapproximations of the true network behavior. While this safety is essential for verification, it also raises questions about the expressiveness and precision of the interval domain. In the next sections, we formalize these concerns through the lens of universal approximation.

5.3 Squashable and Step Functions

The constructions in this section form the backbone of our main results: the Interval Universal Approximation (IUA) theorem and the corresponding hardness results for range approximation. These results rely on the expressiveness of neural networks with *squashable activation functions*, which can approximate *step functions*—a key primitive for building indicator functions, Boolean formulas, and encoding discrete logic within a continuous framework.

5.3.1 Neural Networks and Grammar

We define neural networks as compositions of primitive arithmetic operations and a fixed unary activation function $\sigma : \mathbb{R} \to \mathbb{R}$. Formally:

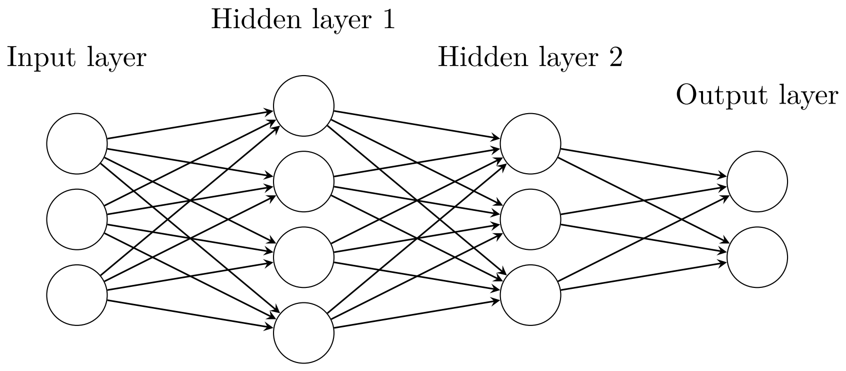


Figure 5.1: Layerwise representation of a typical feed-forward neural network consisting of several hidden layers. Adjacent layers are connected through affine transformations (matrix multiplications and bias additions), while activation layers apply non-linear functions, enabling the network to model complex relationships.

Definition 5.3.1 (Neural Network Grammar)**.** Let $\mathbf{x} = (x_1, \dots, x_m)$ be the input vector. A neural network $\mathcal{N}$ is defined inductively by:

$$\begin{array}{rcl} N & := & c \\ & | & x_i \\ & | & \mathcal{N}_1 + \mathcal{N}_2 \\ & | & c * \mathcal{N}_1 \\ & | & \sigma(\mathcal{N}_1) \end{array}$$

where $c \in \mathbb{R}$ is a constant, x_i is the i-th input, and σ is a fixed activation function. ■

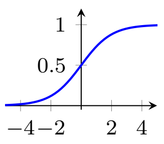
This grammar captures standard feedforward networks and is expressive enough for encoding logical circuits when paired with appropriate activations. The network can also be visualized in its layerwise form, where each layer applies a nonlinear activation σ, and adjacent layers are connected by affine transformations. An example is illustrated in Figure 5.1. Common activation functions include $\mathrm{sigmoid}, \mathrm{ReLU}, \mathrm{softplus}, \mathrm{smoothReLU}_a, \mathrm{ELU}$, whose definitions and shapes are shown in Figure 5.2.

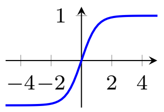
5.3.2 Squashable Activation Functions

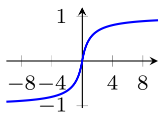
We now define a broad class of activation functions that generalize classical squashing functions like sigmoid and tanh.

Definition 5.3.2 (Squashable Activation Functions)**.** A function $\sigma : \mathbb{R} \to \mathbb{R}$ is *squashable* if:

Activation functions that satisfy Equation (5.1)

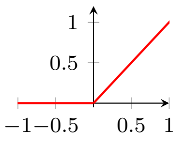


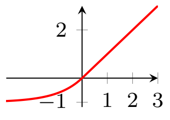




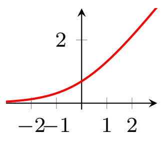
$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}} \qquad \tanh(x) = \frac{2}{1+e^{-2x}} - 1 \qquad \text{Softsign}(x) = \frac{x}{1+|x|}$$

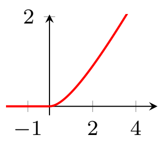
Activation functions that do not directly satisfy Equation (5.1)





$$\text{ReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} \qquad \text{ELU}(x) = \begin{cases} x, & x \geq 0 \\ e^x - 1, & x < 0 \end{cases}$$





$$\text{softplus}(x) = \log(1+e^x) \qquad \text{smoothReLU}_a(x) = \begin{cases} x - \frac{1}{a}\log(ax+1), & x \geq 0, a > 0 \\ 0, & x < 0 \end{cases}$$

Figure 5.2: Example activation functions. Smooth ReLU (smoothReLU_a) is parameterized by $a > 0$ ($a = 1$ is plotted).

1. there is $a < b \in \mathbb{R}$ such that

$$\lim_{x\to-\infty} \sigma(x) = a, \qquad \lim_{x\to\infty} \sigma(x) = b, \quad \text{and} \quad \forall x < y.\, \sigma(x) \leq \sigma(y) \tag{5.1}$$

2. *or* a function $\sigma' : \mathbb{R} \to \mathbb{R}$ that satisfies Equation (5.1) and can be expressed using the grammar in Definition 5.3.1 with copies of σ. For example, $\sigma'(x) = \sigma(2 * \sigma(x) - \sigma(x + 10))$.

■

In other words, squashable functions are those that can construct, via a finite network, a monotonic function with bounded limits in both directions. This allows them to approximate step functions when composed and scaled.

Proposition 5.3.3. *Let* $\sigma \in \{ReLU, \text{softplus}, \text{smoothReLU}_a, ELU\}$. *Then:*

$$\sigma'(x) = \sigma(1 - \sigma(-x))$$

is monotonic and bounded, hence satisfies the squashable condition.

Proof. Sketch: All listed functions are monotonic with finite left limits and diverge to ∞ as $x \to \infty$. The composition $\sigma(1 - \sigma(-x))$ is then increasing and bounded between $\sigma(1 - \sigma(\infty)) = \sigma(1 - \infty) = \sigma(-\infty)$ and $\sigma(1 - \sigma(-\infty)) = \sigma(1 - \text{const}) < \infty$. □

This means ReLU, ELU, Softplus, and Smooth ReLU all fall under our framework, even though they are not squashing functions themselves.

5.3.3 Step and Indicator Functions via Squashable Approximations

The step function:

$$\text{step}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

is a building block for many logical and indicator constructions. For example:

$$\text{step}(x) - \text{step}(x - 1)$$

represents the indicator function of the interval $(0, 1]$.

Squashable activations can approximate step functions via dilation:

$$\lim_{\lambda \to \infty} \sigma(\lambda x) = \text{step}(x),$$

in the pointwise sense, when σ is properly normalized. This idea is central to our constructions in both the approximation and hardness proofs.

5.3.4 Boolean Logic Encoded by Networks

Neural networks with squashable activations can encode Boolean formulas over binary variables $x_i \in \{0, 1\}$:

- Negation: $\neg x_i \rightsquigarrow 1 - x_i$.
- Conjunction: $x_i \wedge x_j \rightsquigarrow \text{step}(x_i + x_j - 1.5)$.

- Disjunction: $x_i \vee x_j \rightsquigarrow \text{step}(x_i + x_j - 0.5)$.

These encodings can be composed recursively to represent full CNF or DNF Boolean formulas. In later sections, we leverage this representation to show reductions from satisfiability problems to network verification.

5.4 Interval Universal Approximation: Definition

The classical universal approximation theorem states that feedforward neural networks with non-polynomial activation functions are dense in the space of continuous functions on compact domains [24, 36]. That is, for any continuous function $f : C \to \mathbb{R}$, where $C \subseteq \mathbb{R}^m$ is compact, and any $\varepsilon > 0$, there exists a neural network $\mathcal{N}$ such that:

$$\sup_{x \in C} |f(x) - \mathcal{N}(x)| < \varepsilon.$$

This classical result concerns *pointwise* approximation. In contrast, the goal of this paper is to study the approximation of the *range* of a function over sets—specifically, axis-aligned boxes—motivated by interval analysis and abstract interpretation.

Interval-Based Approximation. Given a function f and an input set $B \subseteq \mathbb{R}^m$, we are often interested in bounding the image set $f(B) := \{f(x) \mid x \in B\}$. This is especially relevant in robustness analysis, where B may represent all perturbations around an input point.

In the context of neural network verification, we consider an abstract transformer $N^{\#}(B)$, which soundly overapproximates the output of a neural network $\mathcal{N}$ on the input box B. Our goal is to find a neural network $\mathcal{N}$ such that $N^{\#}(B)$ closely matches the range $f(B)$ of the true target function f.

Definition 5.4.1. [δ-Interval Approximation] Let $f : C \to \mathbb{R}$ be a continuous function on a compact domain $C \subseteq \mathbb{R}^m$. A neural network $\mathcal{N}$ is said to *δ-interval approximate f* if, for every axis-aligned box $B \subseteq C$, the following holds:

$$[\min f(B) + \delta, \max f(B) - \delta] \subseteq N^{\#}(B) \subseteq [\min f(B) - \delta, \max f(B) + \delta],$$

where $N^{\#}(B)$ is the abstract interpretation (interval output) of $\mathcal{N}$ on B. ■

In words, $N^{\#}(B)$ approximates the true range $f(B)$ with additive slack δ on both lower and upper bounds (See Figure 5.3 for an illustration of δ-interval approximation).

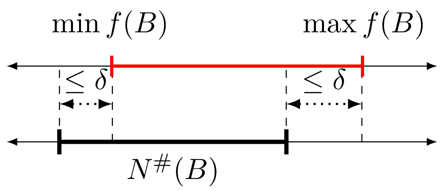


Figure 5.3: Illustration of δ-Interval Approximation.

5.5 Positive Result: Existence of Interval Approximators

We now show that interval approximation is always possible, provided the activation functions are squashable.

Theorem 5.5.1 (Interval Universal Approximation)**.** *Let $f : C \to \mathbb{R}$ be continuous over compact domain $C \subseteq \mathbb{R}^m$, and let σ be a squashable activation function. Then, for every $\delta > 0$, there exists a neural network $\mathcal{N}$ with activation σ such that $\mathcal{N}$ δ-interval approximates f.*

The proof is constructive (detailed in the next chapter): we use fine box partitions of C, and approximate indicator functions for each box using squashable activations.

Informally, the theorem says that we can always find a neural network whose abstract interpretation is arbitrarily close to the collecting semantics of the approximated function. Note also that there exists such a neural network for any fixed squashable activation function σ.

The IUA theorem has very exciting implications: We can show that one can always construct provably *robust* neural networks using any squashable activation function. The robustness property, which states that small perturbations in the input result in the same classification by a neural network, has been heavily studied recently, and the interval domain has been used to prove robustness in a range of domains [20, 50, 2]. Our result hints at a very close theoretical connection between robust neural networks and proofs using interval-based abstract interpretation.

5.6 Negative Result: Hardness of Interval Approximation

Even though interval approximators always exist, they may not be efficiently constructible. The hardness stems from the fact that approximating the range of a neural network (even over simple boxes) is computationally difficult.

Definition 5.6.1 (Range Approximation Problem)**.** Given a neural network $\mathcal{N} : [0,1]^m \to [0,1]$ and $\delta > 0$, compute an interval $[l, u]$ such that:

$$[\min \mathcal{N}(\mathbf{x}) + \delta, \max \mathcal{N}(\mathbf{x}) - \delta] \subseteq [l, u] \subseteq [\min \mathcal{N}(\mathbf{x}) - \delta, \max \mathcal{N}(\mathbf{x}) + \delta].$$

■

Theorem 5.6.2. *[Hardness of Range Approximation] Let $\delta < \frac{1}{2}$. The range approximation problem is Δ_2^P-intermediate (recall that a Δ_2^P-intermediate problem is harder than all problems in* NP *and* coNP*, see Definition 7.0.4). In particular, unless* $\mathsf{P} = \mathsf{NP}$*, the problem cannot be solved in polynomial time in the worst case.*

The result follows by encoding logical constraints (e.g., satisfiability of 3SAT) as neural networks and showing that even approximating the output range requires solving NP or coNP-hard problems.

Relationship Between Range and Interval Approximation

The Range Approximation (RA) problem is a natural subproblem of the Interval Universal Approximation (IUA) task. Whereas RA asks for approximate upper and lower bounds of a fixed network over a single input box (typically $[0,1]^m$), IUA seeks to construct a neural network $\mathcal{N}$ that, under interval abstract interpretation, approximates an arbitrary continuous function f over *all* axis-aligned boxes $B \subseteq [0,1]^m$.

This distinction implies that IUA is strictly harder than RA in multiple dimensions:

1. In RA, we only need f to be a neural network, while in IA, we aim to approximate any function;

2. In IA, we require the approximation holds for any B in the domain, while in RA, we only need it holds for the domain $[0,1]^m$;

3. In IA, we need to find a neural network that approximates f, but in RA, we do not require any specific ways to find a, b. If one can find the δ-interval approximation neural network, abstractly executing the neural network will return a, b automatically.

Since RA is a special case of IUA—namely, approximating a specific function $f = \mathcal{N}$ over a fixed box—it follows that any efficient algorithm for IUA must also solve RA. Therefore, our hardness result for RA implies a conditional hardness for IUA. In particular, if RA cannot be solved in polynomial time, then neither can IUA, unless one restricts the class of target functions or allows exponentially large networks.

Conclusion. This formal separation explains why interval-based reasoning about neural networks is difficult in practice. Even though pointwise universal approximation is classically easy (via standard neural construction theorems), achieving the same under interval semantics—i.e., certified robustness over entire input regions—faces a fundamental computational barrier.

Remark. The key idea is that even though real neural networks are continuous-valued, their expressive power mimics logic over $\{0, 1\}$, and approximating logical behavior (via step functions) is sufficient to show computational hardness.

Chapter 6

Interval Universal Approximation: Constructive Proof

6.1 Proof of the IUA Theorem

We prove the Interval Universal Approximation (IUA) theorem stated in Theorem 5.5.1. The central idea is to explicitly construct neural networks whose *abstract interpretation*—their behavior on input *intervals*—approximates the set semantics of a continuous target function with arbitrarily small error. Our construction generalizes and streamlines the ReLU-based argument of Baader et al. [5] by exploiting *squashable activation functions*, which enable smooth, tractable surrogates of indicator functions with provable bounds under interval semantics.

Why interval approximation is stricter than pointwise. Classical pointwise approximation often relies on indicator-like behavior over grid cells. Interval approximation is strictly more demanding: it requires the abstract semantics of the network to approximate the *range* of the function over any input box, i.e., the set of all outputs induced by all inputs in that box. Consequently, the network must behave correctly on regions, not merely on isolated points.

Two differences from the classical setting drive the technical work:

1. **Naive gridding fails for boxes.** For pointwise approximation, partitioning the domain and assigning cellwise values suffices. For interval semantics, a single input box may intersect multiple grid cells, and the output must conservatively overapproximate the union of function values across all intersected cells. Baader et al. [5] control this effect via an output-range *slicing* trick: they decompose the

target into bands of nearly constant height and approximate each band separately, thereby bounding the error uniformly across input boxes.

2. **Interval abstractions invalidate pointwise indicators.** A network that behaves like an indicator pointwise may exhibit large or unsound intervals under abstraction. Indeed, Theorem 5.6.2 shows that synthesizing interval-approximating networks can be computationally hard. ReLU indicators built in Baader et al. [5] (following He et al. [23]) are technically intricate and activation-specific. In contrast, our approach uses the *squashable* functions of Section 5.3 to form smoothed indicators that are easier to analyze and apply to a broader activation class, including ReLU.

Structure of the proof. The proof proceeds in three steps:

1. Using squashable activations, we construct smoothed "bump" functions that approximate axis-aligned box indicators and prove that their interval semantics are well behaved (Section 6.2).

2. We decompose the target f into narrow bands via a layered approximation scheme and tile the domain by small boxes.

3. We sum the bump functions within each slice to obtain a neural network whose abstract interpretation closely matches the function's output range.

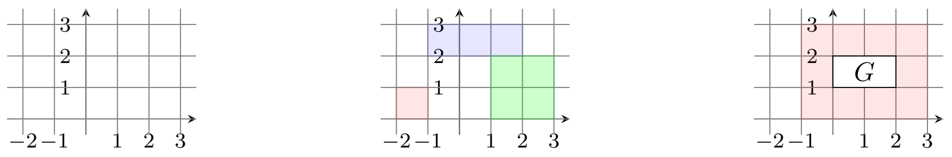
The resulting network $\mathcal{N}$ has abstract semantics $N^{\#}$ that approximate the collecting semantics of f within any prescribed tolerance δ, thereby establishing the constructive IUA theorem.

6.2 Approximating Indicator Functions

We formalize the construction of indicator surrogates using squashable activations. The goal is a network that is near 1 on a given axis-aligned box and near 0 outside a small neighborhood of that box—*even* when interpreted abstractly over input intervals.

Let C be compact and consider an ϵ-grid over C, i.e., a grid whose axis-aligned neighboring vertices are at l_∞-distance ϵ. Let $G = [a_1, b_1] \times \cdots \times [a_m, b_m]$ be a grid-aligned box (so each $b_i - a_i$ is a multiple of ϵ), and let $\mathcal{G}$ be the family of all such boxes. Define the *neighborhood*

$$\nu(G) \; = \; [a_1 - \epsilon, b_1 + \epsilon] \times \cdots \times [a_m - \epsilon, b_m + \epsilon].$$



(a) 1-grid over $\mathbb{R}^2$ (b) Three boxes in $\mathcal{G}$ (c) Box G & its neighborhood

Figure 6.1: A grid illustration

Our objective is an indicator-like function that is ≈ 1 on G and ≈ 0 outside $\nu(G)$. This grid-based idea parallels the nodal basis of He et al. [23]; see Figure 6.1 for an illustration.

We proceed in two stages:

1. Construct a one-dimensional approximate indicator $\hat{\sigma}$ via a squashable activation.
2. Combine the $\hat{\sigma}$ to obtain an m-dimensional approximate indicator $\mathcal{N}_G$ for any axis-aligned box $G \subseteq \mathbb{R}^m$.

We first build a 1D indicator for an interval using a squashable activation as in Section 5.3. The main design choice is a dilation factor that preserves precision under abstract interpretation.

By the IUA hypothesis, we are given a squashable activation σ. Without loss of generality we assume:

1. σ satisfies Equation (5.1) (Definition 5.3.2):

$$\lim_{x\to-\infty} \sigma(x) = a, \quad \lim_{x\to\infty} \sigma(x) = b, \quad \text{and} \quad \sigma(x) \in [a, b] \ \ \forall x \in \mathbb{R}.$$

   If necessary, Definition 5.3.2 allows replacing σ by an affine rescaling σ' satisfying Equation (5.1).

2. The left and right limits of σ are 0 and 1, respectively; if not, an affine transformation of the output enforces these limits.

Precision loss from asymptotic limits. Although σ has finite limits at both ends, it typically never attains them (e.g., the sigmoid's right limit is 1, but sigmoid$(x) \neq 1$ for all x). Approximating a step thus incurs a controllable precision loss, which we bound explicitly.

Dilation to realize a step surrogate. We obtain step-like behavior by dilation. By the definition of limits, we have:

Lemma 6.2.1. *For every $\theta > 0$ there exists $D > 0$ such that:*

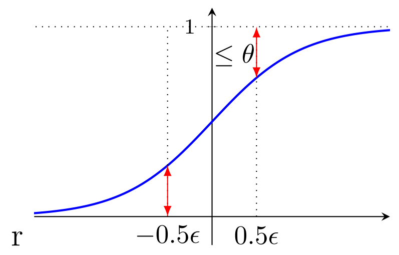


Figure 6.2: Precision loss θ when using a squashable activation to approximate a step.

1. *If $x \geq D$, then $\sigma(x) \in (1-\theta, 1]$.*
2. *If $x \leq -D$, then $\sigma(x) \in [0, \theta)$.*

Because the grid size is ϵ, we want the transition from ≈ 0 to ≈ 1 to occur within width ϵ. Let μ be the dilation factor. Using Lemma 6.2.1, it suffices to ensure:

1. if $x \geq \epsilon/2$, then $\sigma(\mu x) \in (1-\theta, 1]$;
2. if $x \leq -\epsilon/2$, then $\sigma(\mu x) \in [0, \theta)$.

Thus $\mu = 2D/\epsilon$ suffices.

Lemma 6.2.2. *For any $\theta > 0$, let $\mu = 2D/\epsilon$ where D is from Lemma 6.2.1. Then:*

1. *if $x \geq \epsilon/2$, $\sigma(\mu x) \in (1-\theta, 1]$;*
2. *if $x \leq -\epsilon/2$, $\sigma(\mu x) \in [0, \theta)$.*

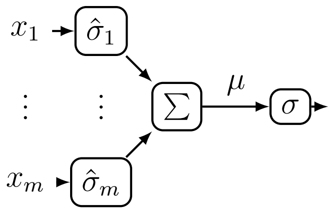
Example 6.2.1. *Figure 6.2 visualizes the loss θ incurred by the step surrogate.* ■

A 1D indicator on dimension i. We now construct an indicator-like function for the ith coordinate of a grid box G. Let the ith projection be $[a_i, b_i]$. Since G is grid-aligned, $b_i - a_i \geq \epsilon$, and the ith projection of $\nu(G)$ is $[a_i - \epsilon, b_i + \epsilon]$. We seek a function that is ≈ 1 on $[a_i, b_i]$ and ≈ 0 on $\mathbb{R} \setminus [a_i - \epsilon, b_i + \epsilon]$; loss within the neighborhood is unavoidable because the construction cannot perfectly distinguish G from its immediate vicinity.

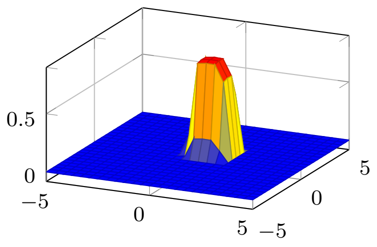
Taking a difference of two shifted step surrogates, define

$$\hat{\sigma}(x) \;=\; \sigma\Big(\mu\Big(x + \tfrac{\epsilon}{2} - a_i\Big)\Big) \;-\; \sigma\Big(\mu\Big(x - \tfrac{\epsilon}{2} - b_i\Big)\Big)\,. \tag{6.1}$$

Basic properties of $\hat{\sigma}$. The next lemmas show that $\hat{\sigma}$ behaves like an indicator: it is near 1 on $[a_i, b_i]$, near 0 outside $[a_i - \epsilon, b_i + \epsilon]$, and globally bounded by 1. We analyze the two terms separately.



(a) Architecture of $\mathcal{N}_G$ (constant shifts elided).



(b) $\mathcal{N}_G$ on $G = [0,1]^2$ with sigmoid, $\mu = 10$, $2\theta = 0.05$, $\epsilon = 1$.

Figure 6.3: **Step 2:** Neural construction of $\mathcal{N}_G$.

Lemma 6.2.3. *If $x \in [a_i, b_i]$, then:*

1. $\sigma(\mu(x + \epsilon/2 - a_i)) \in (1 - \theta, 1]$,

2. $\sigma(\mu(x - \epsilon/2 - b_i)) \in [0, \theta)$.

Lemma 6.2.4. *If $x \leq a_i - \epsilon$, then:*

1. $\sigma(\mu(x + \epsilon/2 - a_i)) \in [0, \theta)$,

2. $\sigma(\mu(x - \epsilon/2 - b_i)) \in [0, \theta)$.

Lemma 6.2.5. *If $x \geq b_i + \epsilon$, then:*

1. $\sigma(\mu(x + \epsilon/2 - a_i)) \in (1 - \theta, 1]$,

2. $\sigma(\mu(x - \epsilon/2 - b_i)) \in (1 - \theta, 1]$.

Abstract precision of $\hat{\sigma}$. We now bound the abstract interpretation of $\hat{\sigma}$. Let $\hat{\sigma}^{\#}(B)$ denote the abstract output interval for a 1D input box B.

Lemma 6.2.6. *For any 1D box B:*

1. $\hat{\sigma}^{\#}(B) \subset (-\infty, 1]$.

2. *If* $B \subset (-\infty, a_i - \epsilon]$ *or* $B \subset [b_i + \epsilon, \infty)$, *then* $\hat{\sigma}^{\#}(B) \subset (-\theta, \theta)$.

3. *If* $B \subset [a_i, b_i]$, *then* $\hat{\sigma}^{\#}(B) \subset (1 - 2\theta, 1]$.

Approximating an m-dimensional indicator

We lift the 1D construction to m dimensions. Let $G = [a_1, b_1] \times \cdots \times [a_m, b_m] \subseteq C$. If $\mathbf{x} \in G$, then $x_i \in [a_i, b_i]$ for all i. If $\mathbf{x} \notin \nu(G)$, then there exists i with $x_i \leq a_i - \epsilon$ or $x_i \geq b_i + \epsilon$.

Constructing $\mathcal{N}_G$. We seek a function with value ≈ 1 on G and ≈ 0 on $C \setminus \nu(G)$. In multiple dimensions we do not know *which* coordinate, if any, witnesses being outside $\nu(G)$. The 1D indicators $\hat{\sigma}$ provide per-coordinate evidence; combining them and then applying a step surrogate yields the desired multi-dimensional indicator.

Formally, define

$$\mathcal{N}_G(\mathbf{x}) \;=\; \sigma\left(\mu\left(\sum_{i=1}^{m} H_i(x_i) + \tfrac{\epsilon}{2}\right)\right), \tag{6.2}$$

where $H_i(x) \;=\; \hat{\sigma}_i(x) - (1 - 2\theta)$ and $\hat{\sigma}_i$ is the 1D construction for the interval $[a_i, b_i]$. The network in Figure 6.3a realizes $\mathcal{N}_G$.

By design, $\sum_{i=1}^{m} H_i(x_i)$ is positive on G and negative on $C \setminus \nu(G)$. The constant shift by $(1 - 2\theta)$ ensures negativity as soon as any coordinate is outside the neighborhood. Applying σ as a step surrogate then maps positive sums to ≈ 1 and negative sums to ≈ 0.

Example 6.2.2. *Figure 6.3b plots $\mathcal{N}_G$ for $\mathbf{x} \in \mathbb{R}^2$.* ■

Abstract precision of $\mathcal{N}_G$. We analyze the abstract precision via the auxiliary terms H_i. For any box $B \subseteq C$, let B_i be its ith projection.

Lemma 6.2.7 (Abstract interpretation of H_i)**.** *For any box $B \subseteq C$:*

1. *If $B \subseteq G$, then $\sum_{i=1}^{m} H_i^{\#}(B_i) \subseteq (0, \infty)$.*
2. *If $B \subseteq C \setminus \nu(G)$, then $\sum_{i=1}^{m} H_i^{\#}(B_i) \subseteq (-\infty, -\epsilon)$.*

The next theorem quantifies the abstract precision of $\mathcal{N}_G$.

Theorem 6.2.8 (Abstract interpretation of $\mathcal{N}_G$)**.** *For any box $B \subseteq C$:*

1. *$\mathcal{N}_G^{\#}(B) \subseteq [0, 1]$.*
2. *If $B \subseteq G$, then $\mathcal{N}_G^{\#}(B) \subseteq (1 - \theta, 1]$.*
3. *If $B \subseteq C \setminus \nu(G)$, then $\mathcal{N}_G^{\#}(B) \subseteq [0, \theta)$.*

Proof. (1) By Equation (6.2), the outer map is σ, whose range is $[0,1]$ by squashability and the assumed limits. Thus $\mathcal{N}_G^\#(B) \subseteq [0,1]$.

(2)If $B \subseteq G$, from Lemma 6.2.7, we know that $\sum_{i=1}^m H_i^\#(B_i) \subseteq (0,\infty)$. Then,

$$\sum_i^m H_i^\#(B_i) +^\# (0.5\epsilon)^\# \quad \subseteq \quad (0,\infty) +^\# (0.5\epsilon)^\# \quad \subseteq \quad (0.5\epsilon,\infty)$$

By Lemma 6.2.2, if $x \geq \epsilon/2$ then $1-\theta < \sigma(\mu x) \leq 1$, so

$$\mathcal{N}_G^\#(B) \;=\; \sigma^\#\Big(\mu^\# *^\# \left(\tfrac{\epsilon}{2},\infty\right)\Big) \;\subset\; (1-\theta,1].$$

(3) If $B \subseteq C \setminus \nu(G)$, from Lemma 6.2.7, we know that $\sum_{i=1}^m H_i^\#(B_i) \subseteq (-\infty,-\epsilon)$. Then,

$$\textstyle\sum_{i=1}^m H_i^\#(B_i) +^\# (0.5\epsilon)^\# \quad \subseteq \quad (-\infty,-\epsilon) +^\# (0.5\epsilon)^\# \quad \subseteq \quad (-\infty,-0.5\epsilon)$$

By Lemma 6.2.2, if $x \leq -\epsilon/2$ then $0 \leq \sigma(\mu x) < \theta$, hence

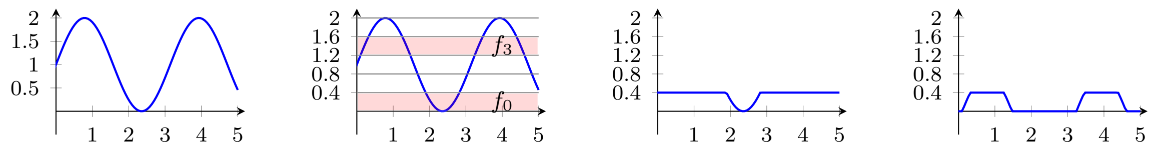
$$\mathcal{N}_G^\#(B) \;=\; \sigma^\#\Big(\mu^\# *^\# \left(-\infty,-\tfrac{\epsilon}{2}\right)\Big) \;\subset\; [0,\theta).$$

□

Complexity of the construction. A single m-dimensional indicator uses $2m+1$ activation evaluations, with depth 2 and width $2m$. Under ReLU, this corresponds to $4m+2$ neurons with depth 4 and width $2m$. For comparison, Baader et al. [5] require $10m-3$ ReLU units, depth $3+\log_2(m)$, and width $4m$.

6.3 Complete Proof of the IUA Theorem

We now complete the proof of the Interval Universal Approximation (IUA) theorem by assembling the full neural network $\mathcal{N}$. In the preceding sections, we constructed approximate indicator functions and established quantitative bounds on the precision of their abstract interpretation (Theorem 6.2.8). Building on these results, we follow the general framework of Baader et al. [5] for ReLU-based networks, extending it to arbitrary *squashable* activations. Because the latter approximate step functions only asymptotically, they introduce an additional source of imprecision compared to ReLUs. To accommodate this, we employ a finer partition of the target function's range, using slicing granularity $\delta/3$ instead of $\delta/2$ as in Baader et al. [5]. A detailed error analysis of this adjustment is deferred to the supplementary material. Below, we outline the complete constructive procedure for building $\mathcal{N}$ satisfying the IUA theorem.



(a) $f(x) = sin(2x) + 1$ (b) Sliced $f(x)$ (c) Example slice f_0 (d) Example slice f_3

Figure 6.4: Slicing $f(x) = sin(2x) + 1$ with approximation tolerance $\delta = 1.2$.

Slicing the target function. Let $f : C \to \mathbb{R}$ be the continuous target function, and let δ denote the desired approximation tolerance, as specified in the IUA theorem (Theorem 5.5.1). Without loss of generality, we assume $\min f(C) = 0$.[1] Let $u = \max f(C)$, so the range of f is $[0, u]$.

Define $\tau = \delta/3$, and decompose f into a sequence of *function slices* f_i, each taking values within $[0, \tau]$. Let $K = \lfloor u/\tau \rfloor$. The sum of these slices recovers f, i.e.,

$$f = \sum_{i=0}^{K} f_i.$$

Each slice $f_i : C \to [0, \tau]$ is defined as:

$$f_i(\mathbf{x}) = \begin{cases} f(\mathbf{x}) - i\tau, & \text{if } i\tau < f(\mathbf{x}) \leq (i+1)\tau, \\ 0, & \text{if } f(\mathbf{x}) \leq i\tau, \\ \tau, & \text{if } f(\mathbf{x}) > (i+1)\tau. \end{cases}$$

Example 6.3.1. *An illustration of the slicing procedure is provided in Figure 6.4, which depicts the decomposition of $f(x) = \sin(2x) + 1$ with $\delta = 1.2$.* ■

Approximating each slice. For each slice f_i, we construct a neural network $\mathcal{N}_i$ that approximates it using the indicator approximation $\mathcal{N}_G$ from Equation (6.2). Because C is compact, the number of grid boxes $|\mathcal{G}|$ is finite. The normalized function $\frac{1}{\tau} f_i(\mathbf{x})$ behaves similarly to the indicator of the set

$$S_i = \{\mathbf{x} \in C \mid f(\mathbf{x}) > (i+1)\tau\},$$

which identifies regions where $f(\mathbf{x})$ exceeds the upper boundary of the ith slice. To approximate $\frac{1}{\tau} f_i(\mathbf{x})$, we consider all boxes $G \in \mathcal{G}$ entirely contained in S_i and construct indicator networks to identify such boxes. Formally, define

$$\mathcal{G}_i = \{G \in \mathcal{G} \mid f(G) > (i+1)\tau\}.$$

[1]If not, we can apply an affine shift to ensure $\min f(C) = 0$.

Then the neural network $\mathcal{N}_i$ is given by

$$\mathcal{N}_i(\mathbf{x}) = \sigma\left(\mu\left(\sum_{G \in \mathcal{G}_i} \mathcal{N}_G(\mathbf{x}) - 0.5\right)\right),$$

which outputs values close to 1 for $\mathbf{x} \in S_i$ and close to 0 otherwise.

Summing the approximations. Since $f = \sum_{i=0}^{K} f_i$ and each $\mathcal{N}_i$ approximates $\frac{1}{\tau} f_i$, we define the final network as

$$\mathcal{N}(\mathbf{x}) = \tau \sum_{i=0}^{K} \mathcal{N}_i(\mathbf{x}).$$

By construction, $\mathcal{N}$ approximates f within δ under interval semantics. Therefore, $\mathcal{N}$ satisfies the requirements of the Interval Universal Approximation theorem, completing the constructive proof.

Chapter 7

Hardness of Interval Approximation

This chapter investigates the computational complexity of constructing interval approximators. The previous chapter established a constructive Interval Universal Approximation (IUA) result: for any continuous target in the universal approximation class, there exists a neural network whose *abstract interpretation* δ-approximates the function's collecting semantics. Here we show a complementary phenomenon: even when restricting to a *single* network over the domain $[0,1]^m$, obtaining a guaranteed-quality interval (or even coarse range) approximation is intractable. In particular, we prove a dichotomy: for tolerance $\delta \geq 1/2$ the problem is trivial, whereas for $\delta < 1/2$ it becomes simultaneously NP-hard and coNP-hard; under mild, standard assumptions (polynomial-time evaluability and finite-precision inputs), the problem is Δ_2-intermediate.

7.0.1 The Polynomial Hierarchy

We briefly recall the fragments of the polynomial hierarchy needed in this chapter, which was also defined in Section 4.1.1.

Definition 7.0.1 (NP and coNP)**.** A language L is in NP if there exist a polynomial-time Turing machine M and a polynomial q such that

$$x \in L \iff \exists\, u \in \{0,1\}^{q(|x|)}.\, M(x,u) = 1.$$

A language is in coNP if its complement is in NP, equivalently if there exist M, q such that

$$x \in L \iff \forall\, u \in \{0,1\}^{q(|x|)}.\, M(x,u) = 1.$$

■

Example 7.0.1. *SAT (satisfiability of a Boolean formula) is* NP*-complete; TAUT (tautology) is* coNP*-complete.* ■

We write $\Sigma_1 = \mathsf{NP}$ and $\Pi_1 = \mathsf{coNP}$. Higher levels alternate quantifiers:

Definition 7.0.2 (Σ_2 and Π_2)**.** A language L is in Σ_2 if there exist a polynomial-time M and polynomial q such that

$$x \in L \iff \exists\, u_1 \in \{0,1\}^{q(|x|)}\, \forall\, u_2 \in \{0,1\}^{q(|x|)}.\, M(x, u_1, u_2) = 1.$$

Similarly, $L \in \Pi_2$ if

$$x \in L \iff \forall\, u_1 \in \{0,1\}^{q(|x|)}\, \exists\, u_2 \in \{0,1\}^{q(|x|)}.\, M(x, u_1, u_2) = 1.$$

■

Definition 7.0.3 (Δ_2)**.** $\Delta_2 \ = \ \Sigma_2 \cap \Pi_2$. ■

Clearly $\mathsf{NP}, \mathsf{coNP} \subseteq \Delta_2$ (by fixing an empty witness/adversary string). We recall that it is open whether $\Delta_2 = \mathsf{NP} \cup \mathsf{coNP}$; under the standard belief $\mathsf{coNP} \not\subseteq \mathsf{NP}$, any Δ_2-intermediate class strictly contains NP but is strictly contained in Δ_2.

Definition 7.0.4 (Δ_2-intermediate)**.** A collection of languages $\mathbb{L}$ is Δ_2*-intermediate* if $\mathsf{NP} \cup \mathsf{coNP} \subseteq \mathbb{L} \subseteq \Delta_2$. ■

7.0.2 The Range Approximation Problem

We next formalize a basic decision/approximation task that captures the essence of interval approximation at the domain scale. Throughout, we consider networks $f : [0,1]^m \to [0,1]$, and by "polynomial time" we mean polynomial in m.

Definition 7.0.5 (δ-range approximation)**.** Let $\delta > 0$ and let $f : [0,1]^m \to [0,1]$ be a neural network. Write

$$\ell \ = \min_{\mathbf{x} \in [0,1]^m} f(\mathbf{x}), \qquad u \ = \max_{\mathbf{x} \in [0,1]^m} f(\mathbf{x}).$$

We say that we *δ-range approximate* f if we produce $a \leq b$ in $[0,1]$ such that

$$[\ell + \delta,\, u - \delta] \ \subseteq \ [a, b] \ \subseteq \ [\ell - \delta,\, u + \delta].$$

■

This notion is strictly weaker than the δ-*interval* approximation of Definition 5.4.1 in three respects: (i) RA speaks about an *existing* network f (IA approximates an arbitrary target function); (ii) RA concerns the whole domain $[0,1]^m$ (IA demands precision for *every* input box); and (iii) RA does not require constructing another network—any valid $[a,b]$ suffices. Consequently, *if* RA is hard, then building a polynomial-time *constructive* IA network is at least as hard.

We now state a precise dichotomy.

Theorem 7.0.6 (Dichotomy for δ-range approximation)**.** *Let $f : [0,1]^m \to [0,1]$ be a neural network with any squashable activation.*

1. *If $\delta \geq \frac{1}{2}$, then δ-range approximation is trivial.*
2. *If $\delta < \frac{1}{2}$, then δ-range approximation is both NP-hard and coNP-hard. Moreover, if f is polynomial-time evaluable and inputs have finite precision, then δ-range approximation is Δ_2-intermediate.*

Proof sketch of (1). Taking $a = b = 1/2$ satisfies the inclusions since $0 \leq \ell \leq u \leq 1$ implies $u - \delta \leq 1/2 \leq u + \delta$ and $\ell - \delta \leq 1/2 \leq \ell + \delta$ whenever $\delta \geq 1/2$.

We prove Item (2) in Section 7.1. The reduction encodes Boolean formulas by networks with step-like activations; since squashable units approximate steps arbitrarily well, the hardness transfers.

Implications. One might hope that *approximate* range computation is easier than exact range computation (as happens for many NP-hard optimization problems [49]). Theorem 7.0.6 shows a sharp threshold: either trivial or intractable, depending on δ. In particular, even a pointwise-accurate approximator $\mathcal{N}_0$ offers no easy path to an interval approximator whose abstract semantics δ-approximate the set semantics of $\mathcal{N}_0$. This underscores the nontriviality of IUA despite the classical nature of UA.

7.1 Hardness of Range Approximation

We establish Item (2) of Theorem 7.0.6: for $\delta < 1/2$ and squashable activations, δ-range approximation is NP-hard and coNP-hard; with polynomial-time evaluability and finite input precision, it is Δ_2-intermediate. We proceed in two stages. First, we analyze the Boolean analogue (deciding the range of a Boolean formula). Second, we lift the result to neural networks via reductions.

7.1.1 Deciding the Range of a Boolean Formula

Let ϕ be a Boolean formula, and let $R_\phi \subseteq \{0,1\}$ denote its range. Determining whether $R_\phi = \{0\}$, $R_\phi = \{1\}$, or $R_\phi = \{0,1\}$ lies in Δ_2 and is both NP-hard and coNP-hard.

Membership in Δ_2. We can express:

$$\begin{aligned} R_\phi = \{0\} &\iff \forall x.\, \phi(x) = 0, \\ R_\phi = \{1\} &\iff \forall x.\, \phi(x) = 1, \\ R_\phi = \{0,1\} &\iff \exists x, y.\, \phi(x) = 1 \wedge \phi(y) = 0. \end{aligned}$$

placing the problem in both Σ_2 and Π_2.

Hardness. If we could decide R_ϕ, we could decide SAT (whether $R_\phi \neq \{0\}$) and TAUT (whether $R_\phi \neq \{0,1\}$). Hence the problem is Δ_2-intermediate.

7.1.2 Range Approximation for Neural Networks

We now show that the δ-range approximation (RA) problem for neural networks inherits this intermediate hardness.

RA is in Δ_2 (under mild assumptions). We require that (i) f is polynomial-time evaluable in m, and (ii) inputs have finite precision (hence exponentially many candidates). Exact range equality

$$\exists x, y\, \forall z. \Big(f(x) = a \wedge f(y) = b \wedge f(z) \leq b \wedge f(z) \geq a \Big) \tag{7.1}$$

is expressible in both Σ_2 and Π_2 (the universal z does not depend on x, y), so exact range is in Δ_2; RA is no harder.

Lemma 7.1.1. *Under polynomial-time evaluability and finite input precision, the δ-range approximation problem of Theorem 7.0.6 lies in Δ_2 for $\delta < 1/2$.*

NP- and coNP-hardness. We sharpen the result by reductions from 3SAT and 3DNF-TAUT. The key is to encode Boolean structure by networks with squashable activations that implement step surrogates (as in Section 5.3). We first isolate an NP-hard *gap* decision about the network maximum.

Let F_m be the set of networks $f : [0,1]^m \to [0,1]$, and $F = \bigcup_{m \geq 1} F_m$. Fix $\delta < 1/2$ and define

$$F_\delta^+ = \bigcup_{m \geq 1} \Big\{ f \in F_m \;\Big|\; \max_{\mathbf{x} \in [0,1]^m} f(\mathbf{x}) > \tfrac{1}{2} + \delta \Big\}, \tag{7.2}$$

$$F_\delta^- = \bigcup_{m \geq 1} \Big\{ f \in F_m \;\Big|\; \max_{\mathbf{x} \in [0,1]^m} f(\mathbf{x}) \leq \tfrac{1}{2} - \delta \Big\}. \tag{7.3}$$

Lemma 7.1.2. *Given $f \in F_\delta^+ \cup F_\delta^-$, determining whether $f \in F_\delta^+$ or $f \in F_\delta^-$ is* NP*-hard.*

Since a δ-range approximation yields a δ-approximation to $\max f$ (hence separates F_δ^+ from F_δ^-), Lemma 7.1.2 implies:

Lemma 7.1.3. *For $\delta < 1/2$, the δ-range approximation problem is* NP*-hard.*

Reduction from 3SAT. Given a 3CNF instance $\phi = \bigwedge_{j=1}^{k} C_j$ over variables $X_1, \dots, X_m$, we construct a network N_ϕ with squashable activations that realizes a *gap*:

$$\phi \text{ satisfiable } \Rightarrow \max N_\phi > \tfrac{1}{2} + \delta, \qquad \phi \text{ unsatisfiable } \Rightarrow \max N_\phi \le \tfrac{1}{2} - \delta.$$

We implement literals by inputs $x_i \in [0,1]$ with negation $1 - x_i$, pass them through a calibrated squashable surrogate σ_1 that maps "false" to a small negative value and "true" to a positive value, aggregate disjunctions via σ_2 (ensuring a small positive contribution if any literal is true, else a negative penalty), and finally aggregate clauses by a sum passed through σ_3 that thresholds around $1/2$ with margin δ. Concretely, choose three squashable functions $\sigma_1, \sigma_2, \sigma_3$ satisfying:

$$\sigma_1(z) \in [-0.2, -0.1] \text{ for } z \le 0.6, \qquad \sigma_1(z) \in [0.5, 0.6] \text{ for } z \ge 0.7, \tag{7.4}$$

$$\sigma_2(z) \le -1 \text{ for } z \le 0, \qquad \sigma_2(z) \in \left[\tfrac{1}{2k}, \tfrac{1}{k}\right] \text{ for } z \ge 0.1, \tag{7.5}$$

$$\sigma_3(z) \in [0,\, \tfrac{1}{2} - \delta) \text{ for } z \le 0, \qquad \sigma_3(z) \in (\tfrac{1}{2} + \delta,\, 1] \text{ for } z \ge \tfrac{1}{2}. \tag{7.6}$$

(Any squashable activation can be shifted/dilated to meet these ranges.) For each clause $C_j = L_{j1} \vee L_{j2} \vee L_{j3}$ define

$$c_j \;=\; \sigma_2\Big(\sigma_1(l_{j1}) + \sigma_1(l_{j2}) + \sigma_1(l_{j3})\Big), \qquad y \;=\; \sigma_3\Big(\sum_{j=1}^{k} c_j\Big).$$

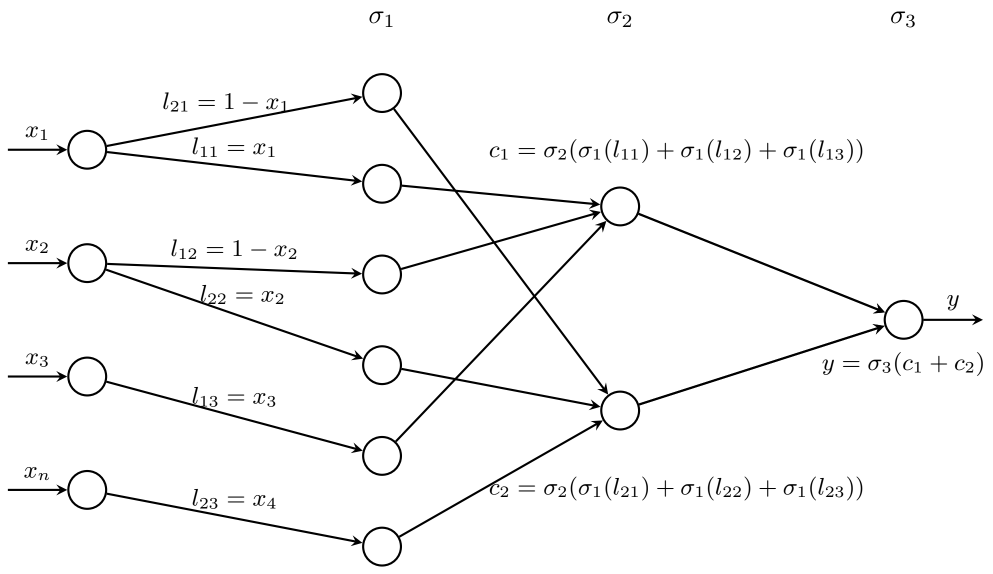
Then N_ϕ outputs y.

Proposition 7.1.4. *Let $y_u = \max_{\mathbf{x} \in [0,1]^m} N_\phi(\mathbf{x})$. Then:*

1. *If ϕ is satisfiable, $y_u > \frac{1}{2} + \delta$.*

2. *If ϕ is unsatisfiable, $y_u \le \frac{1}{2} - \delta$.*

Proof. (1) For a satisfying assignment $\mathbf{x} \in \{0,1\}^m$, each clause C_j has a true literal, hence by (7.4) its σ_1-image is ≥ 0.5 while the other two are ≥ -0.2, so the sum ≥ 0.1, giving $c_j \ge 1/(2k)$ by (7.5). Summing, $\sum_j c_j \ge 1/2$, so $y > \frac{1}{2} + \delta$ by (7.6).



σ_1
σ_2
σ_3
x_1
x_2
x_3
x_n
$l_{21} = 1 - x_1$
$l_{11} = x_1$
$l_{12} = 1 - x_2$
$l_{22} = x_2$
$l_{13} = x_3$
$l_{23} = x_4$
$c_1 = \sigma_2(\sigma_1(l_{11}) + \sigma_1(l_{12}) + \sigma_1(l_{13}))$
$c_2 = \sigma_2(\sigma_1(l_{21}) + \sigma_1(l_{22}) + \sigma_1(l_{23}))$
y
$y = \sigma_3(c_1 + c_2)$

Figure 7.1: The neural network encoding for $(X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee X_2 \vee X_4)$

(2) Conversely, if $y_u > \frac{1}{2} - \delta$, then for some $\mathbf{z}$ we have $y(\mathbf{z}) > \frac{1}{2} - \delta$, forcing $\sum_j c_j(\mathbf{z}) > 0$ by (7.6). Thus each $c_j(\mathbf{z}) > -1$; otherwise $\sum_j c_j \leq -1 + (k-1) \cdot (1/k) < 0$ using (7.5). Hence for each clause j, $\sigma_1(l_{j1}) + \sigma_1(l_{j2}) + \sigma_1(l_{j3}) > 0$, implying at least one literal input exceeds 0.6 (again by (7.4)). Rounding $\mathbf{z}$ by $x_i \leftarrow \mathbf{1}[z_i \geq 0.6]$ yields a satisfying assignment, contradicting unsatisfiability. □

Proposition 7.1.4 implies Lemma 7.1.2, hence Lemma 7.1.3. A symmetric construction with 3DNF encodes TAUT, yielding:

Lemma 7.1.5. *For $\delta < 1/2$, the δ-range approximation problem is* coNP*-hard.*

Decision version and corollary. Combining Δ_2-membership of exact range with the above hardness yields:

Corollary 7.1.6. *Let $f : [0,1]^m \to [0,1]$ be polynomial-time evaluable with finite-precision inputs. Then deciding the exact range of f is Δ_2-intermediate.*

Consequence for verification. As an immediate application, consider falsification (safety violation) queries that ask whether linear input constraints together with an output predicate are satisfiable [29].

Corollary 7.1.7. *It is* NP*-hard to falsify correctness of neural networks with any squashable activation (i.e., to decide satisfiability of a conjunction of linear input constraints and a linear output threshold).*

Proof. Given ϕ, form N_ϕ as above and ask whether there exists $\mathbf{x} \in [0,1]^m$ with $y > 0.5$ subject to $0 \leq x_i \leq 1$. This is satisfiable iff $N_\phi \in F_\delta^+$, which is NP-hard to decide by Lemma 7.1.2. $\square$

Discussion. Because ReLU activations are squashable, Corollary 7.1.7 subsumes the NP-hardness of falsification for ReLU networks [29]. More broadly, Theorem 7.0.6 shows that interval-level guarantees for even a *fixed* network present inherent computational hurdles that are absent in pointwise universal approximation. This gap clarifies why the constructive IUA proof in the previous chapter must delicately balance approximation and abstraction error: while the existence of such approximators is assured, their efficient synthesis or certification is provably difficult once $\delta < 1/2$.

This chapter established that precise interval range approximation is Δ_2–intermediate, revealing a fundamental complexity barrier for interval-based reasoning even when the underlying network is simple. These results explain why interval semantics, while expressive, become computationally fragile at high precision. The hardness findings also mark the limit of what interpretation alone can provide: coarse abstractions are universal but not always tractable. The next part refines the semantic lens by moving from interval abstraction to symbolic and algebraic interpretations that preserve more structure and expose new dualities.

Part III

Algebraic Interpretation of Neural Networks

This part advances the dissertation's second principle: *duality*, viewed through a symbolic and algebraic interpretation of neural networks. While Part II employed interval semantics to reason about sets of inputs and their abstract transformers, interval-based methods inevitably lose precision, reflecting the tradeoffs inherent in coarse-grained abstractions. To overcome these limitations, Part III refines the semantic lens: instead of propagating intervals, we symbolically *unfold* a network into a system of polynomial constraints that captures its computation exactly.

Working in this algebraic regime transforms neural network semantics into the language of real algebraic geometry. Each layer—affine transformation, activation, pooling, or group operation—is encoded as a conjunction of polynomial equalities and inequalities. The resulting *symbolic network* is a single semialgebraic object whose variables represent the intermediate activations y_i, x_i and whose feasible region captures all valid input–output behaviors. This viewpoint yields several benefits: (i) it enables finer-grained reasoning than interval semantics; (ii) it unifies many verification questions as first-order statements over the reals; and (iii) it reveals a clear boundary between operations that admit quadratic encodings (degree ≤ 2) and those that require higher-degree or transcendental structure. These distinctions become central to the complexity-theoretic and optimization-based developments that follow.

A key theme in this part is *quadratic expressiveness*. Many core neural network components—ReLU, leaky-ReLU, maxout, max-pooling, GroupSort, and a range of input constraint sets such as ℓ_p balls—can be represented or approximated within a quadratic constraint system. Even transcendental activations like sigmoid and tanh, which are not semialgebraic, admit arbitrarily precise semialgebraic envelopes using ReLU-based constructions. This quadratic boundary marks the point where we can obtain both exact logical characterizations and principled approximation schemes.

Beyond expressiveness, the algebraic interpretation brings powerful logical tools. Because symbolic networks live in the first-order theory of real closed fields, classical results such as Tarski–Seidenberg quantifier elimination immediately yield decidability for a broad class of verification queries. Although these procedures are far too costly for practical use, their existence provides a theoretical baseline: the difficulty of computation stems not from undecidability but from quantitative and geometric complexity, which Part III explores through optimization relaxations.

Structure of Part III. Chapter 8 develops the symbolic and algebraic interpretation of neural networks. It constructs the polynomial constraint system that represents affine layers, activation graphs, pooling/group operations, and standard perturbation sets, emphasizing which components admit exact quadratic encodings. It also classifies verification properties—adversarial reachability, output dominance, layer invariants, and slope-restricted global sensitivity—that become first-order properties over these algebraic systems.

Chapter 9 builds on this foundation to study quantitative robustness via semidefinite relaxations. Using Shor's lifting and Lagrangian duality, it translates the quadratic encodings from Chapter 8 into semidefinite programs that upper bound global Lipschitz constants. These SDPs admit constant-factor approximation guarantees derived from Banach-space geometry, clarifying the fundamental limits of global robustness estimation.

Part III therefore completes the algebraic half of the dissertation's duality theme: symbolic logical reasoning provides generality and exactness, while convex optimization leverages the same algebraic structure to obtain tractable, quantitative guarantees. Together, these chapters bridge the gap between abstract semantics and computational analysis, preparing the ground for the incremental methods developed in Part IV.

Chapter 8

Symbolic and Algebraic Interpretation of Neural Networks

This chapter moves from interval-based abstraction to a precise *algebraic* account of neural-network semantics. Instead of over-approximating behavior by intervals, we *symbolically execute* a network into a system of polynomial equalities and inequalities over fresh variables representing intermediate layer values. Working in this symbolic/algebraic regime yields (i) finer-grained reasoning than interval semantics, (ii) a unifying view of many verification questions as first-order properties over the reals, and (iii) a clear boundary between what can and cannot be represented by *quadratic* relations (degree ≤ 2). Throughout, we emphasize problems and constructions that admit an interpretation entirely in terms of quadratic forms, postponing any discussion of optimization or relaxations to the next chapter.

8.1 Algebraic Unfolding of Feedforward Networks

Consider a feedforward network $f : \mathbb{R}^m \to \mathbb{R}^\ell$ with layers

$$x_0 = x \in \mathbb{R}^m, \qquad y_i = W_i x_{i-1} + b_i, \qquad x_i = \sigma_i(y_i) \;\; (i = 1, \dots, L), \qquad z = x_L = f(x).$$

We introduce a distinct symbol for each y_i and x_i and relate them by algebraic (polynomial) constraints:

- **Affine layers.** For each i, the relation $y_i = W_i x_{i-1} + b_i$ is linear and hence quadratic.

- **Activation layers.** For each i, we add constraints capturing $x_i = \sigma_i(y_i)$. Which constraints we use depends on σ_i; our goal is to express them as conjunctions of polynomial relations, ideally quadratic.

The conjunction over all layers yields a *symbolic network*:

$$\Phi_{\text{net}}(x, \{y_i\}, \{x_i\}, z) \;\equiv\; \bigwedge_{i=1}^{L} \Big(y_i = W_i x_{i-1} + b_i\Big) \;\wedge\; \bigwedge_{i=1}^{L} \Gamma_{\sigma_i}(y_i, x_i) \;\wedge\; (z = x_L),$$

where Γ_σ is a conjunction of polynomial constraints defining the activation graph.

Quadratic encodings for common elementwise activations. We now record quadratic encodings that will be used repeatedly.

ReLU (faithful local graph). For $x = \mathrm{ReLU}(y)$, the equivalence

$$x \geq 0, \quad x \geq y, \quad (x - y)\, x \leq 0$$

is sound and complete for the ReLU graph, coordinate-wise.

Slope-restricted view (global, pairwise). For any scalar activation with derivative in $[a, b]$ one has, for any two inputs $y, \tilde{y}$ and outputs $x = \sigma(y)$, $\tilde{x} = \sigma(\tilde{y})$,

$$\big(\, x - \tilde{x} - a(y - \tilde{y})\,\big) \cdot \big(\, x - \tilde{x} - b(y - \tilde{y})\,\big) \;\leq\; 0,$$

a single quadratic inequality encoding monotonicity and slope bounds. Taking $a = 0, b = 1$ recovers the ReLU case.

Absolute value, max, and group operations. These primitive semialgebraic operations admit quadratic encodings:

$$\begin{aligned} u = |v| \;&\Longleftrightarrow\; u^2 = v^2 \;\wedge\; u \geq 0, \\ w = \max(u, v) \;&\Longleftrightarrow\; 2w = (u + v) + |u - v|, \\ \text{GroupSort: } (w_1, w_2) &= (\max(u, v), \min(u, v)), \quad \min(u, v) = -\max(-u, -v). \end{aligned}$$

Consequently, max-pooling and maxout layers are expressible by iterating the binary max encoding.

Non-elementwise but semialgebraic activations. Any activation whose graph is a semialgebraic set (finite Boolean combinations of polynomial inequalities) admits an exact quadratic presentation by introducing auxiliary variables to reduce degree (e.g., encode t^3 via $t^2 = s$, st). Examples include ReLU, leaky-ReLU, $\mathrm{Softsign}(x) = x/(1 + |x|)$ via

$$x = y(1 + u), \quad u^2 = x^2, \quad u \geq 0 \quad \text{and} \quad y = \mathrm{Softsign}(x).$$

Transcendental activations are not semialgebraic. By contrast, $\mathrm{sigmoid}(x) = 1/(1+e^{-x})$ and tanh are not semialgebraic; see Section 8.5 for a proof outline. Thus they admit no *exact* finite conjunction of polynomial constraints. Nevertheless, they can be *arbitrarily well* sandwiched by semialgebraic (piecewise-linear) upper/lower graphs, hence approximated within any prescribed tolerance using ReLU gadgets (cf. Theorem 8.5.2).

8.2 Classification Semantics as Algebraic Relations

We focus on classification models $f : \mathbb{R}^m \to \mathbb{R}^\ell$. Let $f^{(j)}$ denote the j-th logit and $\mathcal{C}(f,x) = \arg\max_{j\in[\ell]} f^{(j)}(x)$.

Logit-margin predicates. Fix a base input a with predicted class j. For any $k \neq j$, define the *margin* predicate

$$\mathrm{Margin}_{j\to k}(x) \;\equiv\; f^{(k)}(x) - f^{(j)}(x) \;>\; 0.$$

When we unfold f symbolically, each $f^{(k)}$ is a linear functional of the last hidden representation x_{L-1}:

$$f^{(k)}(x) - f^{(j)}(x) \;=\; (w^{(k)} - w^{(j)})^\top x_{L-1} + (b^{(k)} - b^{(j)}),$$

which is linear, hence quadratic. Therefore, *every* misclassification event "x causes $j \to k$" is a quadratic inequality over the layer symbols.

Adversarial perturbations as quadratic constraints. Let $B_p(a,\varepsilon) = \{x : \|x-a\|_p \leq \varepsilon\}$. For $p \in \{2,\infty\}$,

$$\|x-a\|_2^2 \leq \varepsilon^2 \quad \text{or} \quad (x_i - a_i)^2 \leq \varepsilon^2 \;\; \forall i$$

are quadratic. For $p = 1$, the encoding

$$\|x-a\|_1 \leq \varepsilon \iff \exists u \geq 0 : \sum_i u_i \leq \varepsilon, \;\; (x_i - a_i)^2 \leq u_i^2$$

is quadratic after introducing u. More generally, rational $p \geq 1$ are handled via auxiliary variables that reduce exponents to degree 2.

Local robustness as a first-order sentence. For fixed (a,j) the *existence* of an adversarial example within $B_p(a,\varepsilon)$ is the first-order sentence over reals

$$\exists x, \{y_i\}, \{x_i\}, z : \; \Phi_{\text{net}}(x,\ldots,z) \;\wedge\; x \in B_p(a,\varepsilon) \;\wedge\; \mathrm{Margin}_{j\to k}(x),$$

a conjunction of quadratic constraints. Robustness at (a,j) is the negation of a finite disjunction over $k \neq j$ of these existential formulas, hence also first-order.

Global (Lipschitz-style) sensitivity in algebraic form. For activations with derivative bounded in $[a, b]$, pairwise slope constraints yield, for any two inputs $x, \tilde{x}$ and corresponding layer pairs (y_i, x_i), $(\tilde{y}_i, \tilde{x}_i)$, quadratic relations in the *differences* $\Delta y_i = y_i - \tilde{y}_i$, $\Delta x_i = x_i - \tilde{x}_i$:

$$\Delta y_i = W_i \Delta x_{i-1}, \qquad (\Delta x_i - a\Delta y_i)(\Delta x_i - b\Delta y_i) \leq 0.$$

Thus, properties expressible purely in terms of permissible layerwise slopes (e.g., norm bounds on Jacobians expressed via quadratic inequalities in paired trajectories) fall within the same symbolic framework, without committing to any particular optimization viewpoint.

8.3 What Fits into Quadratic Form: Problems and Constructions

The following families of questions admit a direct quadratic (degree ≤ 2) encoding via the symbolic network Φ_{net} and the norm/activation gadgets above.

1. **Adversarial reachability (existential).** "Is there x within $B_p(a, \varepsilon)$ that flips $j \to k$?" — existential quadratic constraints (§8.2).

2. **Output dominance over a set (universal).** "For all $x \in S$, $f^{(j)}(x) \geq f^{(k)}(x)$" with S given by conjunctions of quadratic constraints (boxes, balls, polytopes with quadratic faces).

3. **Layer-wise invariants.** Predicates of the form $x_i \in \mathcal{S}$ where $\mathcal{S}$ is semialgebraic defined by quadratic inequalities (e.g. component-wise non-negativity, groupwise max relations).

4. **Pooling and group activations.** Max-pooling, maxout, and GroupSort are encodable via max/min gadgets; any property phrased over such layers remains quadratic.

5. **Compositional properties.** Any finite Boolean combination (conjunction/disjunction via introduction of indicator variables and case splits) of the above preserves a quadratic presentation after standard algebraic encodings of disjunction.

8.4 Input Constraints: A Quadratic Toolkit

We collect quadratic encodings for common perturbation models (all variables are real and constraints are conjoined).

- ℓ_2-ball: $\sum_{i=1}^{m}(x_i - a_i)^2 \leq \varepsilon^2$.
- ℓ_∞-ball: $(x_i - a_i)^2 \leq \varepsilon^2$ for all i.
- ℓ_1-ball: $\exists u \geq 0 : \sum_i u_i \leq \varepsilon,\ (x_i - a_i)^2 \leq u_i^2$.
- Rational ℓ_p: reduce $|x_i - a_i|^p \leq s_i$ to degree-2 via auxiliary variables and identities (e.g., $t^3 \leq s \iff t^2 = r,\ rt \leq s$), preserving a quadratic system.

These encodings interact compositionally with Φ_{net} and logit-margin predicates.

8.5 Activation Expressivity and Semialgebraic Limits

Semialgebraic activations admit exact quadratic graphs. Any activation whose graph is defined by finitely many polynomial constraints can be captured *exactly* using quadratic relations plus auxiliaries. This covers ReLU, leaky-ReLU, maxout, GroupSort, and Softsign.

Sigmoid and tanh **are not semialgebraic.**

Theorem 8.5.1. *The activation* $\text{sigmoid}(x) = 1/(1 + e^{-x})$ *is not semialgebraic.*

Proof sketch. If its graph Γ_{sigmoid} were semialgebraic, the cell decomposition of semialgebraic sets would yield a finite union of pieces each cut out by a polynomial equation $P_i(x, y) = 0$ and strict polynomial inequalities in (x, y). Multiplying the P_i's gives a nonzero polynomial P with $P(x, \text{sigmoid}(x)) = 0$ for all x, contradicting that a nonzero bivariate polynomial cannot vanish on the transcendental curve $y = \text{sigmoid}(x)$ (one formalizes this via differentiation and linear independence of $\{e^{-kx}\}_k$ over the reals). The same argument applies to tanh and to piecewise definitions containing an exponential branch. □

Approximation by quadratic-encodable envelopes.

Theorem 8.5.2. *For any continuous* $f : \mathbb{R} \to \mathbb{R}$ *and* $\epsilon > 0$ *there exist piecewise-linear* $g_1 \leq f \leq g_2$ *with* $|f - g_i| < \epsilon$ *pointwise; each* g_i *can be implemented exactly by ReLU gadgets and hence by a conjunction of quadratic constraints.*

8.6 Decidability via the First-Order Theory of the Reals

The symbolic encodings above live in the first-order theory of real closed fields (polynomials with $+, \times, \leq$, quantifiers). This yields an immediate meta-theorem.

Theorem 8.6.1. *Assume all network parameters and perturbation radii are algebraic numbers. For networks whose activations are semialgebraic (e.g., ReLU, leaky-ReLU, maxout, GroupSort, Softsign), any verification task that can be posed as a first-order sentence over their symbolic unfolding (e.g., existence of adversarial examples in $B_p(a, \varepsilon)$, layer invariants, output dominance on a semialgebraic set) is* decidable.

Discussion. Despite uncountably many inputs, decidability follows from Tarski–Seidenberg. The algebraicity assumption is benign in practice (rational parameters). For networks with non-semialgebraic activations (e.g., sigmoid), decidability falls outside this theory; the first-order theory of the reals with exponentiation is a central open problem in model theory. Approximating such activations by quadratic-encodable envelopes brings these tasks back within the semialgebraic setting at the price of controlled approximation error.

8.7 Summary and Scope

We presented a uniform method to interpret neural networks *symbolically* as systems of quadratic (degree ≤ 2) constraints, capturing affine computation, a broad family of activations, common pooling/group layers, and standard perturbation models. Many verification and analysis questions—local adversarial reachability, output dominance on regions, layer invariants, and slope-restricted global sensitivity statements—admit direct formulations as first-order properties over these quadratic systems. This algebraic viewpoint provides exact encodings for semialgebraic activations and principled approximation schemes for transcendental ones, and it leads to general decidability results without invoking any optimization machinery. In the next chapter we will build on the same symbolic foundation to study algorithmic estimators that operate on these quadratic encodings.

This chapter developed an exact symbolic interpretation of neural networks by encoding their computations as systems of polynomial constraints. By moving from set abstraction to algebraic structure, we obtained a precise semantic model capable of expressing verification and robustness queries as first-order properties over the reals. This algebraic lens highlights the dual roles of expressiveness and complexity, and sets the stage for leveraging convex relaxation techniques. The next chapter builds on these encodings to construct semidefinite relaxations that quantify global robustness through geometric and duality-based principles.

Chapter 9

Semidefinite Relaxation and Quantitative Guarantees

This chapter develops the computational side of the algebraic encodings introduced in Chapter 8, focusing exclusively on *global* (dataset-independent) robustness via the formal global Lipschitz (FGL) constant. While Tarski's quantifier-elimination implies decidability for many algebraic encodings, its double-exponential complexity is impractical. We therefore adopt semidefinite relaxations (SDPs) of the quadratic programs (QPs) that arise from symbolic/algebraic interpretations, and show that these relaxations admit *constant-factor* approximation guarantees stemming from Banach-space geometry.

9.1 From Algebraic Encodings to a Single SDP Formulation

Two-layer setup and QCQP form. Consider a one-hidden-layer network

$$f(x) = u\,\mathrm{diag}(y)\,Wx,$$

with $W \in \mathbb{R}^{n\times m}$, $u \in \mathbb{R}^{1\times n}$, and diagonal $\mathrm{diag}(y)$ encoding piecewise-linear activation slopes (e.g., ReLU with entries in $\{0,1\}$). For the ℓ_p-FGL (with Hölder dual q), the global objective reduces to

$$\mathrm{FGL}_q(f) = \max_{y\in\{0,1\}^n} \left\| W^\top \mathrm{diag}(y)\, u^\top \right\|_q = \max_{y\in\{0,1\}^n} \|Ay\|_q, \qquad A := W^\top \mathrm{diag}(u). \tag{9.1}$$

For $q = 2$ this is the positive-semidefinite (PSD) quadratic optimization

$$\max_{y\in\{0,1\}^n} y^\top (A^\top A) y. \tag{9.2}$$

For $q = 1$, the mixed-norm form is

$$\max_{y\in\{0,1\}^n} \|Ay\|_1 = \max_{\substack{y\in\{0,1\}^n \\ s\in\{-1,1\}^m}} s^\top Ay. \tag{9.3}$$

A unified Shor lifting. Both cases can be lifted to a single semidefinite program via Shor's relaxation. Illustratively, for (9.2) let $M = A^\top A \succeq 0$ and lift $X = yy^\top$:

$$\max_{X\succeq 0} \langle M, X\rangle \quad \text{s.t.} \quad X_{ii} = 1,\ i \in [n],$$

where we dropped the nonconvex rank-one constraint $X = yy^\top$. For (9.3), after the standard $\{-1, 1\}$ reparametrization of the 0–1 cube, the bilinear form $s^\top At$ is handled by the same lifting on the joint sign vector. Concretely, define

$$\begin{aligned} \max_{X\succeq 0} \quad & \tfrac{1}{2}\operatorname{tr}(BX) \\ \text{s.t.} \quad & X_{ii} = 1, \quad i \in [n+m+1], \end{aligned} \tag{9.4}$$

for an explicit block matrix B assembled from A and the affine $\{-1, 1\}$-shift (details below in Section 9.2); (9.4) is the canonical SDP upper bound for ℓ_∞–FGL. In all cases, the resulting SDPs are solvable in polynomial time.

Dual (Lagrangian) view. An equivalent route introduces Lagrange multipliers for the (quadratic) activation-slope constraints obtained in Chapter 8. The resulting *dual* SDP enforces a single linear matrix inequality (LMI) whose feasibility certifies an FGL upper bound. This dual is mathematically equivalent to the Shor (primal) relaxation above; the two views differ only by perspective: primal encodes feasible vector configurations, dual encodes optimality certificates.

9.2 Quantitative Guarantees via Banach Geometry

We now state constant-factor bounds for the SDP objectives in the two principal norms. These guarantees are dimension-free and arise from classical inequalities in Banach-space geometry.

ℓ_∞–FGL ($q = 1$): Grothendieck-tightness

Starting from (9.3), reparametrize $y_i = (t_i + 1)/2$ with $t \in \{-1, 1\}^n$, introduce an auxiliary sign $\tau \in \{-1, 1\}$, and write

$$\max_{y\in\{0,1\}^n} \|Ay\|_1 = \max_{(t,\tau,s)\in\{-1,1\}^{n+1+m}} s^\top A\,(t + \tau e_n) = \tfrac{1}{2} \max_{z\in\{-1,1\}^{n+1+m}} \langle B, zz^\top\rangle,$$

for the explicit block matrix

$$B = \begin{pmatrix} 0 & 0 & 0 \\ A & Ae_n & 0 \end{pmatrix}.$$

Its Shor relaxation is (9.4). By the real Grothendieck inequality, for any real matrix C,

$$\max_{x_i,y_j\in\{-1,1\}} \textstyle\sum_{ij} C_{ij}x_iy_j \;\le\; \max_{\|u_i\|,\|v_j\|\le 1} \sum_{ij} C_{ij}\langle u_i, v_j\rangle \;\le\; K_G \max_{x_i,y_j\in\{-1,1\}} \sum_{ij} C_{ij}x_iy_j,$$

with Grothendieck constant $1.676 < K_G < 1.783$. The SDP objective corresponds to the vector (Gram) relaxation on the middle term, hence:

Theorem 9.2.1 (Constant-factor SDP for ℓ_∞–FGL)**.** *For any one-hidden-layer network with piecewise-linear activations, the SDP* (9.4) *yields an upper bound on* $\mathrm{FGL}_1(f)$ *within multiplicative factor* $K_G < 1.783$ *of the true optimum. The bound is computable in polynomial time.*

ℓ_2–FGL ($q = 2$): PSD rounding

For (9.2) let $M = A^\top A \succeq 0$ and apply the standard $\{-1, 1\}$ shift (augment by a coordinate corresponding to e_n) to obtain

$$\max_{y\in\{0,1\}^n} y^\top My = \tfrac{1}{4} \max_{x\in\{-1,1\}^{n+1}} x^\top \hat{M}x, \qquad \hat{M} = \begin{pmatrix} M & Me_n \\ e_n^\top M & e_n^\top Me_n \end{pmatrix} \succeq 0.$$

The Shor relaxation is

$$\max_{X\succeq 0} \tfrac{1}{4}\,\mathrm{tr}(\hat{M}X) \quad \text{s.t.} \quad X_{ii} = 1,\ i \in [n+1]. \tag{9.5}$$

Nesterov's rounding for PSD forms over the hypercube implies a $\pi/2$ ratio on the *squared* objective, hence a $\sqrt{\pi/2}$ ratio on norms:

Theorem 9.2.2 (Constant-factor SDP for ℓ_2–FGL)**.** *For any one-hidden-layer network with piecewise-linear activations, the SDP* (9.5) *estimates* $\mathrm{FGL}_2(f)$ *within factor* $\sqrt{\pi/2} \approx 1.253$ *of the true optimum. The bound is computable in polynomial time.*

Complexity-theoretic tightness. Improving the constants in Theorems 9.2.1 and 9.2.2 in a black-box manner would contradict known hardness results for cut norms / Max-Cut–type objectives. Thus, under standard assumptions, these SDPs are optimal up to constant factors.

9.3 Beyond Two Layers: Practical Relaxations and Duality

Tensor vs. Jacobian relaxations. For a 3-layer network

$$f(x) = u\,\mathrm{diag}(Z)\,V\,\mathrm{diag}(Y)\,Wx,$$

the FGL objective becomes a trilinear form

$$\max_{X,Y,Z\in\{0,1\}} \sum_{i,j,k} \mathcal{W}_{ijk} x_i y_j z_k, \qquad \mathcal{W}_{ijk} = W_{ji} V_{kj} u_k,$$

i.e., a tensor cut-norm instance. Constant-factor approximation for general tensor cut norms is open. Two scalable relaxations are commonly used:

1. *Tensor (flattening) relaxation:* matricize the tensor along a mode and apply the two-layer SDP bound. This is more precise but less scalable (variables grow with the product of hidden widths).

2. *Jacobian relaxation:* decompose the network into two-layer subnetworks and use sub-multiplicativity, e.g.,

   $$\|W^\top \mathrm{diag}(Y)\,V^\top \mathrm{diag}(Z) u^\top\|_1 \;\le\; \|W^\top \mathrm{diag}(Y)\,V^\top\|_1 \cdot \|u^\top\|_1.$$

   This yields layerwise SDPs with K_G or $\sqrt{\pi/2}$ factors per block, trading some tightness for scalability.

Primal–dual correspondence. The Shor (primal) SDPs above are dual to the Lagrangian-multiplier LMIs obtained by relaxing the slope-restricted activation constraints (cf. Chapter 8). In practice, both forms produce numerically identical bounds for FGL; the choice is largely implementational (solver interface, certificate style).

Summary of guarantees. For one-hidden-layer networks, ℓ_∞–FGL admits a K_G-approximation and ℓ_2–FGL admits a $\sqrt{\pi/2}$-approximation, both in polynomial time. For deeper networks, tensor flattening and Jacobian decompositions propagate these constants while controlling scalability.

Notation recap used in this chapter.

- $\mathrm{diag}(\cdot)$: diagonal operator collecting activation slopes (entries in $[0, 1]$ for ReLU; in global analysis they are treated independently).

- $A = W^\top \mathrm{diag}(u)$; $M = A^\top A$; e_n: all-ones vector of length n.
- Shor lift: replace $yy^\top$ by $X \succeq 0$ with $X_{ii} = 1$; drop rank-one constraint.

9.4 Evaluation

We implemented all proposed algorithms in MATLAB [40], using the CVX convex-optimization framework [13] and the MOSEK solver [4]. The resulting prototype system, named **GeoLIP**, provides an end-to-end implementation of the semidefinite relaxations and their dual formulations developed in this chapter.

To assess both theoretical soundness and practical performance, we designed our empirical study around the following three research questions:

RQ1: Precision: How accurate are GeoLIP's bounds compared to existing methods?

RQ2: Scalability: How does GeoLIP perform computationally as network size and layer width increase?

RQ3: Dual correctness: Do the dual programs implemented in GeoLIP yield the same optimal values as their primal counterparts?

Experimental details and quantitative results are presented in Appendix B.1. Empirically, GeoLIP consistently achieves tighter Lipschitz bounds than prior approaches while maintaining favorable runtime scaling, and the dual formulations are verified to be numerically equivalent to their primal SDPs, confirming their theoretical validity.

This chapter connected algebraic encodings to convex optimization, deriving semidefinite relaxations with provable approximation guarantees for global Lipschitz constants. These SDPs reveal a deep alignment between symbolic semantics, geometric structure, and Lagrangian duality, providing quantitative tools for robustness analysis. However, solving SDPs at scale remains computationally challenging, motivating more incremental and first-order alternatives. The next part develops such incremental approaches, beginning with an iterative SDP solver and then extending the perspective to discrete prompt optimization.

Part IV

Incremental Computation for Neural Networks and Language Models

This part develops the dissertation's third principle: *incremental computation.* Where Parts II and III focused on semantic interpretation and duality, Part IV shows how difficult optimization and verification tasks can be solved by organizing them into *structured trajectories* of simpler problems. The key idea is that many hard questions in robustness, verification, and prompting do not need to be solved in a single leap; instead, one can incrementally traverse a sequence of related problems that gradually transform an easy instance into the target instance of interest.

Incremental computation arises naturally in both discrete and continuous settings. On the discrete side, adversarial prompt generation for large language models is a combinatorial optimization problem over token sequences, and is NP-hard even in highly restricted network settings. Yet practical jailbreak attacks succeed because humans and algorithms rarely attack a model *directly*; rather, they adaptively explore simplified versions of the model, weakening alignment constraints and discovering transferable structures. On the continuous side, global robustness estimation can be framed as a nonsmooth semidefinite optimization problem. Direct solutions require expensive interior-point methods, but one can instead follow a sequence of incrementally refined certificates whose structure mirrors the dominant eigenvectors of the relaxation.

Part IV formalizes these ideas in two complementary contributions. The first is *LipDiff*, an incremental algorithm for semidefinite relaxations of global Lipschitz constants. LipDiff begins from a spectral initialization and performs first-order updates that track the critical eigenvectors of a lifted quadratic form. Each iterate produces a valid semidefinite certificate, and successive updates monotonically tighten the Lipschitz bound. This incremental view sheds light on the geometry of semidefinite relaxations and offers a scalable alternative to full interior-point methods, while preserving the theoretical guarantees developed in Part III.

The second contribution is *functional homotopy* (FH) for adversarial prompting. FH treats model parameters and token sequences as joint inputs to a functional loss, and constructs a trajectory of models by progressively "de-robust-training" the base model into a series of weaker checkpoints. Prompt optimization is carried out along this trajectory: each intermediate model yields an easier subproblem whose solution seeds the next. This approach converts a single hard search over the base model's alignment barrier into a chain of easier searches, demonstrating substantial empirical gains over gradient-based and gradient-free baselines.

Structure of Part IV. Chapter 10 develops incremental semidefinite computation for global Lipschitz bounds. It describes the LipDiff algorithm, its spectral initialization, and its monotone refinement of semidefinite certificates. The chapter connects LipDiff's behavior to the geometric and duality principles of Part III, illustrating how incremental computation yields tractable approximations to semidefinite relaxations.

Chapter 11 introduces functional homotopy as an incremental method for adversarial prompting. It formalizes the NP-hardness of model-agnostic input generation, analyzes the limitations of token-gradient heuristics, and presents the FH algorithm. The chapter concludes with a detailed evaluation comparing FH to GCG, GR, and AutoDAN across multiple instruction-tuned LLMs.

Together, these chapters demonstrate how incremental computation provides a powerful and general methodological bridge between discrete optimization, continuous relaxations, and the semantic perspectives developed throughout this dissertation.

Chapter 10

First-Order Algorithms for LipSDP via Exact-Penalty Eigenvalue Reformulation

10.1 Motivation and Transition from Algebraic SDP Bounds

The previous chapter established an algebraic (quadratic) encoding of global robustness queries, and showed how Shor/Lagrangian semidefinite relaxations yield tight, polynomial-time computable upper bounds on formal global Lipschitzness (FGL). Semidefinite programs (SDPs) have thus emerged as a principled way to obtain *less conservative* Lipschitz bounds than naive matrix-norm products, with compelling quantitative guarantees (e.g., Grothendieck-type constants for $\ell_\infty \to \ell_1$ and $\sqrt{\pi/2}$ approximations for ℓ_2 cases).

However, the dominant general-purpose solution paradigm for SDPs—interior-point methods (IPMs)—is inherently *second order*. While they enjoy polynomial worst-case complexity, their memory footprint and per-iteration cost scale poorly with modern networks (tens to hundreds of thousands of activation variables), making them impractical beyond small architectures. Recent progress exploiting chordal sparsity helps but still struggles at CIFAR-scale multilayer networks.

This chapter develops a *first-order friendly* alternative: we transform the relevant LipSDP formulations into *nonsmooth eigenvalue* optimization via an *exact penalty* reformulation. The resulting problems require only matrix–vector products with a fixed sparse block matrix and admit subgradient, bundle, and Lanczos-type routines that (i) integrate naturally with autodiff frameworks, (ii) use dramatically less memory than IPMs, and (iii) admit warm-starts that exactly recover the spectral product bound at initialization and monotonically improve thereafter. Crucially, the exact-penalty construction preserves the *same* Lipschitz upper bound as the original SDP.

Organization. We first recall (single- and multi-layer) LipSDP in both dual and primal/Shor forms, then derive exact-penalty eigenvalue programs that are provably equivalent in optimal value. We cover scalar and vector outputs, and conclude with implementation-oriented techniques (Lanczos eigenvalue approximation, sparse mat–vecs, and an analytical initialization that recovers the weight-norm product bound).

Notation. I and 0 denote identity and zero matrices of compatible size. For $v \in \mathbb{R}^n$, $\operatorname{diag}(v)$ is the diagonal matrix with diagonal v; $\|v\|$ is the Euclidean norm. For a matrix M, $\|M\|$ is the operator norm. For a symmetric matrix A, $\lambda_{\max}(A)$ is its largest eigenvalue and $\lambda^{+}_{\max}(A) := \max\{0, \lambda_{\max}(A)\}$.

10.2 LipSDP: Dual and Primal (Shor) Forms

We begin with the single-hidden-layer, scalar-output case

$$f(x) = v\,\sigma(Wx + b^0) + b^1, \quad x \in \mathbb{R}^{n_x},\; W \in \mathbb{R}^{n\times n_x},\; v \in \mathbb{R}^{1\times n},$$

where σ is slope-restricted on $[0,1]$ (e.g., ReLU in its slope-restricted form). The classical *dual* LipSDP from Fazlyab et al. [18] is

$$\begin{aligned} \min_{\zeta,\tau}\quad & \zeta \\ \text{s.t.}\quad & \begin{bmatrix} -\zeta I & W^{\mathsf{T}}\operatorname{diag}(\tau) \\ \operatorname{diag}(\tau)W & -2\operatorname{diag}(\tau) + v^{\mathsf{T}}v \end{bmatrix} \preceq 0, \qquad \tau \ge 0,\; \zeta \ge 0. \end{aligned} \tag{10.1}$$

By the standard quadratic constraints argument, feasibility certifies $\|f(x') - f(x)\| \le \sqrt{\zeta}\,\|x' - x\|$, so the optimal objective gives a squared Lipschitz bound.

A *primal* QCQP encoding the same Lipschitz bound (up to the standard scaling argument) is:

$$\begin{aligned} \max_{\Delta x,\Delta z}\quad & v\,\Delta z \\ \text{s.t.}\quad & (\Delta z_i - W_i\Delta x)\,\Delta z_i \le 0, \quad \forall i \in [n], \\ & \|\Delta x\| \le 1. \end{aligned} \tag{10.2}$$

Shor's relaxation of (10.2) yields an SDP whose *dual* form is (see, e.g., [52, 53])

$$\begin{aligned} \min_{\zeta,\tau,\gamma} \quad & \zeta \\ \text{s.t.} \quad & \begin{bmatrix} \gamma-\zeta & 0 & v \\ 0 & -\gamma I & W^{\mathsf{T}}\mathrm{diag}(\tau) \\ v^{\mathsf{T}} & \mathrm{diag}(\tau)W & -2\mathrm{diag}(\tau) \end{bmatrix} \preceq 0, \qquad \zeta \geq 0,\ \tau \geq 0,\ \gamma \geq 0. \end{aligned} \tag{10.3}$$

By Schur-complement manipulations, (10.1) and (10.3) produce the same Lipschitz upper bound (modulo a consistent scaling of ζ).

10.3 Exact-Penalty Eigenvalue Reformulation (Single Layer)

A naive penalty on the LMI in (10.1) would read

$$\min_{\zeta\geq 0,\ \tau\geq 0} \ \zeta \ + \ \rho\,\lambda_{\max}^{+}\left(\begin{bmatrix} -\zeta I & W^{\mathsf{T}}\mathrm{diag}(\tau) \\ \mathrm{diag}(\tau)W & -2\mathrm{diag}(\tau)+v^{\mathsf{T}}v \end{bmatrix}\right), \tag{10.4}$$

but it is unclear how to pick a finite ρ guaranteeing exactness. To obtain an *exact* penalty with an explicit finite constant, we add a redundant but valid bound to the primal QCQP (10.2):

$$\begin{aligned} \max_{\Delta x,\Delta z} \quad & v\,\Delta z \\ \text{s.t.} \quad & (\Delta z_i - W_i\Delta x)\,\Delta z_i \leq 0, \quad \forall i\in[n], \\ & \|\Delta x\| \leq 1, \\ & \Delta z_i^2 \leq \|W_i\|^2, \quad \forall i\in[n]. \end{aligned} \tag{10.5}$$

Let $a_i := \|W_i\|^2$. Shor's relaxation of (10.5) now has an *explicit trace bound* $1+\|\Delta x\|^2+\|\Delta z\|^2 \leq 2+\sum_{i=1}^{n} a_i$, which yields the following dual SDP:

$$\begin{aligned} \min_{\zeta,\lambda,\tau,\gamma} \quad & \zeta \\ \text{s.t.} \quad & \begin{bmatrix} \sum_{j=1}^{n} a_j\lambda_j+\gamma-\zeta & 0 & v \\ 0 & -\gamma I & W^{\mathsf{T}}\mathrm{diag}(\tau) \\ v^{\mathsf{T}} & \mathrm{diag}(\tau)W & -2\mathrm{diag}(\tau)-\mathrm{diag}(\lambda) \end{bmatrix} \preceq 0, \\ & \zeta\geq 0,\ \lambda\geq 0,\ \tau\geq 0,\ \gamma\geq 0. \end{aligned} \tag{10.6}$$

Define the block matrix

$$C(\zeta,\lambda,\tau,\gamma) := \begin{bmatrix} \sum_{j=1}^{n} a_j\lambda_j + \gamma - \zeta & 0 & v \\ 0 & -\gamma I & W^{\mathsf{T}}\mathrm{diag}(\tau) \\ v^{\mathsf{T}} & \mathrm{diag}(\tau)W & -2\mathrm{diag}(\tau) - \mathrm{diag}(\lambda) \end{bmatrix}.$$

Using the explicit trace bound, the *exact-penalty* form of (10.6) is

$$\min_{\zeta\geq 0,\ \lambda\geq 0,\ \tau\geq 0,\ \gamma\geq 0} \ \zeta \ + \ \Big(2 + \sum_{j=1}^{n} a_j\Big)\, \lambda_{\max}^{+}\Big(C(\zeta,\lambda,\tau,\gamma)\Big). \tag{10.7}$$

Theorem 10.3.1. *Let* opt_1 *be the optimal value of* (10.6)*,* opt_2 *that of* (10.7)*, and* opt_3 *that of* (10.1)*. Then* $opt_1 = opt_2 = 2\sqrt{opt_3}$*. Hence* (10.1) *and* (10.7) *produce the same Lipschitz upper bound.*

Proof. (Unchanged in structure and details from the original derivation.) The trace-bounded Shor dual ensures an exact penalty with coefficient equal to the trace cap; equivalence to (10.1) follows from duality and the usual Schur-complement scaling of ζ. □

Practical consequence. Problem (10.7) replaces the LMI by a single nonsmooth term $\lambda_{\max}^{+}(C)$, enabling first-order methods that need only the mapping $x \mapsto Cx$ and subgradients of $\lambda_{\max}$. This dramatically lowers memory and improves scalability.

10.4 Multi-Layer Networks and Vector Outputs

Consider a d-layer scalar-output network

$$x^{(0)} = x,\ \ x^{(k)} = \phi\Big(W^{(k-1)}x^{(k-1)} + b^{(k-1)}\Big),\ k = 1,\dots,d,\ \ f(x) = v\,x^{(d)} + b^{(d)}, \tag{10.8}$$

with $x^{(k)} \in \mathbb{R}^{n_k}$. A dual form equivalent (in value) to the original multi-layer LipSDP [18] is

$$\begin{aligned} \min_{\zeta,\gamma,\{\tau^{(k)}\}} \quad & \zeta \\ \text{s.t.} \quad & \zeta \geq 0,\ \gamma \geq 0,\ \Lambda_k = \mathrm{diag}(\tau^{(k)}),\ \tau^{(k)} \geq 0\ \ (k = 1,\dots,d), \\ & \begin{bmatrix} \gamma - \zeta & 0 & 0 & 0 & \cdots & v \\ 0 & -\gamma I & (W^{(0)})^{\mathsf{T}}\Lambda_1 & 0 & \cdots & 0 \\ 0 & \Lambda_1 W^{(0)} & -2\Lambda_1 & \ddots & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & (W^{(d-2)})^{\mathsf{T}}\Lambda_{d-1} & 0 \\ \vdots & \vdots & \ddots & \Lambda_{d-1}W^{(d-2)} & -2\Lambda_{d-1} & (W^{(d-1)})^{\mathsf{T}}\Lambda_d \\ v^{\mathsf{T}} & 0 & 0 & 0 & \Lambda_d W^{(d-1)} & -2\Lambda_d \end{bmatrix} \preceq 0. \end{aligned} \tag{10.9}$$

Let $W_j^{(k)}$ be the j-th row of $W^{(k)}$, and define

$$c_{kj} \;:=\; \left\|W_j^{(k-1)}\right\|^2 \cdot \prod_{i=0}^{k-2} \left\|W^{(i)}\right\|^2 \qquad (k = 1, \ldots, d;\; j = 1, \ldots, n_k).$$

As in the single-layer case, these induce an explicit trace bound and an exact penalty. Introduce $S_k = \operatorname{diag}(\lambda^{(k)})$ with $\lambda^{(k)} \geq 0$ and set

$$C \;:=\; \begin{bmatrix} \sum_{k=1}^{d}\sum_{j=1}^{n_k} c_{kj}\lambda_j^{(k)} + \gamma - \zeta & 0 & 0 & 0 & \cdots & v \\ 0 & -\gamma I & (W^{(0)})^{\mathsf{T}}\Lambda_1 & 0 & \cdots & 0 \\ 0 & \Lambda_1 W^{(0)} & -2\Lambda_1 - S_1 & \ddots & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & (W^{(d-2)})^{\mathsf{T}}\Lambda_{d-1} & 0 \\ \vdots & \vdots & \ddots & \Lambda_{d-1}W^{(d-2)} & -2\Lambda_{d-1} - S_{d-1} & (W^{(d-1)})^{\mathsf{T}}\Lambda_d \\ v^{\mathsf{T}} & 0 & 0 & 0 & \Lambda_d W^{(d-1)} & -2\Lambda_d - S_d \end{bmatrix}. \tag{10.10}$$

Then the multi-layer exact-penalty program is

$$\min_{\zeta,\gamma,\ \{\lambda^{(k)}\geq 0\},\ \{\tau^{(k)}\geq 0\}} \quad \zeta \;+\; \Big(2 + \sum_{k=1}^{d}\sum_{j=1}^{n_k} c_{kj}\Big)\, \lambda_{\max}^{+}\big(C\big) \quad \text{s.t.} \quad \zeta \geq 0,\ \gamma \geq 0. \tag{10.11}$$

Theorem 10.4.1. *Let opt_4 be the optimal value of* (10.9)*, opt_5 that of* (10.11)*, and opt_6 that of the original multi-layer LipSDP. Then $opt_4 = opt_5 = 2\sqrt{opt_6}$. Hence* (10.11) *and LipSDP give the same Lipschitz upper bound.*

Proof. Identical in structure to Theorem 10.3.1: the added diagonal blocks S_k encode redundant componentwise bounds that yield an explicit trace cap, enabling an exact penalty with the displayed coefficient; dual equivalence to the original LipSDP follows by the same Schur-complement scaling argument. □

Vector outputs. For $f(x) = V\,\sigma(Wx + b^0) + b^1 \in \mathbb{R}^m$ (single layer), the primal QCQP reads

$$\begin{aligned} \max_{\Delta x, \Delta z} \quad & \|V\,\Delta z\|^2 \\ \text{s.t.} \quad & (\Delta z_i - W_i \Delta x)\,\Delta z_i \leq 0, \quad \forall i, \\ & \|\Delta x\| \leq 1, \\ & \Delta z_i^2 \leq \|W_i\|^2, \quad \forall i. \end{aligned} \tag{10.12}$$

The Shor dual becomes

$$\begin{aligned}
\min_{\zeta,\lambda,\tau,\gamma}\quad & \zeta \\
\text{s.t.}\quad & \begin{bmatrix} \sum_{j=1}^{n} a_j\lambda_j + \gamma - \zeta & 0 & 0 \\ 0 & -\gamma I & W^{\mathsf{T}}\mathrm{diag}(\tau) \\ 0 & \mathrm{diag}(\tau)W & -2\mathrm{diag}(\tau) - \mathrm{diag}(\lambda) + V^{\mathsf{T}}V \end{bmatrix} \preceq 0, \qquad (10.13)\\
& \zeta \geq 0,\ \lambda \geq 0,\ \tau \geq 0,\ \gamma \geq 0,
\end{aligned}$$

and—with the same trace bound $2 + \sum_j a_j$—the exact-penalty form is

$$\min_{\zeta,\lambda,\tau,\gamma}\ \zeta + \left(2+\sum_{j=1}^{n} a_j\right) \lambda_{\max}^{+}\left(\begin{bmatrix} \sum_{j=1}^{n} a_j\lambda_j + \gamma - \zeta & 0 & 0 \\ 0 & -\gamma I & W^{\mathsf{T}}\mathrm{diag}(\tau) \\ 0 & \mathrm{diag}(\tau)W & -2\mathrm{diag}(\tau) - \mathrm{diag}(\lambda) + V^{\mathsf{T}}V \end{bmatrix}\right), \qquad (10.14)$$

which is equivalent in optimal value to the vector-output LipSDP.

10.5 First-Order Implementation: Eigenvalue-Oriented Structure

The exact-penalty objectives in (10.7), (10.11), and (10.14) decompose into a simple convex scalar term (ζ) and a nonsmooth spectral term $\lambda_{\max}^{+}(\cdot)$. This structure is well suited to scalable first-order optimization once three implementation aspects are exploited:

(i) Lanczos approximation of the leading eigenvalue. Evaluating $\lambda_{\max}$ and obtaining a subgradient via a leading eigenvector requires only the linear map $u \mapsto Cu$. Krylov methods, in particular the Lanczos iteration, yield accurate approximations from a low-dimensional subspace, with computational cost dominated by a small number of sparse matrix–vector products with the fixed block-sparse matrix C induced by the network architecture.

(ii) Native sparse matrix–vector kernels. The block pattern of C mirrors layerwise connectivity and is highly sparse (nearly banded for standard feedforward topologies). Implementing $u \mapsto Cu$ without forming C explicitly reduces memory and improves throughput, especially for deep and wide networks. All operations in the Lanczos loop are then expressed as calls to these kernels.

(iii) Analytical initialization that recovers the norm-product bound. A constructive choice of $(\zeta, \gamma, \tau, \lambda)$ makes C negative semidefinite at initialization and certifies the classical spectral product bound. For the single-layer, scalar-output case, set

$$\gamma := \|W\|^2, \qquad \tau := 0, \qquad \lambda := 0, \qquad \zeta := \|v\|^2\,\|W\|^2.$$

The block diagonal of C then reduces to $\operatorname{diag}(\gamma - \zeta,\ -\gamma I,\ -2 \cdot 0)$ plus rank-two couplings from v and W. A direct Schur-complement calculation gives $\lambda_{\max}^{+}(C) = 0$, hence the objective equals $\zeta = \|v\|^2\|W\|^2$ and the induced Lipschitz bound is $\|v\|\,\|W\|$. For d layers, the analogous choice $\gamma := \|W^{(0)}\|^2$, $\Lambda_k := 0$, $S_k := 0$, and $\zeta := \|v\|^2 \prod_{i=0}^{d-1} \|W^{(i)}\|^2$ recovers the product bound in (10.11). Consequently, the first iterate matches the naive certificate and subsequent iterations monotonically improve it.

Autodiff-compatible subgradients. The mapping $(\zeta, \lambda, \tau, \gamma) \mapsto \lambda_{\max}(C)$ is convex and differentiable almost everywhere; whenever the leading eigenvalue is simple, a unit leading eigenvector u yields the subgradient $u^\top(\partial C)u$. This enables direct integration with automatic differentiation frameworks for end-to-end first-order optimization.

10.6 Evaluation

We empirically evaluate the first-order, exact-penalty formulation to assess numerical equivalence with LipSDP and to quantify scalability gains in time and memory. The study is organized around the following questions:

RQ1: Does the eigenvalue-penalty formulation attain the same Lipschitz bound as LipSDP on the same instances?

RQ2: Does the first-order method offer advantages in running time and memory compared to LipSDP?

RQ3: What is the marginal contribution of each engineering component (analytical initialization, Lanczos approximation, sparse mat–vec kernels)?

Implementation. We implemented the algorithm in PyTorch and refer to the tool as **LipDiff**. The objective is (10.11) (or its single-layer/vector-output variants), evaluated via Lanczos with user-controlled subspace size. All decision variables appearing in (10.10) are optimized by a first-order routine; the default optimizer is ADAM, though other schedules and optimizers can be substituted without changing the formulation.

Findings (summary). Detailed experimental results appear in Appendix B.2; we summarize the main observations here.

Algorithm 1 LipDiff (first-order exact-penalty solver for LipSDP)

```
Input: weight matrices $[W_i]_{i=1}^d$; Hyperparameters: iterations $N$, Lanczos steps $\ell$, step size $\alpha$.
Output: upper bound on the network Lipschitz constant.
1: Initialize $(\zeta, \gamma, \{\Lambda_k\}, \{S_k\})$ using the analytical construction in §10.5 (norm-product certificate).
2: Instantiate a first-order optimizer (e.g., ADAM) on the decision variables with step size $\alpha$.
3: for $t = 1, \dots, N$ do
4:     Form the linear operator $u \mapsto Cu$ defined by (10.10) using native sparse mat–vec kernels.
5:     Compute an $\ell \times \ell$ Lanczos tridiagonal surrogate of $C$ and its largest eigenpair.
6:     Define the loss (objective) via (10.11) with $\lambda_{\max}$ replaced by its Lanczos approximation.
7:     Backpropagate to obtain (sub)gradients and update the decision variables.
8: end for
9: return final objective value; convert to a Lipschitz bound via Theorems 10.3.1 and 10.4.1.
```

RQ1 (value equivalence). Across all instances tested, the eigenvalue-penalty formulation matches the bounds from LipSDP to within solver tolerance, consistent with Theorems 10.3.1 and 10.4.1.

RQ2 (scalability). *Memory.* LipDiff's memory usage scales essentially linearly with the size of the SDP block in (10.10), since only operator evaluations $u \mapsto Cu$ and small Lanczos bases are stored. In contrast, IPM-based LipSDP requires storing dense factorizations whose footprint grows superlinearly, becoming prohibitive for blocks on the order of a few thousand. *Time.* Because iteration budgets and step sizes are tunable, LipDiff attains high-quality certificates rapidly; the ability to terminate early with a valid bound provides additional flexibility not available in closed-form IPM solves.

RQ3 (ablation). Analytical initialization consistently improves final bounds and convergence, especially on larger networks with many free variables. Lanczos approximation substantially reduces runtime with negligible impact on the final value at moderate subspace sizes. Sparse mat–vec kernels yield pronounced gains on deep/wide models; for smaller networks, their effect is modest, as expected.

Overall, the experiments support the theoretical conclusion that the exact-penalty eigenvalue reformulation preserves the tightness of LipSDP while providing a practical path to modern-scale models via first-order methods.

10.7 Summary

Before the introduction of LipDiff, the only practically scalable method for estimating the Lipschitz constants of large neural networks was the layerwise spectral-norm product. Although easy to compute, this bound is notoriously loose and often orders of magnitude larger than the true global sensitivity. Traditional SDP formulations yield much tighter bounds, but their computational cost renders them infeasible for modern architectures.

LipDiff closes this precision–scalability gap by introducing a first-order algorithm that begins from the analytical spectral-product initialization and incrementally moves toward the SDP optimum. This initialization ensures stability and recovers the classical bound in the first iterate, while each subsequent step refines the certificate without sacrificing tractability. The result is a continuous path of monotonically improving, sound upper bounds that interpolate between a coarse approximation and an essentially optimal semidefinite solution.

Incremental Computation Perspective. LipDiff exemplifies the broader paradigm of *incremental computation*, in which complex analyses are solved through a sequence of efficiently maintainable refinements. An incremental algorithm maintains a state that incorporates prior progress and can be updated without restarting the computation from scratch. This idea appears across program analysis (e.g., differential dataflow, self-adjusting computation), optimization (online convex methods, iterative refinement, streaming algorithms), and machine learning (stochastic gradient updates). In all of these settings, intermediate results are both valid and informative, enabling early stopping or adaptive resource allocation.

LipDiff fits naturally into this paradigm. Its iterates are certified Lipschitz upper bounds, each strictly tighter than the previous one, and each obtained at a fraction of the cost of a full SDP solve. This incremental refinement enables practitioners to trade computational effort for precision, scaling global robustness estimation to network sizes far beyond the reach of classical SDP solvers.

In the next chapter, we extend this incremental-computation viewpoint to discrete optimization, showing how the same philosophy can be used to smooth the hard, non-differentiable search problems that arise in adversarial prompting and model alignment for large language models.

Chapter 11

Homotopy Optimization as Incremental Computation: From Spectral Relaxation to Jailbreak Synthesis

11.1 Introduction: From Semidefinite Relaxations to Functional Homotopy

The previous chapter developed scalable first–order methods for semidefinite relaxations (SDPs) arising in global Lipschitz analysis. There, we replaced second–order interior–point machinery with a nonsmooth eigenvalue formulation and iterative, certificate–preserving updates. Two features were crucial: (i) *incrementality*, i.e., each iterate provided a valid (and often improving) bound; and (ii) *functional structure*, where model parameters entered continuously and enabled efficient first–order refinement.

This chapter extends that incremental viewpoint from continuous convex relaxations to *discrete* optimization over token sequences. Our focus is the synthesis of adversarial prompts ("jailbreaks") for large language models (LLMs). We show that naively transplanting gradient heuristics from continuous image domains to the discrete token domain is theoretically brittle and practically weak. We then introduce *functional homotopy* (FH): a principled homotopy strategy that lifts the static discrete problem into a *joint* space of model parameters and inputs. By first walking in the continuous parameter space to produce a family of easier objectives and then warm–starting the discrete search along this path, FH turns a hard combinatorial problem into a sequence of progressively harder—but tractable—subproblems. In this sense, FH is an instance of *incremental computation*: intermediate states are sound, informative, and reusable, and the final solution is reached by continuous deformation from easy to hard instances.

Positioning.. Classical adversarial optimization for vision relies on continuous ℓ_p perturbations and gradient steps such as FGSM [22] and PGD [39]. Prompt optimization for LLMs has adopted analogous gradient heuristics in embedding space (e.g., GCG [56]), but the underlying search remains over a *discrete* token set. We rigorously formalize the model–agnostic input–generation objective, establish its NP–hardness, and analyze when token–gradient selection coincides with a linearized (one–step) surrogate. This motivates a different algorithmic lever: rather than differentiating *the discrete input*, differentiate *the continuous model* to generate a homotopy of easier objectives, and then solve the discrete problems incrementally along that path.

11.2 Introduction: Incremental Computation in Optimization

Incremental computation refers to the strategy of solving a sequence of related computational problems by gradually refining intermediate results. In numerical optimization, such methods are especially valuable when the target problem is difficult to solve directly, but easier subproblems can be constructed and leveraged. In prior chapters, we saw this principle in action in the context of Lipschitz constant estimation via spectral bundle methods, where we incrementally refined coarse spectral estimates toward more accurate semidefinite relaxations.

This chapter explores a new direction for incremental computation: homotopy optimization in discrete spaces. Specifically, we apply it to the generation of adversarial prompts for language models, introducing the *functional homotopy* (FH) method. Our main insight is that the combinatorial hardness of prompt optimization can be mitigated by defining a sequence of easier optimization problems over a homotopy of model parameters.

Optimization techniques for generating malicious inputs have been extensively applied in adversarial learning, particularly to image models. The most prevalent methods include gradient-based approaches such as the Fast Gradient Sign Method (FGSM) [22] and Projected Gradient Descent (PGD) [39]. These techniques have demonstrated that many deep learning models exhibit vulnerability to small ℓ_p perturbations to the input. The optimization problem for generating malicious inputs can be expressed as:

$$\min_x f_p(x), \tag{11.1}$$

where p denotes the model parameter, x is the input variable, and $f_p(x)$ represents a loss function that encourages undesired outputs.

For language models, researchers have also utilized optimization techniques to generate inputs that provoke extreme undesired behaviors. Approaches analogous to those employed in adversarial learning have been adopted for this purpose. For example, Greedy Coordinate Gradient [56] (GCG) employs gradient-based methods to identify tokens that induce jailbreak behaviors. Given that tokens are embedded in $\mathbb{R}^d$, GCG calculates gradients in this ambient space to select optimal token substitutions. This methodology has also been adopted by other studies for related prompt synthesis challenges [26, 38].

Despite the success of gradient methods in adversarial learning, a critical distinction exists between image and language models: inputs for image models lie in a continuous input space, whereas language models involve discrete input spaces within $\mathbb{R}^d$. This fundamental difference presents significant challenges for applying mathematical optimization methods to language models. Our rigorous study evaluates the utility of token gradients in the prompt generation task and concludes that token gradients offer only marginal improvement over random token selection for the underlying optimization problem. Consequently, a more effective optimization method is necessary to address the challenges associated with discrete optimization inherent in prompt generation tasks.

In this chapter, we establish that the model-agnostic optimization problem for LLM input generation is NP-hard, implying that efficient optimization algorithms likely require exploitation of problem-specific structures. To address this challenge, we propose a novel optimization method tailored to the problem defined in Equation (11.1). This approach can be interpreted as utilizing the model-alignment property by intentionally modifying the model to reduce its alignment. Despite the discrete input space, the problem in Equation (11.1) exhibits a unique characteristic: the function f_p is parameterized by p, which lies in a continuous domain. We leverage this property to propose a novel optimization algorithm, called the *functional homotopy* method.

The *homotopy method* [17] involves gradually transforming a challenging optimization problem into a sequence of easier problems, utilizing the solution from the previous problem to *warm start* the optimization process of the next problem. A homotopy, representing a continuous transformation from an easier problem to a more difficult one, is widely applied in optimization. For instance, the well-known interior point method for constrained optimization by constructing a series of soft-to-hard constraints [8]. Various approaches exist for constructing a homotopy, such as employing parameterized penalty terms, as demonstrated in the interior point method, or incorporating Gaussian random noise [43].

In our functional homotopy (FH) method, we go beyond the conventional interpretation of f_p in Equation (11.1) as a *static* objective function, which was the perspective taken in previous work [56, 37, 26, 3]. Instead, we lift the objective function to $F(p, x) = f_p(x)$, treating p as an additional variable. Equation (11.1) thus becomes:

$$\min_x F(p, x). \tag{11.2}$$

Therefore, the objective $f_p(x)$ in Equation (11.1) represents a projection of $F(p, x)$ for a fixed value of p. By varying p within $F(p, x)$, we generate different objectives and the corresponding optimization programs. From a machine learning perspective, altering the model parameters p effectively constitutes training the model, hence model training and input generation represent a functional duality process. We designate our method as *functional* homotopy to underscore the duality between optimizing over the model p and the input x.

In the FH method for Equation (11.2), we first optimize over the continuous parameter p. Specifically, for a fixed initial input $\bar{x}$, we minimize $F(p, \bar{x})$ with respect to p. We employ gradient descent to update p until a desired value of $F(p', \bar{x})$ is achieved. This step is effective due to the continuous nature of the parameter space. As the parameter p is iteratively updated in this process, we retain all intermediate states of the parameter, denoted as $p_0 = p, p_1, \ldots, p_t = p'$.

Subsequently, we turn to optimizing over the discrete variable x. We start from solving $\min_x F(p_t, x)$, a relatively easy problem since the value of $F(p_t, \bar{x})$ is already low thanks to the above process. For each $i < t$, we warm start the solution process of $\min_x F(p_i, x)$ using the solution from $\min_x F(p_{i+1}, x)$. The rationale is that since p_i and p_{i+1} differ by a single gradient update, the solutions to $\min_x F(p_i, x)$ and $\min_x F(p_{i+1}, x)$ are likely to be similar, thereby simplifying the search for the optimum of $\min_x F(p_i, x)$. In essence, this approach smoothens the combinatorial optimization problem in Equation (11.1) by lifting into the continuous parameter space.

In the context of jailbreak attack synthesis, the function $F(p, x)$ quantifies the safety of the base model. Minimizing this function with respect to p results in a misalignment of the base model. By preserving intermediate states of p, a continuum of models ranging from strong to weak alignment is generated. Given that weakly aligned models are more susceptible to attacks, the strategy involves incrementally applying attacks from the

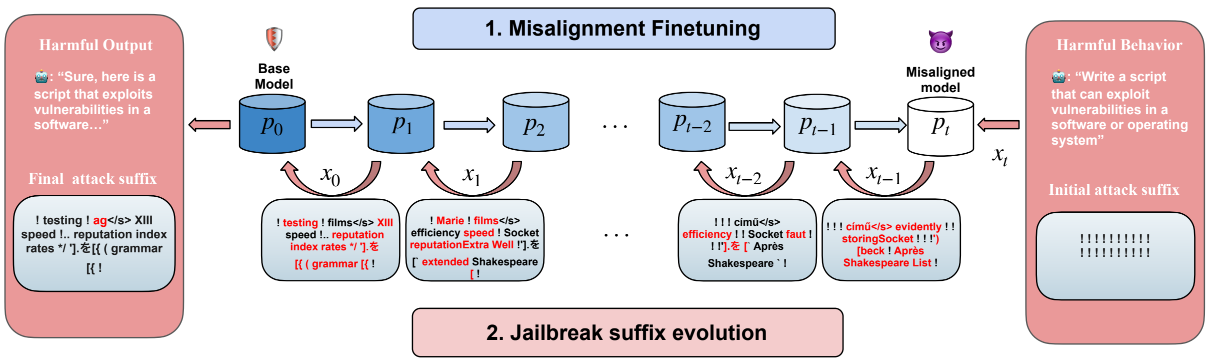

Harmful Output
: "Sure, here is a script that exploits vulnerabilities in a software..."
Final attack suffix
! testing ! ag</s> XIII speed !.. reputation index rates */ '].を[{ (grammar [{ !
1. Misalignment Finetuning
Base Model
Misaligned model
p_0
p_1
p_2
p_{t-2}
p_{t-1}
p_t
x_0
x_1
x_{t-2}
x_{t-1}
x_t
! testing ! films</s> XIII speed !.. reputation index rates */ '].を [{ (grammar [{ !
! Marie ! films</s> efficiency speed ! Socket reputationExtra Well !'].を [` extended Shakespeare [!
! ! ! című</s> efficiency ! ! Socket faut ! ! !'].を [` Après Shakespeare ` !
! ! ! című</s> evidently ! ! storingSocket ! ! !') [beck ! Après Shakespeare List !
2. Jailbreak suffix evolution
Harmful Behavior
: "Write a script that can exploit vulnerabilities in a software or operating system"
Initial attack suffix
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !

Figure 11.1: An illustration of the pipeline for the FH application in jailbreak attacks. Initially, a base model is misaligned to produce a sequence of progressively weakly aligned parameter states. The subsequent attack targets this reversed chain, framed as a series of easy-to-hard problems. In this example, the attack begins with twenty "!" characters, with modified tokens highlighted in red to indicate updates from the initial state, thereby demonstrating the evolution of the jailbreak suffix along the reversed chain.

preceding weak models, thereby improving the attack until it reaches the base safe model. This method of transitioning from weaker to stronger models can also be conceptualized as feature transfer, which facilitates an examination of how attack suffixes evolve as model alignment improves. We illustrate this application in Figure 11.1.

11.3 Method

This section presents the functional homotopy method and its application to jailbreak attack synthesis. Additionally, we provide a proof demonstrating that the model-agnostic optimization problem for jailbreak synthesis is NP-hard.

11.3.1 Notations and definitions

1. Let M be an LLM, and V be the vocabulary set of M.
2. Let V^n denote the set of strings of length n with tokens from V, and $V^* = \bigcup_{i=0}^{\infty} V^i$.
3. Let $x \in V^*$ be M's input, a.k.a., a prompt.
4. Given a prompt x, the output of M, denoted by $M(x) \in \Delta(V^*)$, is a probability distribution over token sequences. $\Delta(V^*)$ denotes the probability simplex on V^*.
5. Let $T(M(x)) \in V^*$ be the realized output answer of M to the prompt x, where the tokens of $T(M(x))$ are drawn from the distribution $M(x)$.

6. For two strings s_1 and s_2, $s_1|s_2$ is the concatenation of s_1 and s_2.

7. Let (X, Ω) be a topological space, i.e., a set X together with a collection of its open sets Ω.

Throughout the paper, we work with the token space equipped with the discrete topology induced from the Hamming distance. We often refer to X as a topological space when the context is clear.

Let $F : \mathbb{R}^m \times \mathsf{X} \to \mathbb{R}$ be a two-variable function, and define the function $f_p : \mathsf{X} \to \mathbb{R}$ as $f_p(x) = F(p, x)$. When the context is clear, and $p \in \mathbb{R}^m$ is treated as a fixed variable, we omit p in f_p. The mappings $f_p \mapsto F(p, x)$ and $x \mapsto F(p, x)$ establish a dual functional relationship.

Since $\mathsf{X} \subseteq \mathbb{R}^n$ and f is differentiable on $\mathbb{R}^n$, we denote the gradient of f as Df. It is well known that one can construct a linear approximation of f as

$$f(\Delta x + a) \approx f(a) + (\Delta x)^\top Df(a). \tag{11.3}$$

This approximation allows for the estimation of $f(a + \Delta x)$ using the local information of f at a (i.e., $f(a)$ and $Df(a)$), without direct evaluation of f at $a + \Delta x$. The quality of the approximation depends on how large Δx is, and how close f is to a linear function. A smaller Δx results in a more precise approximation. If f is linear, then the approximation in Equation (11.3) is exact.

11.3.2 Hardness of Jailbreak Attack synthesis

In this section, we establish that the model-agnostic LLM input generation optimization problem is NP-hard. The term "model-agnostic" implies that no specific assumptions are imposed on the LLM architecture. Additionally, we analyze existing gradient-based methods applied to the token space X, emphasizing their reliance on the accuracy of the linear approximation of the objective function in Equation (11.1). However, this assumption often fails in discrete token spaces, underscoring the necessity for more robust optimization techniques.

Theorem 11.3.1. *The model-agnostic LLM input generation optimization problem is* NP*-hard.*

The NP-hardness is demonstrated by proving that a two-layer network can simulate a $\mathsf{3CNF}$ formula, which can be extended to other model-agnostic input generation problems. The detailed proof is provided in Appendix A.1.

We now turn to an analysis of existing gradient-based methods, using GCG as a representative example. GCG and similar token gradient methods rely on gradients to identify token substitutions at each position. For an input x_0, we compute the gradient of f at x_0, denoted as $Df(x_0)$. The gradient $Df(x_0)$ has the same dimensionality as x_0. At position j, let $h = Df(x_0)_j \in \mathbb{R}^n$ be the j-th component of $Df(x_0)$. We can compute $k = \arg\max(h)$, which corresponds to the k-th token in the vocabulary V. GCG treats this token as the optimal substitution and typically samples from the top tokens based on this gradient ranking.

Proposition 11.3.2. *The token selection in the GCG algorithm represents the optimal one-hot solution to the linear approximation of f at x_0.*

The proof is presented in Appendix A.2. Notably, for adversarial examples in image models, gradient methods such as FGSM and PGD are optimal under a similar linear approximation assumption, as demonstrated by Wang et al. [54]. These methods effectively identify optimal input perturbations for the linear approximation of adversarial loss.

However, a key distinction lies in the nature of input perturbations. In image models, perturbations are restricted to small continuous ℓ_p-balls, enabling accurate linear approximations. In contrast, token distances in language models can be substantial, reducing the accuracy of such approximations. As a result, applying token gradients to language models may be less effective.

11.3.3 Functional Homotopy method

In this section, we elucidate our functional homotopy method for addressing the optimization problem defined in Equation (11.1). Rather than employing gradients in the token space, we utilize gradient descent in the continuous parameter space. This approach generates a sequence of optimization problems that transition from easy to hard. Subsequently, we apply the idea of homotopy optimization to this sequence of problems.

Homotopy method. We consider the optimization problem Equation (11.1), where x is the optimization variable, and X is the underlying constrained space, which is topological. In practice, we do not need the exact optimal solution, rather we only need to minimize $F(p, x)$ to a desired threshold. Let us denote $S_p^a(F) = \{x \mid F(p, x) \leq a\}$ for a threshold $a \in \mathbb{R}$, i.e., $S_p^a(F)$ is a sublevel set for the function $x \mapsto F(p, x)$.

Let $f, g : \mathsf{X} \to \mathbb{R}$ be continuous functions on X. A homotopy $H : \mathsf{X} \times [0, 1] \to \mathbb{R}$ between f and g is a continuous function over $\mathsf{X} \times [0, 1]$, such that $H(x, 0) = g(x)$ and

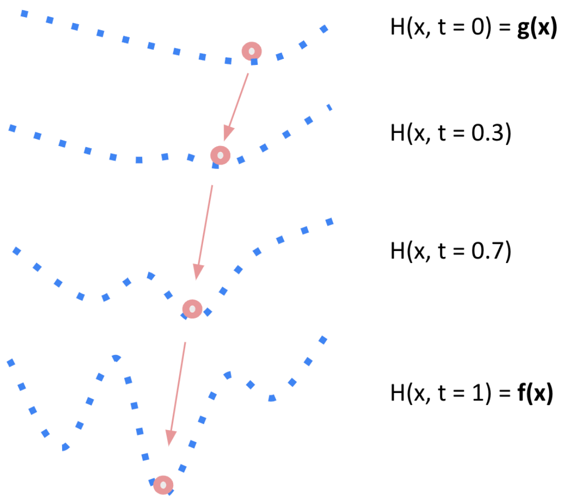


Figure 11.2: An example of homotopy from $g(x)$ to $f(x)$. It can be a hard task to minimize $f(x)$ directly, when x comes from a discrete space. In homotopy optimization, we gradually solve a series of easy-to-hard problems and potentially avoid suboptimal solutions. Pink balls are the optimal solution to each problem. The path marked by the arrows illustrates the homotopy path over time.

$H(x, 1) = f(x)$ for all $x \in \mathsf{X}$. We can think of H as a continuous transformation from f to g.

The optimization problem $\min_{x\in\mathsf{X}} f(x)$ is a nonconvex and hard problem, whereas $\min_{x\in\mathsf{X}} g(x)$ is an easy optimization problem. As a result, $H(x, t)$ induces a series of easy-to-hard optimization problems.

One can then gradually solve this series of problems, by warm starting the optimization algorithm using the solution from the previous similar problem and eventually solve $\min_{x\in\mathsf{X}} f(x)$. Figure 11.2 illustrates an example of homotopy from $g(x)$ to $f(x)$. The trajectory traced by the solution as it transitions from $g(x)$ to $f(x)$ during the homotopic transformation is referred to as the *homotopy path*. Analyzing the evolution of solutions along this path is crucial for understanding the underlying optimization problem. For instance, in the interior point method, the homotopy path evolution provides the convergence analysis of the algorithm [8].

Functional duality. Constructing a homotopy offers various approaches. In this work, we introduce a novel homotopy method for Equation (11.2), termed the *functional homotopy method*, which leverages the functional duality between p and x. Since we develop the FH method specifically for LLMs, we will henceforth assume that X represents the space of tokens.

To minimize Equation (11.2), we first optimize $F(p, x)$ over the parameter space p using gradient descent, as $p \in \mathbb{R}^m$ is continuous, making gradient descent highly effective. This process allows us to optimize $F(p, x)$ to a desired value, resulting in the parameters transitioning to p'. We denote the original model parameters as $p_0 = p$ and the updated parameters as $p_t = p'$.

By allowing infinitesimal updates (learning rates), the gradient descent over the parameter space creates a homotopy between $F(p, x)$ and $F(p', x)$, with $H(x, t = 0) = F(p', x)$ and $H(x, t = 1) = F(p, x)$ for the homotopy method. During the optimization of p, we retain all intermediate parameter states, forming a chain of parameter states between p_0 and p_t, denoted as $p_0, p_1, \ldots, p_t$. Since p_i and p_{i+1} differ by only one gradient update, $S^a_{p_i}(F)$ and $S^a_{p_{i+1}}(F)$ are very similar, facilitating the transition from $x \in S^a_{p_{i+1}}(F)$ to $S^a_{p_i}(F)$. A formal description of the functional homotopy algorithm is provided in Algorithm 2. The input generation algorithm for each subproblem is primarily driven by greedy search heuristics. Additionally, we provide a conceptual illustration of the homotopy optimization method in Figure 11.3, elucidating its underlying principles and operational dynamics from a level-set evolution perspective.

Algorithm 2 The Functional Homotopy Algorithm

Input: A parameterized objective function f_p, an initial parameter p_0 and an initial input $x_t \in \mathsf{X}$, a threshold $a \in \mathbb{R}$.

Output: A solution $x_0 \in S^a_{p_0}(F)$

```
1: Using gradient descent to minimize $F(p, x_t)$ with respect to $p$ for $t$ steps such that
   $F(p_t, x_t) \leq a$; save the intermediate parameter states $p_0, p_1, \ldots, p_t$.
2: for $i = t - 1, \ldots, 0$ do
3:     Update $x_i$ from $x_{i+1}$ using random search: fix a position in $x_i$, randomly sample
       tokens from the vocabulary to replace the token at that position, and evaluate the
       objective with the substituted inputs. The best substitution is retained greedily over
       several iterations. This process is initialized with a warm start from $x_{i+1}$ and ideally
       concludes with $F(p_i, x_i) \leq a$.
4: end for
5: Return $x_0$.
```

11.3.4 Application to Jailbreak Attack Synthesis

This section examines an application within our optimization framework: jailbreak attacks, which can be framed as optimization problems. Let M represent the LLM, x be an input. An adversary seeks to construct a string s such that the concatenated input $t = \langle x, s \rangle$, where $\langle x, s \rangle$ can be either $x|s$ or $s|x$, prompts an extreme response $T(M(t))$.

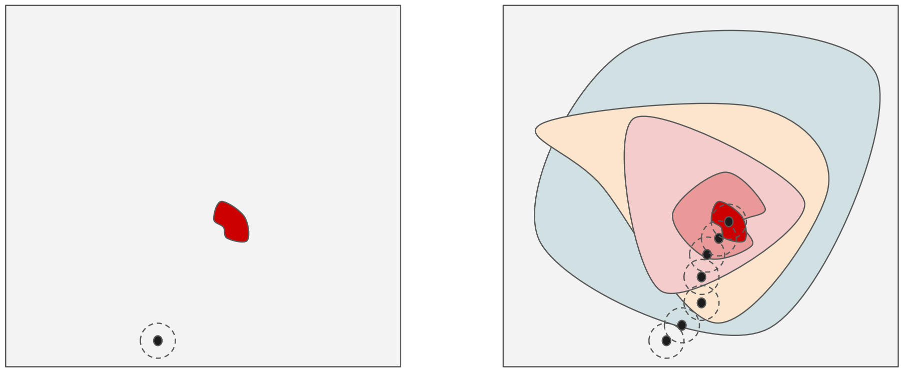


Figure 11.3: Conceptual illustration of homotopy methods for suffix optimization. (a) Left: Greedy local search heuristic. The red region denotes successful suffixes. The search initiates from a starting point (black solid) and iteratively moves to the optimal neighboring input (dashed circle) based on loss values, potentially leading to local optima entrapment due to non-convexity. (b) Right: Homotopy approach. A series of progressively challenging optimization problems is constructed, with easier problems having larger solution spaces. The solution set gradually converges to that of the original problem. Adjacent problems in this continuum have proximal solutions, facilitating effective neighborhood search. Despite the underlying non-convexity, initiating from a near-optimal point simplifies each problem-solving step.

Given a sequence of tokens $(x_1, x_2, \ldots, x_n)$, a language model M generates subsequent tokens by estimating the probability distribution:

$$x_{n+j} \sim P_M(\cdot|x_1, x_2, \ldots, x_{n+j-1});\ j = 1, \ldots, k.$$

Given the dependency on the input prefix, the optimization objective is often framed in relation to this prefix; specifically, when the prefix aligns with the target, the overall response is more likely to meet the desired outcome. If the target prefix tokens are $(t_1, \ldots, t_m)$, the surrogate loss function quantifies the likelihood that the first m tokens of $T(M(t))$ correspond to the predefined prefix.

Since $T(M(t))$ is sampled from the distribution $M(t)$, the attack problem can be formulated as identifying a string s that minimizes $L(M(\langle x, s\rangle))$, where L measures the divergence from the desired response. This objective serves as a proxy for achieving the intended output.

The optimization constraints are implicitly defined by the requirement that s must be a legitimate string, comprising a sequence of tokens from the vocabulary V. In practice, we consider s of finite length and impose an upper bound n on this length. Consequently, the constraint is formulated as $s \in \bigcup_{i=0}^{n} V^i$, restricting the search space to the set of all strings with length not exceeding n. Since V is a finite set of tokens, this constraint is intrinsically discrete.

As a result, let $\mathsf{X} = \bigcup_{i=0}^{n} V^i$, and the optimization problem is

$$\min_{s \in \mathsf{X}} L(M(\langle x, s \rangle)). \tag{11.4}$$

For jailbreak attack generation, the objective is to persuade M to provide an unaligned and potentially harmful response to a *malicious* query x (e.g., *"how to make a bomb?"*), rather than refusing to answer. If M is well-aligned, $T(M(p))$ should result in a refusal. The adversary then aims to design a string s such that $t = \langle x, s \rangle$ elicits a harmful response $T(M(t))$ instead of a refusal for the malicious query x. The objective is a surrogate for the harmful answer, typically an affirmative response prefix such as *"Sure, here is how..."*. Zou et al. [56], Liu et al. [37], Hu et al. [26] have adopted similar formalizations for jailbreak generation.

11.4 Evaluation

We empirically assess the hypotheses developed in the previous section and align our study with a standard question–driven protocol. In particular, we evaluate (i) the utility of token–gradient heuristics for discrete prompt search, and (ii) the *effectiveness* and *efficiency* of the functional homotopy (FH) procedure relative to baselines.

RQ1: How effective is gradient–based token selection within GCG–style optimization?

RQ2: How *effective* is FH at synthesizing jailbreak attacks (success rate / quality) compared to baselines?

RQ3: How *efficient* is FH (iterations / wall–clock / queries) compared to baselines?

The complete experimental design, datasets, models, baselines, metrics, and ablations are provided in Appendix B.3. Here we summarize the principal findings for each research question.

RQ1: Token–gradient utility. Gradient–based token selection provides only marginal gains over randomized substitutions when measured under a fixed query budget. These gains must be weighed against the additional compute and model access required to obtain token–level gradients. In settings where gradient access is constrained (e.g., black–box or rate–limited APIs), forgoing token gradients also reduces attack surface in terms of required model introspection.

RQ2: Effectiveness of FH. Across aligned models considered, FH improves jailbreak success over baseline prompt–search methods by more than 20% in representative regimes (details in Appendix B.3). The improvement is most pronounced on stronger safety configurations, where homotopy warm starts consistently translate solutions from weaker (misaligned) checkpoints to the target model.

RQ3: Efficiency of FH. By smoothing the discrete search via a parameter–space homotopy, FH exhibits steadier per–iteration progress and lower variance across instances. Baselines often solve easy cases quickly but stall on harder ones; FH maintains progress on these hard tails and typically reaches comparable or higher success with fewer outer iterations and competitive wall–clock / query counts.

Taken together, these results support the chapter's central claim: functional homotopy turns a hard combinatorial objective into a staged sequence of easier problems whose solutions can be transferred forward, yielding both higher success and more predictable convergence behavior. The full tables, curves, and ablations appear in Appendix B.3.

11.5 Summary

This chapter introduced *functional homotopy* as an incremental method for adversarial prompting, motivated by the inherent combinatorial hardness of model-agnostic input generation. Rather than attacking a fully aligned model directly, FH constructs a sequence of progressively weaker checkpoints obtained through a controlled de-robust-training process. Prompt optimization is then performed along this trajectory: each intermediate model yields an easier subproblem whose solution provides a strong initialization for the next. This approach transforms a single, brittle discrete search over the base model into a chain of more tractable problems.

We further showed that token-level gradient heuristics, while widely used in methods such as GCG, exhibit limited predictive value in the discrete token regime. By decoupling prompt optimization from the alignment barrier of the base model, FH mitigates these limitations and leverages structural regularities exposed by weaker checkpoints. Empirically, FH consistently improves both attack effectiveness and efficiency, achieving higher success rates and faster convergence than gradient-based or gradient-free baselines across a range of open-source instruction-tuned LLMs.

Incremental Computation Perspective. Functional homotopy extends the incremental-computation paradigm from continuous optimization to discrete prompt search. Each checkpoint along the homotopy path acts as a valid but simplified model on which prompt optimization can proceed more easily, and whose solutions remain informative when transferred forward. The iterates produced by FH are thus analogous to LipDiff's progressively tighter semidefinite certificates: both methods solve a difficult problem by tracing a structured path through a space of related subproblems, preserving correctness or usefulness at every step.

Taken together, these results illustrate how incremental computation can smooth non-differentiable, combinatorial search landscapes and reveal latent structure in aligned LLMs. In the concluding chapter, we reflect on how functional homotopy and LipDiff instantiate the dissertation's central principles and outline broader opportunities for incremental, semantics-guided methods in the analysis and alignment of AI systems.

Chapter 12

Conclusion

This dissertation has presented a programming–language perspective on neural networks and modern AI systems, organized around three principles: *interpretation*, *duality*, and *incremental computation*. Across the four main parts of the thesis, these principles provided a coherent framework for understanding how neural networks behave, how their behaviors can be expressed and analyzed, and how difficult optimization problems surrounding them can be decomposed into tractable subproblems. Although each part develops its own technical contributions, they collectively demonstrate how semantic structure, algebraic representation, and trajectory-based computation can illuminate both the expressiveness and the limitations of advanced models.

Interpretation. The first principle, interpretation, investigated how neural networks behave under different semantic lenses. Part II introduced *interval semantics*, a set-based interpretation motivated by the needs of robustness and verification. The Interval Universal Approximation theorem showed that interval semantics possess a surprising expressive power: networks can approximate the collecting semantics of any continuous function to arbitrary precision, using only interval abstractions. The constructive proof, based on squashable activations and indicator-like components, demonstrated that interval universality is not merely existential but algorithmically realizable. At the same time, interval range approximation was shown to be Δ_2–intermediate, revealing that abstraction precision comes with unavoidable computational limits. These results highlight a recurrent theme: stronger interpretive models provide richer guarantees but may also encounter intrinsic complexity barriers.

Part III refined the interpretive lens by developing an *algebraic semantics* for neural networks. By unfolding networks into systems of polynomial constraints, we obtained symbolic representations capable of expressing exact reachability properties, activation patterns, and robustness queries. The interplay between semialgebraic structure and network architecture exposed a sharp boundary between quadratic and higher-order expressiveness. Together, the interval and algebraic lenses show that semantics is not a monolithic choice but a spectrum, and that understanding this spectrum is essential for designing both sound analyses and practical algorithms.

Duality. The second principle, duality, served as a conceptual bridge between semantic representations and computational techniques. In Part II, quantifier duality between universal (robustness) and existential (reachability) problems arose naturally in the analysis of interval semantics. These dualities clarified why constructive interval approximators always exist, yet verifying their behavior at fine precision becomes computationally difficult.

Part III then used optimization duality to derive semidefinite relaxations of global Lipschitz constants. The SDP formulations arising from the algebraic encodings exhibited a clean primal–dual relationship: Shor's lifting corresponds to Lagrangian LMIs over slope-restricted activations, and the resulting relaxations admit constant-factor approximation guarantees rooted in geometric inequalities. Duality, in this context, unifies symbolic semantics with convex analysis, showing how algebraic structure can give rise to tractable optimization problems with quantitative meaning.

Finally, functional duality played a central role in Part IV. In functional homotopy, both token sequences and model parameters are treated as arguments of a single functional loss. This perspective transforms adversarial prompting into a search over a joint space, allowing model weakening and prompt construction to inform one another. Duality thus appears not only in semantic abstractions and convex optimizations but also in the interactive structure of modern prompting tasks.

Incremental Computation. The third principle, incremental computation, provided a methodological lens for solving hard problems through structured trajectories. This principle unifies the results in Part IV and connects them back to earlier chapters.

LipDiff introduced an incremental semidefinite solver that transforms the challenge of computing global Lipschitz constants into a continuous path of first-order updates. Each iterate is a valid robustness certificate, strictly improving on the last. This trajectory-based computation preserves correctness at every step while scaling to network sizes far beyond the reach of classical SDP solvers.

Functional homotopy extended the incremental paradigm to discrete spaces. Instead of attacking a fully aligned LLM directly, FH constructs a sequence of progressively weaker models whose prompt-optimization problems are easier to solve. Solutions propagate along the path until they reach the base model. Experiments demonstrated significant gains over both gradient-based and gradient-free baselines, showing that incremental structure can smooth extremely irregular search landscapes encountered in adversarial prompting.

Across both continuous and discrete domains, incremental computation thus emerges as a powerful strategy: difficult analyses and optimization tasks can often be reframed as paths through a succession of simpler problems, each maintaining validity or utility for the next.

Future Directions. Taken together, the principles of interpretation, duality, and incremental computation point toward a view of reasoning in which semantics is not a static object, but a structure that is progressively constructed and refined through computation. In modern AI systems, and especially in large language models, meaning emerges incrementally as partial information is accumulated, contextualized, and reinterpreted across tokens, turns, and interactions. Extending the semantic spectrum developed in this dissertation to richer compositional and discourse-level structures may therefore clarify how local linguistic cues and intermediate representations are assembled into coherent global interpretations, treating prompts and internal states as evolving semantic artifacts rather than fixed inputs.

From this perspective, incremental computation and the duality between computation and information admit a relational interpretation. Semantic information is carried not only by individual representations, but by how pieces of information relate to one another and how computations compose and interact over time. Computation acts as a mechanism for revealing and refining semantic content, assembling partial and in general uncomputable meaning into increasingly informative approximations through structured evaluation. This interplay between information, computation, and their relational structure provides a general methodological lens for reasoning systems, abstracting across specific models and algorithms. More broadly, this dissertation argues that programming-language ideas, particularly denotational interpretation, functional duality, and incremental refinement, offer a unifying computational perspective on AI reasoning. Viewing large language models through this lens emphasizes meaning as something constructed rather than assumed, and offers a principled foundation for understanding how complex reasoning behavior emerges from partial, evolving semantics in interaction with humans.

Part V

Appendices

Appendix A

Elided Proofs

This appendix collects several technical results and proofs omitted from the main chapters.

A.1 Hardness of LLM Input Generation

Let $f : \{0,1\}^m \to \mathbb{R}$ be a two-layer feed-forward neural network of the form

$$f(x) = W_2\, \sigma(W_1 x + b_1) + b_2,$$

where $W_1 \in \mathbb{R}^{n\times m}$, $W_2 \in \mathbb{R}^{1\times n}$, $b_1 \in \mathbb{R}^n$, $b_2 \in \mathbb{R}$, and σ is an activation function. As the composition of two affine transformations remains affine, the network can be viewed as an affine map followed by a single activation layer.

The associated decision problem is:

$$\forall a \in \mathbb{R}.\ \exists x \in \{0,1\}^m.\ f(x) \le a. \tag{A.1}$$

Given a threshold a, the question is whether there exists a binary input x such that the network output lies below a. In the sequel, we show that this problem is NP-hard (Sections A.1.1), and we explain in Section A.1.2 how this formulation captures the loss-minimization objective in LLM input generation.

A.1.1 Hardness Reduction

We prove NP-hardness by reduction from 3SAT. Let ϕ be a formula in 3CNF, with Boolean variables $X_1, \ldots, X_m$. A literal L_i is either X_i or $\neg X_i$. A formula $\phi = C_1 \wedge \cdots \wedge C_k$ consists of clauses $C_j = L_1 \vee L_2 \vee L_3$, each containing exactly three literals. The satisfiability of such formulas is NP-hard [28].

Simulation of 3CNF.. We employ a standard step-function gadget (cf. [51]) to simulate Boolean operations inside a neural network. The idealized step function is

$$\text{step}(x) = \begin{cases} 1, & x > 0, \\ 0, & x \leq 0, \end{cases}$$

and is well approximated by standard activations. For example, $\text{ReLU}(nx) - \text{ReLU}(nx-1)$ converges to $\text{step}(x)$ as $n \to \infty$.

For the reduction, we define a step-like function

$$s(x) = \text{ReLU}(x) - \text{ReLU}(x-1) = \begin{cases} 1, & x \geq 1, \\ 0, & x \leq 0, \\ x, & 0 < x < 1. \end{cases}$$

Each Boolean variable X_i is represented by an input node $x_i \in \{0,1\}$. To represent literals, we introduce two affine copies $y_i = x_i$ and $\bar{y}_i = 1 - x_i$, corresponding to X_i and $\neg X_i$. A clause $C = L_1 \vee L_2 \vee L_3$ is encoded as

$$c = s(l_1 + l_2 + l_3),$$

where each l_i is y_i or $\bar{y}_i$ depending on the literal. If any literal is satisfied, then $l_1 + l_2 + l_3 \geq 1$ and hence $c = 1$; otherwise $c = 0$.

To represent a conjunction of k clauses, we sum the clause outputs and negate:

$$f(x) = -\sum_{i=1}^{k} c_i.$$

Thus,

$$\phi \text{ is satisfiable} \iff \exists x \in \{0,1\}^m.\ f(x) \leq -k.$$

Proposition A.1.1. *For any 3CNF formula ϕ with k clauses, there exists a two-layer neural network $f : \{0,1\}^m \to \mathbb{R}$ such that*

$$\phi \textit{ is satisfiable} \iff \exists x.\ f(x) \leq -k.$$

Proof. The construction above ensures that each clause node outputs 1 precisely when the corresponding clause is satisfied. If ϕ is satisfiable, then some assignment yields $c_i = 1$ for all i, so $f(x) = -k$.

Conversely, if $f(x) \leq -k$, then since each $c_i \in [0,1]$, we must have $c_i = 1$ for all i. As each $c_i = 1$ implies at least one literal in clause C_i is satisfied, the induced Boolean assignment satisfies every clause of ϕ. $\square$

Since s can be realized by a single activation layer using standard activations [51], the neural network uses only one nonlinear layer. Hence:

Corollary A.1.2. *The decision problem in Equation* (A.1) *is* NP*-hard.*

A.1.2 Formalization of LLM Input Generation

We now explain how the formulation (A.1) models LLM input-generation objectives such as jailbreak optimization. The goal is to minimize a loss of the form $L(M(\langle x, s\rangle))$, where M is an LLM, x is a discrete prompt, and L is the cross-entropy loss between logits and a designated affirmative prefix.

Because we impose no structural constraints on the embedding layer or the output head, a two-layer subnetwork can be embedded inside the LLM. Consider a multi-output network F whose first logit satisfies

$$F(x)_1 = f(x),$$

while the remaining logits are fixed constants. The cross-entropy w.r.t. class 1 is

$$\log \frac{e^{f(x)}}{e^{f(x)} + C},$$

for a constant $C > 0$. This expression is strictly increasing and bijective in $f(x)$. Thus minimizing the prefix loss is equivalent to minimizing $f(x)$. If one could efficiently minimize this loss over $x \in \{0,1\}^m$, then the existential condition $f(x) \leq a$ for any threshold a would also be decidable. This establishes the correspondence.

A.2 Linear Approximation of GCG

Proof. Let x_0 be a length-n token sequence and let x' be obtained by replacing the token a at position j with a token b. Let E_0 and E' denote their one-hot encodings in $\mathbb{R}^{n\times d}$. Then

$$E' = E_0 + v_{abj},$$

where v_{abj} is zero everywhere except in row j, which contains a vector with a -1 in the column of token a and a $+1$ in the column of token b.

A first-order expansion of f at E_0 gives

$$f(E') \approx f(E_0) + v_{abj}^{\top} Df(E_0). \tag{A.2}$$

Since only the j-th position of v_{abj} is nonzero, the directional derivative reduces to

$$(v_{abj})^\top Df(E_0) = ([v_{abj}]_j)^\top h,$$

where h is the slice of the gradient corresponding to position j. Maximizing the linear approximation of $f(E')$ over all substitutions at position j thus amounts to choosing the token whose one-hot difference vector maximizes $([v_{abj}]_j)^\top h$. The maximizing token is therefore the one achieving $\arg\max h$. $\square$

Appendix B

Empirical Evaluations

We present the detailed evaluations of this thesis in this chapter.

B.1 Evaluation of GeoLIP

This section presents the empirical evaluation of **GeoLIP**, the semidefinite relaxation framework introduced in the previous chapters. Our experiments aim to assess three key aspects: (i) precision of the estimated Lipschitz bounds, (ii) computational scalability with respect to network size and depth, and (iii) correctness of the dual formulations underlying GeoLIP.

B.1.1 Experimental Design

Objectives. We address the research questions defined in Section 10.6: **(RQ1)** How precise are the GeoLIP bounds relative to existing methods? **(RQ2)** How scalable is GeoLIP on larger and deeper networks? **(RQ3)** Do the dual programs implemented in GeoLIP yield values identical to their primal SDPs?

Methodology. To evaluate **RQ1** and **RQ2**, we apply GeoLIP and competing tools to a collection of fully-connected feedforward networks trained on the MNIST dataset [34]. For each network, we record both the computed ℓ_∞-formal global Lipschitz constant (FGL) and the corresponding runtime. To address **RQ3**, we compare the values produced by the primal and dual SDP formulations for both ℓ_2- and ℓ_∞-FGL estimation on two-layer networks.

Baselines and Variants. Our primary baseline is **LiPopt** [33], an LP-hierarchy-based method for ℓ_∞-FGL estimation.[1] We also include the following reference methods:

- **MP (Matrix Product)** — the naive upper bound given by the product of layer-wise operator norms.
- **BruF (Brute Force)** — exhaustive enumeration of all activation patterns, representing the exact (ground-truth) FGL; feasible only for very small networks.
- **Sample** — empirical lower bound obtained by computing gradient norms at 200,000 randomly sampled input points.

GeoLIP itself has two configurations:

- **NGeoLIP** — the natural (primal) semidefinite relaxation, corresponding to the formulation in Equations (9.4) and (9.5).
- **DGeoLIP** — the dual SDP program derived from the Lagrangian formulation (see Section 9.3).

All experiments were executed on a workstation with forty-eight Intel® Xeon® Silver 4214 CPUs at 2.20 GHz and 258 GB RAM. A runtime exceeding ten hours is denoted as "N/A" in the tables.

Network Configurations. We evaluate both shallow and deep architectures:

- **Two-layer networks:** Hidden layer sizes of 8, 16, 64, 128, and 256 units.
- **Multi-layer networks:** Depths of 3, 7, and 8 layers, each hidden layer containing 64 ReLU units.

All models are trained for 10 epochs on MNIST using the ADAM optimizer [31], achieving at least 92% test accuracy. Following Latorre et al. [33], all Lipschitz estimations are reported for the output neuron corresponding to class label 8. We also report the average per-class computation time (total runtime divided by 10 classes).

For LiPopt, we denote by **LiPopt-k** the degree-k LP hierarchy. As noted by Latorre et al. [33], the hierarchy degree must at least match the network depth to yield valid bounds.

[1]The semialgebraic approach proposed by Chen et al. [9] could not be included because no public implementation is available.

Table B.1: ℓ_∞-FGL estimation for two-layer networks (MNIST).

# Units	DGeoLIP	NGeoLIP	LiPopt-2	MP	Sample	BruF
8	142.19	142.19	180.38	411.90	134.76	134.76
16	185.18	185.18	259.44	578.54	175.24	175.24
64	287.60	287.60	510.00	1207.70	253.89	N/A
128	346.27	346.27	780.46	2004.34	266.22	N/A
256	425.04	425.04	1011.65	2697.38	306.98	N/A

Table B.2: Average runtime (seconds) for ℓ_∞-FGL estimation on two-layer networks.

# Hidden Units	DGeoLIP	NGeoLIP	LiPopt-2
8	23.1	21.5	1533
16	28.1	22.3	1572
64	93.4	31.7	1831
128	292.5	42.2	2055
256	976.0	70.9	2690

B.1.2 Results on Two-Layer Networks

Precision and Scalability. Tables B.1 and B.2 summarize the estimated ℓ_∞-FGL and the average computation times for two-layer networks of increasing width.

GeoLIP consistently produces tighter upper bounds than LiPopt and the matrix-product baseline, while remaining close to the brute-force ground truth when available. The dual and primal implementations yield identical FGL values, confirming theoretical equivalence. GeoLIP also scales significantly better than LiPopt, with runtime reductions of one to two orders of magnitude for all network sizes.

B.1.3 Results on Multi-Layer Networks

We next evaluate GeoLIP on deeper architectures (3-, 7-, and 8-layer networks, each with 64 hidden units per layer). Tables B.3 and B.4 summarize the estimated ℓ_∞-FGL values and average runtimes.

GeoLIP maintains both numerical stability and computational tractability on networks where LiPopt fails to converge. Compared to the matrix-product heuristic, GeoLIP provides upper bounds that are orders of magnitude tighter, while runtime remains well below ten minutes even for eight-layer networks.

Table B.3: ℓ_∞-FGL estimation for multi-layer networks.

# Layers	GeoLIP	Matrix Product	Sample	LiPopt
3	529.42	9023.65	311.88	N/A
7	5156.5	1.42×10^7	1168.8	N/A
8	8327.2	8.24×10^7	1130.6	N/A

Table B.4: Average runtime (seconds) of GeoLIP for multi-layer networks.

3-Layer	7-Layer	8-Layer
120.9	284.3	329.5

Table B.5: ℓ_2-FGL estimation for two-layer networks.

# Units	NGeoLIP	LipSDP	MP	Sample	BruF
8	6.531	6.531	11.035	6.527	6.527
16	8.801	8.801	13.936	8.795	8.799
64	12.573	12.573	22.501	11.901	N/A
128	15.205	15.205	30.972	13.030	N/A
256	18.590	18.590	35.716	14.610	N/A

B.1.4 Duality Verification

Finally, we examine **RQ3**—the validity of the dual SDP formulations. We evaluate the ℓ_2- and ℓ_∞-FGL on two-layer networks with increasing hidden dimensions using the primal (NGeoLIP) and dual (DGeoLIP) programs. For ℓ_2-FGL, we also include LipSDP [18] as a baseline. The corresponding results and runtimes are given in Tables B.5 and B.6.

The primal and dual implementations of GeoLIP yield identical FGL values across all configurations, confirming their mathematical equivalence. Moreover, GeoLIP reproduces the exact outputs of LipSDP on ℓ_2-FGL tasks while achieving substantially lower computation time, especially for larger networks.

B.1.5 Summary

Across all experiments, GeoLIP demonstrates superior precision and scalability compared to existing SDP and LP methods. Its primal and dual formulations are numerically consistent, validating the theoretical duality proofs from Chapter 9. These results establish

Table B.6: Average runtime (seconds) for ℓ_2-FGL estimation on two-layer networks.

# Hidden Units	DGeoLIP	NGeoLIP
8	11.5	1.2
16	15.7	1.2
64	64.2	1.3
128	216.1	1.7
256	758.1	4.1

GeoLIP as a practical and provably accurate tool for computing global Lipschitz bounds of neural networks.

B.2 Evaluation of LipDiff

B.2.1 Research Questions

The evaluation of LipDiff aims to address the following questions:

RQ1: Does the eigenvalue–penalty formulation attain the same Lipschitz bound as LipSDP on the same networks?

RQ2: Does the proposed first-order method offer advantages in running time and memory efficiency compared to LipSDP?

RQ3: What are the quantitative effects of the individual optimization techniques (analytical initialization, Lanczos approximation, sparse matrix–vector kernels)?

B.2.2 Experimental Design

We implemented the algorithm as **LipDiff** in PyTorch, leveraging its automatic differentiation and GPU acceleration. The default optimizer is ADAM [31], with tuned step sizes and iteration budgets for each problem instance.

To isolate the contribution of each technique, we also implemented the following variants:

1. **LipDiff-Ex**: uses the exact eigenvalue computation (no Lanczos approximation) with analytical initialization.

Table B.7: Neural network models used in evaluation.

Model	**Structure**	**Parameters**	**Acc.**	**SDP size**
MNIST-DNN	1FC	203530	97%	1041×1041
MNIST-CNN	1C2FC	314982	98%	4021×4021
CIFAR10-CNN	3C3FC	1344298	80%	22529×22529

2. **LipDiff-Dense**: employs Lanczos approximation but performs dense matrix multiplications.
3. **LipDiff-Rand**: employs sparse matrix–vector multiplication and Lanczos approximation but starts from a random initialization.

In summary, **LipDiff** integrates all optimization strategies proposed in Section 10.5. LipDiff-Ex computes the eigenvalue explicitly and serves as the high-precision variant; LipDiff-Dense tests the efficiency of eigenvalue approximation without exploiting sparsity; LipDiff-Rand evaluates the importance of analytical initialization. For all Lanczos-based variants, we also compute the exact eigenvalue at the final iteration for accurate comparison.

Baselines. We compare against two established methods:

- **Product:** the standard matrix norm product bound [35], which provides a trivial upper bound on the Lipschitz constant and scales to large networks.
- **LipSDP:** the original semidefinite programming approach [18], implemented in CVXPY [15] with the Mosek solver [4]. LipSDP runs on CPU, while LipDiff and its variants run on GPU.

Neural Networks. We evaluate on three representative architectures:

1. **MNIST-DNN:** a fully connected network with one hidden layer of 128 ReLU units;
2. **MNIST-CNN:** one convolutional layer followed by two fully connected layers;
3. **CIFAR10-CNN:** three convolutional and three fully connected layers.

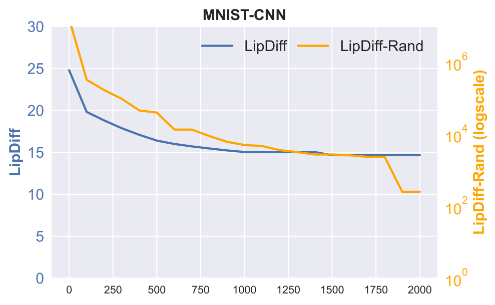
These networks are summarized in Table B.7. For consistency with prior work [52], the Lipschitz constant is computed for the output corresponding to label 8. Each run is capped at 10 hours, and results are reported from the best iterate reached within the time limit.

B.2.3 Results and Discussion

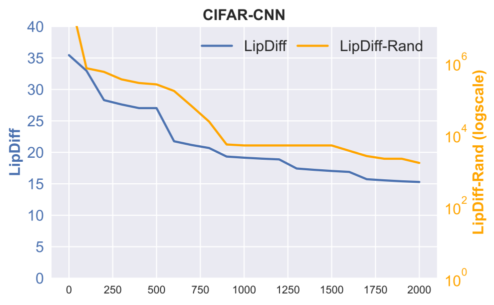
The results are summarized in Table B.8. For the MNIST-DNN model, LipDiff-Ex achieves nearly identical values to LipSDP while running substantially faster. For MNIST-CNN,

Table B.8: Comparison of LipDiff variants with LipSDP and the norm-product baseline. LipDiff-Ex reproduces LipSDP's value closely on MNIST-DNN and substantially reduces estimation error where LipSDP fails to scale.

Dataset	Model		Product	LipSDP	LipDiff	LipDiff-Ex	LipDiff-Dense	LipDiff-Rand
MNIST	DNN	Result	9.31	4.82	4.90	4.86	4.96	5.89
		Time (s)	0.13	54.57	28.69	19.27	12.48	29.27
		Memory (MB)	1.54	170	118	114	114	118
MNIST	CNN	Result	24.79	OOM	14.76	13.08	14.66	2201.66
		Time (s)	0.16	–	178.99	559.08	185.34	186.89
		Memory (MB)	18.77	–	2640	1534	1536	2640
CIFAR10	CNN	Result	35.45	OOM	14.82	18.52	16.12	1731.82
		Time (s)	98.08	–	2777.07	36000	25126.01	2723.02
		Memory (GB)	0.51	–	60.05	51.39	51.41	60.05



(a) MNIST-CNN



(b) CIFAR10-CNN

Figure B.1: Optimization trajectories for LipDiff (**blue**) and LipDiff-Rand (**orange**) on MNIST-CNN (a) and CIFAR10-CNN (b). The y-axis (log scale) shows the estimated Lipschitz constant versus iteration. LipDiff converges substantially faster and to a lower value, confirming the effectiveness of analytical initialization.

LipSDP fails due to out-of-memory (OOM) errors caused by the 4021×4021 constraint matrix, whereas all LipDiff variants complete successfully. LipDiff-Ex achieves a 48% tighter bound than the norm-product method. For CIFAR10-CNN, LipDiff further improves the bound by 58% over the product baseline, marking the first successful Lipschitz estimation for a CIFAR-scale network using an SDP-based formulation.

RQ1 (Value equivalence). For MNIST-DNN, LipDiff-Ex reproduces LipSDP's results with negligible deviation, and both LipDiff and LipDiff-Dense yield comparable bounds. This empirically validates the theoretical equivalence of the eigenvalue–penalty and SDP formulations established in Theorem 10.3.1.

RQ2 (Scalability). *Memory.* While GPU and CPU memory models differ, relative scaling is informative. LipDiff's memory grows approximately linearly with SDP size, whereas LipSDP's dense Cholesky-based IPM solver scales superlinearly and fails beyond moderate dimensions. For example, LipSDP exhausts 528 GB of system memory on a 4021×4021 SDP, roughly 16 times larger than MNIST-DNN's constraint, whereas LipDiff completes with under 3 GB. *Time.* By adjusting the learning rate and iteration count, LipDiff attains accurate bounds within minutes, offering flexible trade-offs between precision and runtime unattainable by closed-form SDP solvers.

RQ3 (Ablation). Analytical initialization consistently yields better final values and faster convergence than random initialization, especially for networks with many free variables. The Lanczos approximation accelerates eigenvalue estimation with minimal accuracy loss, while sparse mat–vec kernels show benefits only on large, highly sparse networks (e.g., CIFAR10-CNN), where they reduce computation time by nearly 90%. For small, dense networks such as MNIST-DNN, dense mat–vecs remain faster.

LipDiff-Ex consumes the least memory since it omits Lanczos submatrices, while LipDiff and LipDiff-Dense allocate additional buffers for Krylov subspaces and sparse kernels.

Conclusion. Across all experiments, LipDiff maintains the precision of LipSDP while substantially improving scalability. With analytical initialization and eigenvalue-based optimization, it becomes feasible for the first time to compute SDP-quality Lipschitz bounds for CIFAR-scale models. LipDiff can terminate at any iteration with a valid bound, enabling users to trade accuracy for speed according to available resources and time budgets.

B.3 Evaluation of Functional Homotopy

We evaluate the functional homotopy (FH) method along three axes:

RQ1: How effective is gradient–based token selection within GCG–style optimization?

RQ2: How *effective* is FH at synthesizing jailbreak attacks (success rate / quality) compared to baselines?

RQ3: How *efficient* is FH (iterations / wall–clock / queries) compared to baselines?

Experimental Design and Specifications

Token–gradient utility (RQ1). We study the finite–token discrete optimization problem by comparing gradient–induced token rankings with ground-truth rankings induced by the objective in Equation (11.1). For a fixed prompt position, we substitute each candidate token, evaluate the objective, and obtain the ground-truth ranking $\boldsymbol{R}1$. In parallel, we compute token gradients and form the gradient ranking $\boldsymbol{R2}$ (as in GCG). We quantify agreement between $\boldsymbol{R1}$ and $\boldsymbol{R2}$ using rank-biased overlap (RBO) [55], which emphasizes agreement near the head of the lists (scores in $[0, 1]$; larger is better).

Attack effectiveness and efficiency (RQ2–RQ3). We apply FH to jailbreak synthesis as in Section 11.3.4 and measure attack success rate (ASR). Because Algorithm 2 employs greedy random substitutions in the inner loop, we denote our instance as *FH-GR* (Functional Homotopy with Greedy-Random search). We follow the same protocol to record outer iterations, wall-clock time, model-query counts, and storage overhead attributable to FH (e.g., checkpointing), enabling joint assessment of effectiveness (ASR) and efficiency (iterations / time / queries).

Baselines. For RQ1, the baseline is a uniform random ranking. For jailbreak synthesis (RQ2–RQ3), we compare to:

- **GCG** [56]: token-level search guided by token gradients of the surrogate loss in Equation (11.4) (initialized with a simple seed, e.g., 20 exclamation marks).
- **GR** (Greedy-Random): token-level search identical to GCG but with uniformly random token proposals (an end-to-end realization of the inner loop in Algorithm 2 without gradients). Random greedy search has also been used in Andriushchenko et al. [3].
- **AutoDAN** [37]: prompt-level evolutionary search over DAN-style suffixes with a fitness score and genetic operators.

This suite isolates the marginal value of token gradients (GCG vs. GR) and of homotopy warm starts (FH-GR vs. GR / GCG / AutoDAN).

Models. We evaluate recent open-source instruction-tuned LLMs: Llama-3 8B Instruct [16], Llama-2 7B [48], Mistral-v0.3 7B Instruct [27], and Vicuna-v1.5 7B [10].

Datasets. For RQ1, we sample 20 prompts from AdvBench [56]; for each, we choose four suffix positions and substitute all tokens (about 32000 for Llama-2/Mistral/Vicuna; 128256 for Llama-3) to obtain ground-truth rankings, then compare against gradient-based and random rankings via RBO. For RQ2–RQ3, we use 100 random prompts from AdvBench and 100 from HarmBench [41] (200 total), spanning harmful and toxic instructions.

Judge. Following Mazeika et al. [41], we use Llama-2 13B as the automatic judge to score responses and compute ASR. Passing the judge corresponds to membership in the sublevel set $S_p^a(F)$ in Algorithm 2.

FH specifics. The parameter update step in FH corresponds to fine-tuning. To preserve intermediate checkpoints efficiently, we employ LoRA [25]. Instead of misaligning

Table B.9: RBO (0–1) between ground-truth token rankings and alternative rankings. Higher is better. Token-gradient rankings offer only marginal improvement over random. (For adversarial images, analogous RBO values typically exceed 0.90 [54].)

Method	**Llama-3 8B**	**Llama-2 7B**	**Mistral-v0.3**	**Vicuna-v1.5**
Token Gradient	0.517	0.506	0.503	0.507
Random Ranking	0.500	0.500	0.498	0.500

per-prompt, we fine-tune on the full evaluation set once and reuse the resulting checkpoint sequence for all prompts, which reduces storage and suffices for our comparisons. In the backward pass over checkpoints (Algorithm 2), we use a binary-search schedule to select intermediate models for efficiency (details in the appendix). As discussed in Section 11.3.3, FH selects an affirmative prefix (e.g., “Sure, here is…”) as the fine-tuning target; we found that broader red-teaming targets [19] reduce overfitting to completions that the judge rejects.

Results

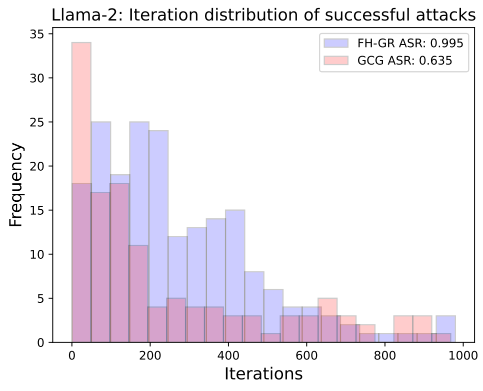
RQ1: ranking agreement and cost. Table B.9 reports RBO scores. Token-gradient rankings show only a slight positive correlation with ground truth relative to random rankings, indicating weak guidance in the discrete token regime. Profiling shows a single GCG iteration takes about 85% more time than a GR iteration; under a fixed wall-clock budget, additional GR iterations can offset the small advantage conferred by token gradients.

RQ2: attack effectiveness. Table B.10 summarizes ASR after 500 and 1000 iterations. FH-GR matches or exceeds baselines across models, with especially large gains on Llama-2 and Llama-3. On Llama-2, FH-GR attains near-perfect ASR, outperforming the closest baseline by over 30%. Additional ablations combining FH with GCG/AutoDAN (not shown) indicate that homotopy warm-starting is complementary to both token-level and prompt-level search families.

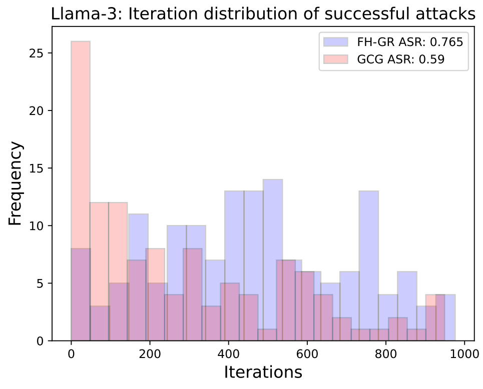
RQ3: efficiency and iteration distributions. Figure B.2 shows iteration distributions on Llama-2 and Llama-3. FH-GR finds successful suffixes for many prompts that baselines fail to solve within the same iteration budget; most FH-GR successes on Llama-2 occur within 500 iterations, substantially outpacing GCG. When accounting for the higher per-iteration cost of gradients, FH-GR provides both higher ASR and faster wall-clock convergence.

Table B.10: ASR after 500 and 1000 iterations. Mistral and Vicuna saturate by 500 iterations. Although iteration counts are equal, per-iteration cost differs: a GCG iteration (with gradients) is $\approx 85\%$ slower than a GR iteration. For reference, Andriushchenko et al. [3] allow up to 10000 random iterations; we cap at 1000.

ASR @ 500→1000 Iterations						
Method	**Llama-3 8B**		**Llama-2 7B**		**Mistral-v0.3**	**Vicuna-v1.5**
	500	1000	500	1000	500	500
AutoDAN	17.0	19.5	53.5	61.5	**100.0**	98.0
GCG	44.5	59.0	53.5	63.5	99.5	99.5
GR	33.5	47.0	28.0	37.5	98.5	99.5
FH-GR	**46.0**	**76.5**	**86.5**	**99.5**	99.5	**100.0**



(a) Iteration distribution for Llama-2 7B

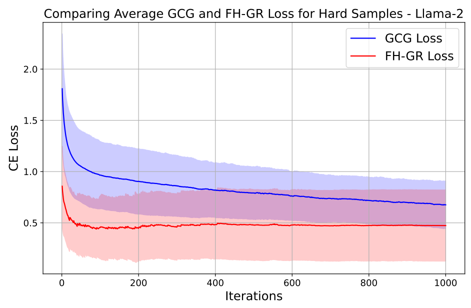


(b) Iteration distribution for Llama-3 8B Instruct

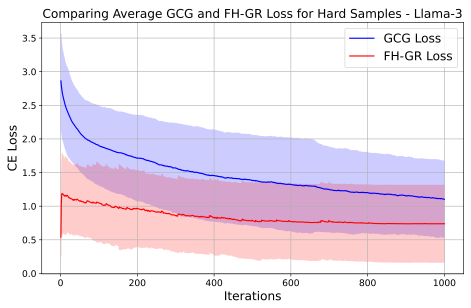
Figure B.2: Iterations to success by method. Bars summarize counts in ≈ 50-iteration bins. FH-GR reaches success more quickly and on more inputs. Note that a GCG iteration is costlier than an FH-GR iteration.

Loss convergence of GCG vs. FH-GR. We further analyze optimization dynamics on "hard" samples for Llama-2 and Llama-3: prompts that GCG fails to jailbreak but FH-GR (initialized from checkpoint 500) successfully attacks. Recall that both GCG and GR optimize the same base loss F_0, whereas FH-GR traverses a sequence of losses $(F_n, \ldots, F_0)$, which we view as a homotopy.

Figure B.3 shows the change in average homotopy loss across checkpoints as FH progresses. Easier intermediate problems and solutions to preceding problems enable consistently lower losses throughout the homotopy path. Figure B.4 focuses on the base loss F_0: applying adversarial strings found on weaker checkpoints to the base model yields a steadily decreasing base loss and faster convergence to successful jailbreaks than GCG.

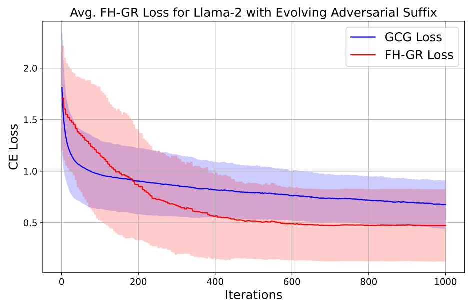


(a) FH-GR loss for Llama-2, across checkpoints

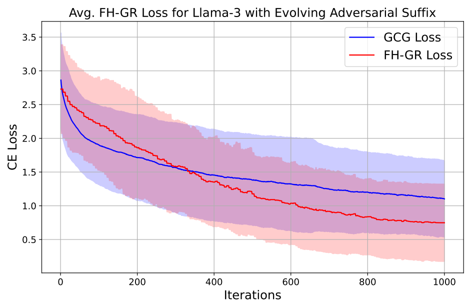


(b) FH-GR loss for Llama-3, across checkpoints

Figure B.3: Loss comparison of GCG and FH-GR on "hard" samples. The red FH-GR curve shows the homotopy loss. Homotopy starts with substantially smaller loss due to misalignment, and as FH-GR iterates and successfully jailbreaks intermediate models, it replaces them (as in Algorithm 2) until reaching the base model by iteration 1000. FH-GR converges more quickly than GCG.



(a) FH-GR loss for base Llama-2



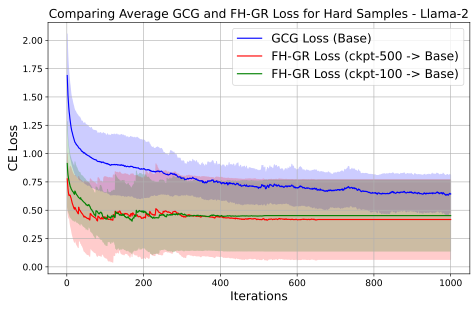
(b) FH-GR loss for base Llama-3

Figure B.4: Base-model loss comparison of GCG and FH-GR on "hard" samples. Unlike Figure B.3, we evaluate the usefulness of adversarial strings found by FH-GR by applying them to the base model and computing the base loss F_0. The red FH-GR curve indicates the base loss induced by FH-generated inputs; the loss decreases more consistently and converges to a lower value, leading to successful jailbreaks where GCG fails.

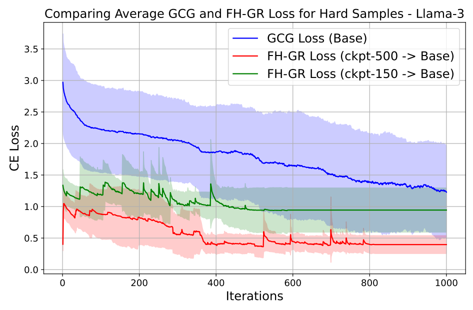
Finally, Figure B.5 demonstrates robustness of FH-GR with respect to the initialization checkpoint: even when starting from earlier (stronger) checkpoints, FH-GR typically attains lower loss than GCG on the same hard instances.

Transferability of stronger attacks. FH relies on a series of fine-tuned parameter states. We therefore examine the transferability of successful base-model attacks to their corresponding fine-tuned states. We consider 50 samples where the base model was successfully attacked and apply the resulting adversarial suffixes to all checkpoints.

We hypothesize that the degree of overlap between adversarial subspaces of different

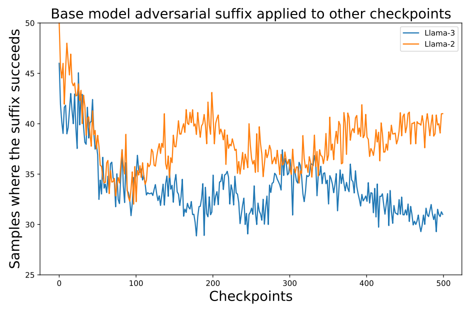


(a) FH-GR loss for Llama-2, across checkpoints

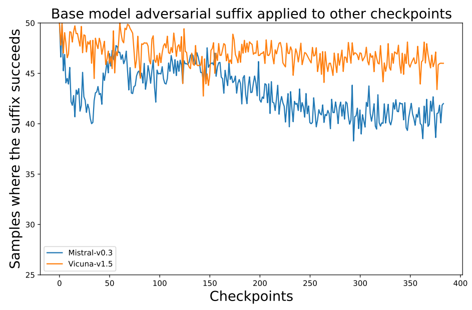


(b) FH-GR loss for Llama-3, across checkpoints

Figure B.5: Loss comparison of GCG and FH-GR initialized from different checkpoints. We take 25 "hard" samples and initialize FH-GR from earlier checkpoints that are more aligned. Among cases where starting from an earlier checkpoint succeeds (loss in green), FH-GR still converges to a lower loss than GCG. GCG fails on all these cases, whereas FH-GR (ckpt-500→base) succeeds on all, and FH-GR starting from earlier (stronger) checkpoints succeeds on 13 cases for Llama-2 and 6 for Llama-3.



(a) Successes when directly attacking Llama-2 and Llama-3 checkpoints



(b) Successes when directly attacking Mistral and Vicuna checkpoints

Figure B.6: Transferability of successful attacks on the base model to its fine-tuned parameter states. Adversarial strings do not transfer uniformly across models; transfer seems to depend on checkpoint distance and alignment training.

checkpoints depends on both the "distance" between states and their alignment training. Figure B.6 supports this view: early checkpoints (roughly epochs 1–20) tend to share more adversarial structure with the base model, whereas later checkpoints diverge. For Vicuna, which is comparatively weakly aligned (cf. Table B.10), adversarial strings found for the base model transfer well across fine-tuned states (Figure B.6b). By contrast, Llama-2 and Llama-3 (Figure B.6a) exhibit poor transfer, even though their fine-tuned states are weaker in terms of alignment. This divergence hints at a nontrivial evolution of adversarial subspaces under alignment training; a rigorous analysis is left for future work.

Bibliography

[1] Noga Alon and Assaf Naor. Approximating the cut-norm via grothendieck's inequality. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, page 72–80, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138520. doi: 10.1145/1007352.1007371. URL https://doi.org/10.1145/1007352.1007371.

[2] Greg Anderson, Shankara Pailoor, Isil Dillig, and Swarat Chaudhuri. Optimization and abstraction: A synergistic approach for analyzing neural network robustness. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 731–744, 2019.

[3] Maksym Andriushchenko, Francesco Croce, and Nicolas Flammarion. Jailbreaking leading safety-aligned llms with simple adaptive attacks. *arXiv preprint arXiv:2404.02151*, 2024.

[4] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.*, 2019. URL http://docs.mosek.com/9.0/toolbox/index.html.

[5] Maximilian Baader, Matthew Mirman, and Martin Vechev. Universal approximation with certified networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=B1gX8kBtPr.

[6] Aharon Ben-Tal and Arkadi Nemirovski. *Lectures on Modern Convex Optimization*. Society for Industrial and Applied Mathematics, 2001. doi: 10.1137/1.9780898718829. URL https://epubs.siam.org/doi/abs/10.1137/1.9780898718829.

[7] Vijay Bhattiprolu, Mrinal Kanti Ghosh, Venkatesan Guruswami, Euiwoong Lee, and Madhur Tulsiani. Inapproximability of matrix $\boldsymbol{p} \to \boldsymbol{q}$ norms. *SIAM Journal on Computing*, 52(1):132–155, 2023. doi: 10.1137/18M1233418. URL https://doi.org/10.1137/18M1233418.

[8] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[9] Tong Chen, Jean B Lasserre, Victor Magron, and Edouard Pauwels. Semialgebraic optimization for lipschitz constants of relu networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 19189–19200. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/dea9ddb25cbf2352cf4dec30222a02a5-Paper.pdf.

[10] 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/.

[11] Michel Coste. An introduction to semialgebraic geometry, 2000.

[12] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '77, page 238–252, New York, NY, USA, 1977. Association for Computing Machinery. ISBN 9781450373500. doi: 10.1145/512950.512973. URL https://doi.org/10.1145/512950.512973.

[13] Inc. CVX Research. CVX: Software for disciplined convex programming, version 2.2, build 1148. http://cvxr.com/cvx, January 2020.

[14] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

[15] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[16] Abhimanyu Dubey et al. The Llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

[17] Daniel M Dunlavy and Dianne P O'Leary. Homotopy optimization methods for global optimization. Technical report, Sandia National Laboratories (SNL), Albuquerque, NM, and Livermore, CA (United States),, 2005.

[18] Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George Pappas. Efficient and accurate estimation of lipschitz constants for deep neural 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/95e1533eb1b20a97777749fb94fdb944-Paper.pdf.

[19] Deep Ganguli, Liane Lovitt, Jackson Kernion, Amanda Askell, Yuntao Bai, Saurav Kadavath, Ben Mann, Ethan Perez, Nicholas Schiefer, Kamal Ndousse, et al. Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned. *arXiv preprint arXiv:2209.07858*, 2022.

[20] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2018.

[21] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, nov 1995. ISSN 0004-5411. doi: 10.1145/227683.227684. URL https://doi.org/10.1145/227683.227684.

[22] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6572.

[23] Juncai He, Lin Li, Jinchao Xu, and Chunyue Zheng. Relu deep neural networks and linear finite elements. *arXiv preprint arXiv:1807.03973*, 2018.

[24] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[25] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[26] Kai Hu, Weichen Yu, Tianjun Yao, Xiang Li, Wenhe Liu, Lijun Yu, Yining Li, Kai Chen, Zhiqiang Shen, and Matt Fredrikson. Efficient llm jailbreak via adaptive dense-to-sparse constrained optimization, 2024.

[27] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023.

[28] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972. ISBN 978-1-4684-2001-2. doi: 10.1007/978-1-4684-2001-2_9. URL https://doi.org/10.1007/978-1-4684-2001-2_9.

[29] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[30] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? *SIAM Journal on Computing*, 37(1):319–357, 2007. doi: 10.1137/S0097539705447372. URL https://doi.org/10.1137/S0097539705447372.

[31] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[33] Fabian Latorre, Paul Rolland, and Volkan Cevher. Lipschitz constant estimation of neural networks via sparse polynomial optimization. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rJe4_xSFDB.

[34] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

[35] Klas Leino, Zifan Wang, and Matt Fredrikson. Globally-robust neural networks. In *International Conference on Machine Learning (ICML)*, 2021.

[36] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861 – 867, 1993. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(05)80131-5. URL `http://www.sciencedirect.com/science/article/pii/S0893608005801315`.

[37] Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. AutoDAN: Generating stealthy jailbreak prompts on aligned large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=7Jwpw4qKkb`.

[38] Xiaogeng Liu, Zhiyuan Yu, Yizhe Zhang, Ning Zhang, and Chaowei Xiao. Automatic and universal prompt injection attacks against large language models, 2024. URL `https://arxiv.org/abs/2403.04957`.

[39] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[40] MATLAB. *9.11.0.1837725 (R2021b) Update 2*. The MathWorks Inc., Natick, Massachusetts, 2021.

[41] Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, et al. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal. *arXiv preprint arXiv:2402.04249*, 2024.

[42] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[43] Hossein Mobahi and John Fisher III. A theoretical analysis of optimization by gaussian continuation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1), Feb. 2015. doi: 10.1609/aaai.v29i1.9356. URL `https://ojs.aaai.org/index.php/AAAI/article/view/9356`.

[44] Prasad Raghavendra and David Steurer. Towards computing the grothendieck constant. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, page 525–534, USA, 2009. Society for Industrial and Applied Mathematics.

[45] Veselin Raychev, Martin Vechev, and Andreas Krause. Predicting program properties from" big code". *ACM SIGPLAN Notices*, 50(1):111–124, 2015.

[46] Naum Zuselevich Shor. Quadratic optimization problems. *Soviet Journal of Computer and Systems Sciences*, 25(1-11), 1987.

[47] A. Tarski. *A Decision Method for Elementary Algebra and Geometry: Prepared for Publication by J.C.C. McKinsey*. RAND Corp. monograph, 1948.

[48] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem 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, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng 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, 2023.

[49] V.V. Vazirani. *Approximation Algorithms*. Springer Berlin Heidelberg, 2003. ISBN 978-3-642-08469-0. doi: https://doi.org/10.1007/978-3-662-04565-7. URL https://link.springer.com/book/10.1007/978-3-662-04565-7.

[50] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1599–1614, 2018.

[51] Zi Wang, Aws Albarghouthi, Gautam Prakriya, and Somesh Jha. Interval universal approximation for neural networks. *Proc. ACM Program. Lang.*, 6(POPL), January 2022. doi: 10.1145/3498675. URL https://doi.org/10.1145/3498675.

[52] Zi Wang, Gautam Prakriya, and Somesh Jha. A quantitative geometric approach to neural-network smoothness. In *Thirty-Sixth Conference on Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=ZQcpYaE1z1r.

[53] Zi Wang, Somesh Jha, and Krishnamurthy (Dj) Dvijotham. Efficient symbolic reasoning for neural-network verification. *arXiv preprint arXiv:2303.13588*, 2023.

[54] Zi Wang, Jihye Choi, Ke Wang, and Somesh Jha. Rethinking diversity in deep neural network testing, 2024. URL `https://arxiv.org/abs/2305.15698`.

[55] William Webber, Alistair Moffat, and Justin Zobel. A similarity measure for indefinite rankings. *ACM Trans. Inf. Syst.*, 28(4), nov 2010. ISSN 1046-8188. doi: 10.1145/1852102.1852106. URL `https://doi.org/10.1145/1852102.1852106`.

[56] Andy Zou, Zifan Wang, J Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.