REINFORCEMENT LEARNING FOR ONLINE PROCESS MONITORING WITH DYNAMIC PARTIAL OBSERVATIONS

By

Haoqian Li

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Industrial and Systems Engineering)

at the

UNIVERSITY OF WISCONSIN-MADISON

2024

Date of final oral examination: 09/09/2024

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

Dr. Kaibo Liu. Faculty, Professor, Industrial and Systems Engineering

Dr. Shiyu Zhou. Faculty, Professor, Industrial and Systems Engineering

Dr. Qiaomin Xie. Faculty, Assistant Professor, Industrial and Systems Engineering

Dr. Yudong Chen. Faculty, Associate Professor, Computer Sciences

© Copyright by Haoqian Li 2024

All Rights Reserve

Acknowledgement

I am profoundly grateful to my advisor, Dr. Kaibo Liu, whose expertise, patience, and dedication have profoundly shaped my academic journey. His relentless pursuit of excellence and his strategic guidance have not only challenged me to push the boundaries of my research but also prepared me for a future of innovative and critical thinking in my field. As a mentor, he taught me invaluable life lessons and provided significant support during critical events in my life. More importantly than the research, he always focused on my mental health, which is invaluable. His kindness, favoritism, and steadfast presence were more than I could have ever asked for.

I extend heartfelt thanks to my committee members, Dr. Shiyu Zhou, Dr. Qiaomin Xie, and Dr. Yudong Chen, whose rigorous evaluation and insightful comments have greatly enhanced the quality of my work. Their expertise in their respective fields has been crucial in broadening my own understanding and approach to complex problems.

A special acknowledgment goes to my family, especially my parents, Luping Li and Yanjun Su. Their endless love, encouragement, and sacrifices have been the cornerstone of my life and academic pursuits. Their belief in my potential and their unwavering support have been sources of strength and motivation during the most challenging times.

To my labmates, Ziqian Zheng and Jiahui Zhang, I owe a debt of gratitude for their friendship, collaboration, and shared enthusiasm for discovery. The environment we nurtured together has been one of mutual growth and persistent inquiry, making our lab a cornerstone of my daily inspiration. I am incredibly fortunate to have their support, as well as the support from my friend Jinwen Sun, whose companionship and understanding have enriched this journey.

Lastly, I cannot forget my cat, Lisa, whose presence has brought comfort and a sense of home to many long days and nights. Her companionship has been a gentle reminder of life beyond academia, and her antics have provided much-needed breaks and laughter.

I would like to extend my gratitude to the U.S. Army Engineer Research and Development Center and the Department of Energy for their financial support of this research under Grant W912HZ20C-0031 and Award DE-NE0009404, respectively. Their contributions were invaluable to the completion of this work.

Table of Contents

List of Tables

List of Figures

Abstract

With the rapid advancement of Internet of Things (IoT) technology and sensing infrastructure, a wide range of systems now generate tremendous amounts of measurements from heterogeneous data streams. These data streams are frequently acquired and utilized for online process monitoring and quick anomaly detection. However, due to practical resource constraints such as limited bandwidth, budget, or processing capability, the full observability of these data streams in real time is often restricted, posing significant challenges in many online monitoring applications. The key research question is how to dynamically select the most informative data streams given the resource constraints to detect anomaly as soon as possible.

This thesis focuses on advancing process monitoring by creating systematic, automated methods for identifying dynamic observations and facilitating rapid anomaly detection in IoT systems. The integration of machine learning, particularly reinforcement learning, with statistical process control (SPC) techniques constitutes the foundation of this approach. Such integration allows for effective system status monitoring in resource-constrained settings, improving both detection speed and efficiency.

The first chapter introduces the background and challenges in process monitoring with resource constraints and establishes the major research objective of the thesis. Chapter 2 diverges from conventional heuristic approaches and proposes a novel reinforcement learning-based algorithm for online monitoring and rapid detection of mean shifts in diverse data streams within dynamic partial observations. This method innovatively combines Q-learning for sampling decisions with a nonparametric SPC-based approach for alarm triggering. However, its effectiveness is limited to monitor a maximum of 10 data streams, and scalability issues arise due to the requirement of discrete state and action spaces. To address these limitations, Chapter 3 introduces deep

reinforcement learning to enhance scalability in monitoring high-dimensional data. To the best of our knowledge, this is the first work that incorporates deep reinforcement learning into the SPC approach to tackle the challenge of resource constraints in monitoring high-dimensional data. Specifically, we construct a nonparametric monitoring statistic for each data stream and develop a deep reinforcement learning framework to automatically identify the most informative data streams for observation at each time epoch. The framework's state space, action space, and rewards are meticulously designed, and a Doubling Dueling Q-network is trained for optimal performance. However, the discrete action space still limits the scalability of the algorithm. Hence, in Chapter 4, we proposed Statistical Advantage Actor Critic algorithm (SA2C) to leverage the continuous action space to increase the scalability. In addition, SA2C is capable of dynamically determining not only the optimal number of data streams to observe but also identifying which specific data streams require surveillance. This enhancement marks a significant boost in adaptability and efficiency in monitoring strategies, reducing overall costs while ensuring prompt and effective anomaly detection. Chapter 5 then summarizes the current thesis and discusses the future work on high-dimensional process monitoring with greater applicability.

In summary, this thesis makes significant contributions to online process monitoring for better quality control, cost reduction, and performance improvement. The developed methods, with their generic framework, can be adapted to diverse sectors like manufacturing, healthcare, energy, and cybersecurity, leading to efficient resource allocation, data acquisition, and substantial cost savings.

Chapter 1 Introduction

1.1 Motivation

In the landscape of modern industry, which encompasses smart manufacturing systems and various other sectors, there is a pressing need to adapt to changing demands and conditions, not only within production facilities but also across supply networks and in meeting evolving customer needs. This adaptability has become increasingly critical with the rapid development of technologies such as sensor systems, communication networks, and enhanced computing power. These advancements have led to an explosion of data across industries, necessitating significant efforts in data acquisition, transmission, and collection. However, this surge in data availability, while offering significant opportunities for performance improvement, also introduces substantial challenges. Among these challenges is the need to effectively manage and analyze partial data observations due to resource limitations and constraints. In real-world applications, due to practical limitations on power usage, transmission bandwidth, and data acquisition cost, practitioners can only obtain a subset of observations at each acquisition time. This resource constraint poses a challenge in adaptively determining the most informative data streams for acquisition and monitoring, with the aim of quickly detecting anomalies in real time. Therefore, there is an urgent need to develop advanced methodologies that will effectively monitor high-dimensional data streams, or extract critical information in general, when only partial observations are available at each acquisition time due to resource constraints. Hence, the objective of this thesis is to advance process monitoring by creating systematic, automated methods for identifying dynamic observations and facilitating rapid anomaly detection in IoT systems under resource constraints.

1.2 Outline of the Thesis

The remainder of the thesis is organized as follows. Chapter 2, diverges from conventional heuristic approaches, proposes a new algorithm based on Q-learning to online monitor and quickly detect mean shifts occurring to heterogeneous data streams in the context of limited resources, where only a subset of observations is available at each acquisition time. In particular, we first use a quantile based nonparametric CUSUM procedure to construct local statistics for each partially observed data stream. Q-learning is then applied to the state space based on discretized local statistics, resulting in the optimal policy for determining the most informative subset of data streams to observe at each acquisition time. Lastly, we follow the optimal policy learned and employ the sum of top-l local statistics as a global monitoring statistic to detect a wide range of possible mean shifts. Both simulations and a case study are thoroughly conducted to evaluate the performance and demonstrate the superiority of the proposed method.

Unfortunately, the proposed method has great scalability issues. Therefore, in Chapter 3, we propose a DRL-based adaptive sampling and monitoring framework. Specifically, SDQ first transforms raw observations of each data stream into a nonparametric local statistic that indicates the likelihood of anomaly occurrences associated with each data stream. The nonparametric statistics and the unattended time form a state representation in the DRL framework. Then, a double-dueling extension of a carefully designed deep Q-network is established. This extension enhances the convergence speed of the algorithm and enables the learning of an optimal dynamic sampling policy. Finally, following the optimal policy suggested by the deep Q-network, we construct the global statistics using the sum of top-l local statistics to indicate whether the system is out of control. The introduction of deep learning and a more complex representation of the state

space which directly includes how long a data stream has not been observed, enables a smarter monitoring strategy with greater scalability.

To further enhance the scalability and the applicability of the proposed method, we propose Statistical Advantage Actor Critic (SA2C) method in Chapter 4. Specifically, the SA2C algorithm initially converts the raw observations from each data stream into a nonparametric local statistic, reflecting the likelihood of anomaly occurrences. These nonparametric statistics, alongside the duration of unattended time, constitute a state representation that informs the input to the deep neural network underpinning the Advantage Actor-Critic (A2C) algorithm. The neural network's output specifies the optimal number of data streams, m_t, to observe, as well as the probabilities p_t that characterize the likelihoods of observing each data stream at each time step. Subsequently, m_t data streams are sampled according to this probability distribution, facilitating the effective identification of the most critical data streams for observation. Ultimately, leveraging the optimal policy generated by A2C, we aggregate the top-l local statistics to construct a global statistic, thereby assessing if the system is out of control.

Finally, Chapter 5 summarizes the thesis and discusses the future work on high dimensional process monitoring with partial observations.

Chapter 2 Online Monitoring of Heterogeneous Partially Observable Data Streams based on Q-learning

2.1 Introduction

The rapid development of modern technology has enabled the monitoring of numerous data streams simultaneously in quality control. However, practical limitations on power usage, transmission bandwidth, and data acquisition cost hinder practitioners from fully observing all data streams at once, resulting in a subset of observations being available. For example, in manufacturing industries, only a limited number of data streams can often be observed and transmitted in real time due to limited bandwidth and processing capability for quick change detection [1]. Similarly, in energy system monitoring, practitioners need to generate a dynamic policy to turn on or off sensors to monitor the health status of power plants due to battery constraint [2]. In military applications, only a limited number of unmanned aerial vehicles are often available to conduct surveillance tasks [3]. As a result, in these applications, there is a practical issue of resource limitation, which results in only a subset of observations available at each acquisition time. This resource constraint poses a challenge in adaptively determining the most informative data streams for acquisition and monitoring, with the aim of detecting anomalies in real time.

To address the resource constraint, various adaptive sampling strategies have been proposed for deciding which partial data streams to observe. These methods commonly require the data stream to satisfy certain distribution assumptions, such as normally distributed [4]–[6], categorically distributed [7], [8], or exchangeable assumptions [9]. Nonparametric schemes have also been

developed [10]–[12], but they do not account for resource constraints and assume the availability of all data streams of interest simultaneously.

Process monitoring can be seen as a sequential decision process, involving the detection of anomalies and the selection of data streams to observe at each acquisition time (also known as "time epoch" in the context of reinforcement learning (RL)). This process exhibits a Markov decision process (MDP) structure. Specifically, at each time epoch, given the current partial observation of data streams due to resource constraints, we need to decide whether there is an anomaly in the system, and if not, which data streams to observe for the next time epoch. However, due to the randomness in the data streams, the transition probability is challenging to be fully described. In this case, conventional methods focusing on directly solving MDPs are not effective. This unique challenge then motivates us to employ the RL as a powerful tool to resolve the MDP problem. RL is particularly suitable for cases with unknown transition probabilities and complex underlying models. In particular, the agent learns through trial and error, systematically exploring the environment and updating strategies based on observed outcomes. This trial-and-error learning process enables RL to adapt to unknown transition probabilities without relying on prior knowledge of the environment's dynamics. RL also considers long-term rewards and handles the exploration-exploitation trade-off. In our problem context, this means to effectively balance between exploiting suspicious data streams and exploring potential mean shifts from every possible data stream. Further, RL considers the long-term discounted reward while existing adaptive sampling approaches [4], [9], [13], [14] are rather heuristic, i.e., only maximizing the short-term benefits when updating the sampling strategy. While RL is promising to be used for online monitoring with partial observations, it faces several unique challenges here, including (1) how to define a reasonable state space and action space that incorporate the information from both

observed and unobserved data streams, as well as satisfy the Markov property which is the key underlying assumption of MDP; (2) how to efficiently raise an alarm and update the sampling strategy based on RL; (3) how to ensure the efficiency of RL as the potential state space and action space can be quite large; and (4) how to smartly define the reward function given the available training dataset to minimize the detection delay. To the best of our knowledge, there is still no effective approach to seamlessly integrate RL with process monitoring for quick anomaly detection of heterogeneous data streams when only partial observations are available at each time epoch. In the paper, we aim to fill this research gap. In particular, to incorporate the information from both observed and unobserved data streams, we construct discretized local statistics and define the state space accordingly in Section 2.3.1.1. To manage the complexity arising from both actions, we separate the decision-making processes for sampling layout (Section 2.3.1.2) and alarm raising (Section 2.3.2). We leverage Q-learning exclusively for optimizing the sampling layout to effectively reduce the Q-table size and employ a statistical approach for alarm raising. The state and action spaces fulfill the Markov Property, a crucial assumption of RL. The BTS algorithm in Section 2.3.1.3 enhances NCQ performance by allowing multiple actions within consecutive data acquisition times. Finally, we establish a reward scheme in Section 2.3.1.4 aimed at promoting prompt anomaly detection.

We now present the detailed description of our problem: suppose there are M data streams of interest in total and $\boldsymbol{X}(t) = \left(X_1(t), \ldots, X_M(t)\right)'$ denotes the numeric observations of data streams $1, \ldots, M$ at the data acquisition time t, where $X_j(t) \in \mathbb{R}$ and $t \in \mathbb{Z}^+$. Note that the observations can be any real numbers including the discrete ones, but in this paper, we mainly focus on monitoring heterogeneous data streams that take numerical values as generated by sensors in IoT systems. In addition, it is a general practice to consider the data acquisition time t as discrete

sample points [15]. Out of M data streams, only m of them are observable at any time t due to resource constraints. Let $\mathcal{O}(t)$ and $\mathcal{U}(t)$ denote the sets of observable and unobservable data streams at time t, respectively. We then have $\mathcal{O}(t) \cup \mathcal{U}(t) = \{1, \ldots, M\}$, $\mathcal{O}(t) \cap \mathcal{U}(t) = \emptyset$, and $|\mathcal{O}(t)| = m$ for each acquisition time t, where $|\cdot|$ denotes the cardinality of a set. In addition, the following assumptions are made: (i) When the process is in control, the mean and standard deviation of $X_j(t)$, $j = 1, \ldots, M$, are 0 and 1. This can be achieved by pre-centering and pre-scaling historical in-control data. (ii) At some unknown time T, the process becomes out of control, and the mean $\boldsymbol{\mu} = \mathbb{E}\big(\boldsymbol{X}(t)\big)$ shifts away from $\mathbf{0}$ to an unknown mean $\boldsymbol{\delta}$. The number of shifted data streams is unknown. (iii) For $j = 1, \ldots, M$, $X_j(t)$ is independent and identically distributed (i.i.d.) over time before the shift occurs (when the process is in control) and after the shift occurs (when the process is out of control), i.e., each data stream $X_j(t)$ is i.i.d. during $t < T$ and i.i.d. during $t \geq T$. (iv) The data streams are heterogeneous in the sense that they may follow distinct and arbitrary distributions. (v) At each time, the sampling strategy can be timely executed and there is no cost for changing the sampling layout. (vi) The process has collected sufficient partially observed data streams under the in-control status in the past and the collected historical data is available for training purposes. (vii) We already know the least mean shift magnitude that we want to detect and define as an anomaly, denoted as δ. The key research goal is to propose a generic monitoring and sampling strategy to decide which subset of data streams to sample at each data acquisition time, i.e., $\mathcal{O}(t)$, such that mean shift occurring in the heterogeneous processes can be detected quickly subject to resources constraints, while maintaining a system-wide in-control average run length (ARL) requirement.

To address the aforementioned challenges, we propose a RL-based adaptive sampling and monitoring methodology by seamlessly integrating the nonparametric cumulative sum (CUSUM)

approach and Q-learning for online monitoring of heterogeneous data streams with partial observations. We call this algorithm the nonparametric CUSUM Q-learning algorithm (denoted as the "NCQ" algorithm). The algorithm first uses a quantile-based nonparametric CUSUM procedure to construct local statistics for each partially observed data stream [13]. Q-learning is then applied to the state space based on discretized local statistics, resulting in the optimal policy for determining the most informative subset of data streams to observe at each acquisition time. Lastly, we follow the optimal policy learned and employ the sum of top-l local statistics as a global monitoring statistic to detect a wide range of possible mean shifts. Consequently, the proposed NCQ algorithm has the following unique advantages: (i) NCQ satisfies the Markov property by defining an appropriate state and action space. This lays the foundation of MDP, and NCQ fully utilizes this internal MDP structure without requiring the knowledge of transition probability distribution. (ii) Different from the existing heuristic approaches [4], [13], [16] which act myopically and only focus on the short-term reward for updating the sampling strategy, NCQ considers long-term discounted rewards and thus produces a more effective sampling scheme. In other words, based on the asymptotic convergence property of Q-learning [17], after trained for sufficient long time, at any time, the action (sampling layout) taken by NCQ is optimal given the current state (data information) in the long run. (iii) NCQ naturally takes care of the exploitation versus exploration trade-off without the need for adding heuristic compensation designed for exploration purpose, which is heavily used in existing approaches [4], [9].

The remainder of this article is organized as follows. In Section 2.2, we review the literature on RL and Q-learning as well as CUSUM-based methodologies, which lay the groundwork for our proposed method. Section 2.3 presents technical details of the proposed NCQ algorithm. In Section 2.4, we conduct a series of simulation studies to evaluate the proposed method and compare it with

several benchmark methods. A case study on the Tennessee Eastman process is further carried out to demonstrate the effectiveness of the proposed method in Section 2.5. Section 2.6 draws a conclusion and discusses future directions.

2.2 Literature Review

In Section 2.2.1, we first review the basic framework of RL and its subfield Q-learning. Then, Section 2.2.2 reviews the CUSUM-based methodologies and extensions with the consideration of partial observations. In Section 2.2.3, we provide a concise overview of the local statistics construction in the QUANTS algorithm [13], since our approach utilizes these local statistics as a demonstration.

2.2.1 RL and Q-learning

Since our proposed methodology relies on Q-learning, we first present some basic foundations of RL [17] and then introduce Q-learning.

2.2.1.1 Basic RL Framework

RL is a branch of machine learning explicitly designed for taking suitable actions to maximize the cumulative reward. The core idea of RL is to improve the policy that returns a series of actions to take, based on the immediate rewards and cumulative weighted rewards (also denoted as values), through interaction with the environment.

At each time epoch t, the agent is in a state S_t, facing n possible actions to take $A_t^1, A_t^2, \dots, A_t^n$. Each of A_t^i brings back an immediate reward R_t^i and sends the agent to a new state S_{t+1}. A ***reward*** R_t is a number the environment sends to the agent indicating the performance level at time t. A ***value*** is the discounted cumulative reward, representing the goodness of each state based on the

long-term expected cumulative rewards. The discounted cumulative reward (also known as discounted return) is defined by

$$G_t \stackrel{\text{def}}{=} R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \quad (1)$$

where γ is a discount rate such that $0 \leq \gamma \leq 1$. Note $G_t = R_{t+1} + \gamma G_{t+1}$. A ***policy*** $\pi(a|s) = P[A = a|S = s]$ is a probability distribution of taking a certain action at a given state, which fully defines an agent's behavior mapping from state to action. The policy is designed, controlled, and constantly modified by the agent based on the interaction with the environment.

In the RL framework, it is usually assumed that the system satisfies the Markov property:

$$p(s', r|s, a) = \Pr(S_t = s',\ R_t = r|S_{t-1} = s,\ A_{t-1} = a). \quad (2)$$

This means that given the current state s and action a, the probability of receiving reward r and going to the next state s' does not depend on the history. The ***state-value function*** of a state s under a policy π is defined as the discounted return when starting in s and following π thereafter:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s].$$

Similarly, the ***action-value function*** is defined as the discounted return starting from s, taking action a, and following policy π:

$$q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a].$$

Consequentially, there is a recursive relationship between $v_\pi(s)$ and $v_\pi(s')$ derived from the equations above, known as the Bellman equation for v_π and q_π : $v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)(r + \gamma v_\pi(s'))$; $q_\pi(s, a) = \sum_{s'} \sum_r p(s', r|s, a)\big(r + \gamma v_\pi(s')\big)$. A policy π is said to outperform another π' if $v_\pi(s) \geq v_{\pi'}(s)$ for $\forall s$, and the agent's objective is to find an optimal policy that is better than or equal to all the other policies. Specifically, the optimal policy π^* identifies the values $v^*(s), q^*(s, a)$ such that

$$v^*(s) = \max_{\pi} v_\pi(s),\ q^*(s,a) = \max_{\pi} q_\pi(s,a)\ \ and\ v^*(s) = \max_{a} q^*(s,a).$$

The training of RL can be regarded as an iterative process of policy evaluation and improvement as shown below. Starting from the initial policy and state, we iteratively evaluate the value function under that policy and then find the best policy given the function values estimated, and repeat the process until we end up with finding the optimal policy:

$$\pi_0 \xrightarrow{Evaluate} q_{\pi_0} \xrightarrow{Improve} \pi_1 \xrightarrow{Evaluate} q_{\pi_1} \xrightarrow{Improve} \ldots \xrightarrow{Improve} \pi^* \xrightarrow{Evaluate} q^*(s).$$

2.2.1.2 Q-Learning

There are many approaches in RL when the underlying MDP is unknown. In this article, we focus on Q-learning algorithm which is one of the most popular RL algorithms. If the state transition model is available, directly applying partially observable Markov decision process (POMDP) algorithms is more efficient. For complex models, Q-learning is a good option to learn an optimal policy through environment interaction without knowing the underlying model. Among reinforcement learning algorithms, model-based approaches aim to learn a model of the environment, such as the transition dynamics and the reward function, and use this model to plan and make decisions. For example, the Model-Based Interval Estimation Reinforcement Learning (MBIE-EB) [18] uses a Bayesian approach to estimate the model of the environment and then makes decisions. Other approaches such as the Dyna algorithm [19] learns a model of the environment through keeping interacting with the environment and then uses this model to generate simulated experience and update the value function. We chose Q-learning due to the complexity of the environment's model, which hampers the efficiency and accuracy of model-based RL techniques. Specifically, Q-learning focuses on the action-value function, i.e., the Q value $Q(S, A)$, for each state (S) and action (A) pair. In this way, a Q-table is constructed in which

the states in the state space serve as the rows and the actions in the action space are the columns, and each Q value corresponds to an entry in the Q-table.

In general, Q-learning consists of two important steps, policy evaluation and policy improvement. In addition, it also involves two different policies, namely the target policy and the behavior policy. The target policy is used to estimate the value functions during policy evaluation and the behavior policy is used to control the process of policy improvement. For evaluation step, given a target policy π, the algorithm updates the value function $Q(S, A)$ towards the estimated return as follows:

$$Q(S, A) \leftarrow Q(S, A) + \alpha\left(R + \gamma \max_{a'} Q(S', a') - Q(S, A)\right), \tag{3}$$

where α is the step size. This equation, commonly referred to as the Bellman equation for Q-learning, serves as an iterative update formula crucial for adjusting Q-values. Q-values signify the anticipated cumulative future rewards associated with taking a specific action in a given state. Essentially, the equation dictates that the updated Q-value for a state-action pair results from a combination of the previous Q-value and a novel estimate derived from both the immediate reward and the highest expected future rewards. This iterative update process unfolds as the agent engages with the environment, facilitating the algorithm's convergence towards optimal Q-values. Note that the target policy is a greedy search during policy evaluation:

$$\pi(S') = argmax_{a'} Q(S', a'). \tag{4}$$

For the improvement step, the behavior policy is improved over $Q(S, a)$ with an ϵ-greedy search:

$$\pi(a|S) = \begin{cases} 1 - \epsilon, & a^* = argmax_{a \in A}\, Q(S, a) \\ \epsilon, & a^* = U(a \in A|S) \end{cases}, \tag{5}$$

where $U(a \in A|S)$ stands for a discrete uniform distribution of all feasible actions a in the

action space (A) given current state S. In essence, with the probability ϵ, the algorithm chooses an action randomly, and with the probability $1-\epsilon$, the algorithm selects the action that brings the highest Q value given the current state. As a result, the algorithm estimates the total discounted reward for state-action pairs assuming the target policy was followed despite that it actually follows the behavior policy which is ϵ-greedy based. Nonetheless, literature has shown that the policy learned through Q-learning converges asymptotically to the optimal one with probability 1[17].

Overall, the process involves initializing a Q-table, where each entry represents the expected cumulative future reward for a state-action pair. The agent alternates between exploration and exploitation, selecting actions, executing them, and observing the resulting state and reward. Q-values are then updated using the Equation 3, incorporating immediate rewards and expected future rewards. This iterative process continues until the agent converges to an optimal policy. Exploration rates may decay over time, encouraging the agent to rely more on exploitation. The learned Q-values are used to extract the optimal policy, which specifies the best action for each state. The final policy is tested and refined as needed.

2.2.2 CUSUM with partial observations

Next, we focus on reviewing the CUSUM-based methodologies, and the extensions with the consideration of partial observations. The CUSUM procedure was first developed by [20], who proposed to monitor a univariate variable. Afterwards, multivariate CUSUM charts were studied extensively to detect shifts in multiple quality characteristics [21]–[23]. However, all these works assume that the variables follow normal or some well-known distributions. To address this issue, nonparametric multivariate CUSUM procedures have been developed. For example, [10], [11] proposed a multivariate nonparametric CUSUM procedure (denoted as “QH01 method”) based on

monitoring the antiranks of all data streams at each time. [24] used a nonparametric double homogeneously weighted moving average sign control chart. Unfortunately, these multivariate CUSUM methods assume that full observations are available at each time and do not consider partial observations due to resource constraints.

To address this issue, there are a few works taking resource constraints into consideration. For example, [4] proposed a top-r based adaptive sampling strategy for high-dimensional process monitoring, where a constant imputation parameter was introduced to compensate for unobserved data streams. [16] proposed a spatial-adaptive sampling and monitoring method that uses the spatial information of data streams to improve detection speed. [25] proposed a detection statistic that incorporates sparse change information as the reward of a multi-armed bandit problem. Unfortunately, all these approaches make normal assumptions on data distributions. To address this issue, [9] proposed a nonparametric adaptive sampling algorithm (denoted as the "NAS" algorithm) with resource constraints assuming data distributions are exchangeable. However, it has been shown that the antirank-based NAS algorithm is less sensitive to detect small mean shifts, and the antirank-based procedures assume that full observations of all data streams are available offline from the in-control process. In addition, due to the exchangeable assumption, the NAS algorithm fails to quickly detect mean shifts for heterogeneous processes. Recently, [13] proposed the QUANTS algorithm that does not make data distribution assumptions, i.e., distribution-free. However, this algorithm is not able to take advantage of the MDP structure and focuses on short term gain, which leads to a longer detection delay especially when the mean shift is small or the resource is limited (i.e., more challenging detection scenarios). Since we adopt the construction of the local statistics in the QUANTS algorithm, we briefly review the mathematical details in Section 2.2.3

In summary, the existing literature does not fully utilize the internal MDP structure embedded in the monitoring problem nor consider leveraging the power of RL yet. To fill the research gap, we propose a nonparametric CUSUM monitoring and sampling methodology based on Q-learning in this paper.

2.2.3 Local Statistics Construction

In this subsection we briefly introduce the construction of the local statistics in the QUANTS algorithm [13], as our proposed method employs this local statistics. We are interested in detecting both positive and negative mean shifts in any variable. Thus, a two-sided local statistic for each variable j at time t is defined as [26]

$$W_{j,t} = max\left(W_{j,t}^{+}, W_{j,t}^{-}\right), \tag{6}$$

where $W_{j,t}^{+}, W_{j,t}^{-}$ are the local statistics for detecting positive and negative mean shifts, respectively. The intuition to construct $W_{j,t}^{+}$ and $W_{j,t}^{-}$ is to keep track of the quantile-based indicators used in the QUANTS algorithm and check whether an underlying change takes place in the distribution of the indicators. Specifically, let $I_{j,1} = \left(-\infty, q_{j,1}\right], I_{j,2} = \left(q_{j,1}, q_{j,2}\right], \ldots, I_{j,d} = \left(q_{j,d-1}, +\infty\right)$ be the partition of the real line, where $-\infty < q_{j,1} < q_{j,2} < \ldots < q_{j,d-1} < +\infty$ are $d-1$ boundary points of the partitioning intervals for data stream $j = 1, \ldots, M$. According to [13], the values of $q_{j,l}$ can be set as the l/dth quantile of the in-control distribution of data stream j. Define the quantile-based indicators $\boldsymbol{Y}_j(t) = \left(Y_{j,1}(t), \ldots, Y_{j,d}(t)\right)^T$, where

$$Y_{j,l}(t) = \mathbb{1}\{X_j(t) \in I_{j,l}\}, \tag{7}$$

denotes whether interval $I_{j,l}$ contains the measurement $X_j(t)$ for data stream j. Then, it is reasonable to model $\boldsymbol{Y}_j(t)$ by a categorical distribution, $\boldsymbol{Y}_j(t) \sim Cat\left(\boldsymbol{\theta}_j(t)\right)$, where $\boldsymbol{\theta}_j(t) =$

$(\theta_{j,1}(t), \dots, \theta_{j,d}(t))$ is a time-varying parameter, and each element denotes the probability of the corresponding partition interval containing $X_j(t)$. Define $\boldsymbol{A}_j^+(t) = \left(A_{j,l}^+(t), \dots, A_{j,d-1}^+(t)\right)^T$ and $\boldsymbol{A}_j^-(t) = \left(A_{j,l}^-(t), \dots, A_{j,d-1}^-(t)\right)^T$, where

$$A_{j,l}^+(t) = 1 - \sum_{i=1}^{l} Y_{j,i}(t)\,, A_{j,l}^-(t) = \sum_{i=1}^{l} Y_{j,i}(t)\,, \tag{8}$$

$j = 1, \dots, M$ and $l = 1, \dots, d-1$. With this transformation, detecting mean shifts in the distribution of $X_j(t)$ is equivalent to detecting shifts in the distribution of $\boldsymbol{A}_j^+(t)$ and $\boldsymbol{A}_j^-(t)$. In particular, $\boldsymbol{A}_j^+(t)$ is more sensitive to detect upward mean shifts while $\boldsymbol{A}_j^-(t)$ is more sensitive to detect downward mean shifts according to the results in [13]. We then have the following recursive updates for the upward direction:

$$C_{j,t}^+ = \left(\boldsymbol{S}_{j,t-1}^{+(1)} - \boldsymbol{S}_{j,t-1}^{+(2)} + \boldsymbol{A}_j^+(t) - \boldsymbol{g}_j^+\right)^T \cdot$$

$$\left(diag\left(\boldsymbol{S}_{j,t-1}^{+(2)} + \boldsymbol{g}_j^+\right)\right)^{-1} \cdot \left(\boldsymbol{S}_{j,t-1}^{+(1)} - \boldsymbol{S}_{j,t-1}^{+(2)} + \boldsymbol{A}_j^+(t) - \boldsymbol{g}_j^+\right), \tag{9}$$

where $\boldsymbol{g}_j^+ = \left(g_{j,1}^+, \dots, g_{j,d-1}^+\right)^T$ is the expectation of $\boldsymbol{A}_j^+(t)$ when process is in control, $g_{j,l}^+ = 1 - \frac{l}{d}$ for $j = 1, \dots, M$ and $l = 1, \dots, d-1$, and

$$\begin{cases} \boldsymbol{S}_{j,t}^{+(1)} = \boldsymbol{0}, \boldsymbol{S}_{j,t}^{+(2)} = \boldsymbol{0} & \text{if } C_{j,t}^+ \le k, \\ \boldsymbol{S}_{j,t}^{+(1)} = \frac{\left(\boldsymbol{S}_{j,t-1}^{+(1)} + \boldsymbol{A}_j^+(t)\right)\left(C_{j,t}^+ - k\right)}{C_{j,t}^+} & \\ \boldsymbol{S}_{j,t}^{+(2)} = \frac{\left(\boldsymbol{S}_{j,t-1}^{+(2)} + \boldsymbol{g}_j^+\right)\left(C_{j,t}^+ - k\right)}{C_{j,t}^+} & \text{if } C_{j,t}^+ > k. \end{cases} \tag{10}$$

Intuitively, $C_{j,t}^+$ in (9) measures the cumulative difference between the actual status of the process ($\boldsymbol{A}_j^+(t)$) and the in-control expectations ($\boldsymbol{g}_j^+$) in the upward direction. The local statistic for detecting the upward mean shift is then defined as

$$W_{j,t}^{+} = \left(\boldsymbol{S}_{j,t}^{+(1)} - \boldsymbol{S}_{j,t}^{+(2)}\right)' \cdot \left(diag\left(\boldsymbol{S}_{j,t}^{+(2)}\right)\right)^{-1} \cdot \left(\boldsymbol{S}_{j,t}^{+(1)} - \boldsymbol{S}_{j,t}^{+(2)}\right),$$

which is equivalent to the following equation

$$W_{j,t}^{+} = \max\left(C_{j,t}^{+} - k, 0\right). \tag{11}$$

Here, k can be regarded as the allowance parameter. Similarly, for the downward direction, we have:

$$C_{j,t}^{-} = \left(\boldsymbol{S}_{j,t-1}^{-(1)} - \boldsymbol{S}_{j,t-1}^{-(2)} + \boldsymbol{A}_{j}^{-}(t) - \boldsymbol{g}_{j}^{-}\right)^{T} \cdot$$

$$\left(diag\left(\boldsymbol{S}_{j,t-1}^{-(2)} + \boldsymbol{g}_{j}^{-}\right)\right)^{-1} \cdot \left(\boldsymbol{S}_{j,t-1}^{-(1)} - \boldsymbol{S}_{j,t-1}^{-(2)} + \boldsymbol{A}_{j}^{-}(t) - \boldsymbol{g}_{j}^{-}\right), \tag{12}$$

where $\boldsymbol{g}_{j}^{-} = \left(g_{j,1}^{-}, \ldots, g_{j,d-1}^{-}\right)^{T}$, $g_{j,l}^{-} = \frac{l}{d}$ for $j = 1, \ldots, M$ and $l = 1, \ldots, d-1$, and

$$\begin{cases} \boldsymbol{S}_{j,t}^{-(1)} = \boldsymbol{0}, \boldsymbol{S}_{j,t}^{-(2)} = \boldsymbol{0} & \text{if } C_{j,t}^{-} \le k, \\ \boldsymbol{S}_{j,t}^{-(1)} = \frac{\left(\boldsymbol{S}_{j,t-1}^{-(1)} + \boldsymbol{A}_{j}^{-}(t)\right)\left(C_{j,t}^{-} - k\right)}{C_{j,t}^{-}} & \\ \boldsymbol{S}_{j,t}^{-(2)} = \frac{\left(\boldsymbol{S}_{j,t-1}^{-(2)} + \boldsymbol{g}_{j}^{-}\right)\left(C_{j,t}^{-} - k\right)}{C_{j,t}^{-}} & \text{if } C_{j,t}^{-} > k \end{cases} \tag{13}$$

Then, the local statistic for detecting the downward mean shift is

$$W_{j,t}^{-} = max\left(0, C_{j,t}^{-} - k\right). \tag{14}$$

For data stream $j \in \mathcal{O}(t)$, $\boldsymbol{Y}_j(t)$ is directly observed, and thus $W_{j,t}^{+}$ and $W_{j,t}^{-}$ can be calculated straightforwardly. In addition, we can use the Bayesian approach to online update $\boldsymbol{\theta}_j(t)$. Considering the conjugate prior $\boldsymbol{\theta}_j(t) \sim Dir(\boldsymbol{\alpha}_j(t-1))$, where $Dir(\cdot)$ denotes the Dirichlet distribution, and $\boldsymbol{\alpha}_j(t-1) = (\alpha_{j,1}(t-1), \ldots, \alpha_{j,d}(t-1))'$ is the concentration parameter of the Dirichlet distribution for data stream j at time $t-1$. Then the posterior distribution of $\boldsymbol{\theta}_j(t)$ can be computed as $\boldsymbol{\theta}_j(t)|\boldsymbol{Y}_j(t) \sim Dir(\boldsymbol{\alpha}_j(t))$ and

$$\boldsymbol{\alpha}_j(t) = \boldsymbol{\alpha}_j(t-1) + \boldsymbol{Y}_j(t). \tag{15}$$

In this way, the posterior mean, $\widehat{\boldsymbol{\theta}}_j(t)$ can be used as the point estimator of the true parameter $\boldsymbol{\theta}_j(t)$:

$$\widehat{\boldsymbol{\theta}}_j(t) = \frac{\boldsymbol{\alpha}_j(t)}{\sum_{l=1}^{d} \alpha_{j,l}(t)}, \tag{16}$$

where $\widehat{\boldsymbol{\theta}}_j(0) = \left(\frac{1}{d}, \ldots, \frac{1}{d}\right)'$.

For data stream $j \in \mathcal{U}(t)$, $\boldsymbol{Y}_j(t)$ is unobserved. Recall that $\boldsymbol{Y}_j(t) \sim Cat\left(\boldsymbol{\theta}_j(t)\right)$. Thus, to address this issue, if data stream $j \in \mathcal{U}(t)$, we generate a pseudo observation $\boldsymbol{Y}_j(t)$ based on the most updated $\widehat{\boldsymbol{\theta}}_j(t)$, then update $\boldsymbol{A}_j^+(t)$ and $\boldsymbol{A}_j^-(t)$, and construct $W_{j,t}^+$ and $W_{j,t}^-$ based on equations (11) and (14). However, since the generated sample is not real, it will not be used to update $\boldsymbol{\theta}_j(t)$ nor $\boldsymbol{\alpha}_j(t)$, i.e., $\widehat{\boldsymbol{\theta}}_j(t) = \widehat{\boldsymbol{\theta}}_j(t-1)$ and $\boldsymbol{\alpha}_j(t) = \boldsymbol{\alpha}_j(t-1)$ for $j \in \mathcal{U}(t)$. Similarly, $\boldsymbol{S}_{j,t}^{+(1)} = \boldsymbol{S}_{j,t-1}^{+(1)}, \boldsymbol{S}_{j,t}^{+(2)} = \boldsymbol{S}_{j,t-1}^{+(2)}, \boldsymbol{S}_{j,t}^{-(1)} = \boldsymbol{S}_{j,t-1}^{-(1)}$, and $\boldsymbol{S}_{j,t}^{-(2)} = \boldsymbol{S}_{j,t-1}^{-(2)}$ for $j \in \mathcal{U}(t)$.

2.3 Methodology

The proposed methodology involves offline training and online implementation. The offline training consists of two stages. In Stage-I, the nonparametric CUSUM procedure first transforms the incoming raw data into continuous local statistics [13] for each data stream, and then discretizes the continuous local statistics which forms our state space. Then the Q-learning algorithm is applied to learn an optimal sampling policy. In Stage-II, based on the derived optimal policy, we learn the appropriate threshold for raising an alarm with a specified in-control ARL. Once the offline training stage is completed, the optimal policy and threshold are used for real-time process monitoring.

It is important to note that while Q-learning has been successful in solving real-world problems and is widely used in RL, there are unique challenges in applying it to the monitoring problem here. Specifically, it is not clear how to define a reasonable state and action space pair that can fully utilize the partial observation information and satisfy the Markov property which is the key theoretical assumption of Q-learning. When defining the state space, how the agent is able to differentiate between unobserved data streams to decide which ones to observe for the next time epoch is also a great challenge. Since the size of Q-table grows rapidly as the number of state and action pairs increases, how to effectively reduce the size of the Q-table to make it scalable is another practical challenge. Finally, how to define the reward to expedite anomaly detection is also unclear. We will address each of these challenges below. In particular, we first present how to construct local statistics and elaborate the Q-learning details in Section 2.3.1. Then in Section 2.3.2, we describe how to build the global statistics and learn the threshold given the prescribed in-control ARL. We provide real-time online implementations in Section 2.3.3. Next, we discuss the parameter settings of the NCQ algorithm in Section 2.3.4.

2.3.1 Offline Stage-I Q-learning

In this subsection, we elaborate the core elements in RL: the *state* space, the *action* space, and the *reward*. Then, we present an overview of Stage-I Q-learning. Here, we choose the CUSUM-based procedure [13] to be our baseline control chart as a demonstration (see Section 2.2.3 for detail) due to its effectiveness and wide applicability in practice. Yet, our proposed RL framework is not limited to the CUSUM-based procedure, and it can also be easily integrated with other efficient univariate control charts as well.

2.3.1.1 State Space

We have transformed the raw observations into local statistics. However, these statistics are continuous, while Q-learning requires a finite state space. Therefore, we need to discretize the local statistics. Recall that there are in total M data streams and only m of them are observable at each time epoch. We then propose to define the state space as follows: each possible state is a one by M vector $(\beta_1, \beta_2, \dots, \beta_M)$, where $\beta_j \in \{1, \dots, 2n\}$. Each entry β_j represents the discretized local statistic of the corresponding data stream j at a certain time epoch. Specifically, $1,2, \dots, n$ represents the level of shift suspiciousness for observed data streams and $n+1, \dots, 2n$ represents the level of shift suspiciousness inferred by the agent for unobserved data streams. The suspiciousness levels of $n+1$ and 1 are the same, differing only in whether the data stream is observed or not. A higher shift suspiciousness level (e.g., $n > n-1$) indicates a greater likelihood of a shift in the data stream.

First, we focus on observed data streams $j \in \mathcal{O}(t)$ and decide the rule for discretizing the continuous local statistics, W_j, to a discretized one, β_j. The idea is to calculate sufficiently many W_j's based on the historical in-control data and then sort them in an ascending order $W_{j,(1)} < W_{j,(2)} < \cdots < W_{j,(\tau)}$, pick the corresponding $\frac{1}{n}, \frac{2}{n}, \dots, \frac{n-1}{n}$ percentile local statistics $W^*_{j,1}, W^*_{j,2}, \dots, W^*_{j,n-1}$ as the boundaries, such that for newly acquired local statistics $W_{j,t}$ where $j \in \mathcal{O}(t)$, we have

$$\beta_{j,t} = \begin{cases} 1 \text{ if } W_{j,t} \in \left[0, W^*_{j,1}\right) \\ \dots \\ n \text{ if } W_{j,t} \in \left[W^*_{j,n-1}, \infty\right) \end{cases} \text{for } j \in \mathcal{O}(t). \tag{17}$$

Differentiating the suspiciousness levels of both observed and unobserved data streams is crucial. If all unobserved states are treated as a single state, the agent won't be able to distinguish

between different unobserved data streams and make informed decisions on which ones to observe next, even if some data streams have been unobserved for a while. To discretize unobserved data streams $j \in \mathcal{U}(t)$, we follow the same procedure as for $j \in \mathcal{O}(t)$, and simply add n to the discretized state to indicate that j is not being observed, i.e.,

$$\beta_{j,t} = \begin{cases} n+1 \text{ if } W_{j,t} \in [0, W_{j,1}^*) \\ \dots \\ 2n \text{ if } W_{j,t} \in [W_{j,n-1}^*, \infty) \end{cases} \text{ for } j \in \mathcal{U}(t). \quad (18)$$

Using this formulation, the agent establishes a unified standard for communicating with both observed and unobserved discretized states, enabling it to allocate resources based on a systematic rule. An illustrative example can be found in Section 2.3.1.5.

One practical challenge here is that we may only have limited historical in-control data to learn the boundaries $W_{j,1}^*, W_{j,2}^*, \dots, W_{j,n-1}^*$. To deal with this challenge, existing studies bootstrap the available in-control data and here we adopt the same strategy [4], [5], [9], [13]. For each training episode, we bootstrap the available historical in-control data T^0 times with replacement to construct one episode data that contains T^0 in-control time epochs, and then learn the boundaries $W_{j,1}^*, W_{j,2}^*, \dots, W_{j,n-1}^*$. In addition, as in each episode, CUSUM statistics need some time to stabilize. Thus, when calculating and learning boundaries $W_{j,1}^*, W_{j,2}^*, \dots, W_{j,n-1}^*$, we discard the first $warm\_up\_epoch$ time epochs in each episode to obtain more accurate boundary estimations.

2.3.1.2 Action Space

At each time epoch, two main decisions need to be made: the dynamic sampling strategy and whether to raise an alarm. However, including the alarm decision in the Q-learning action space would significantly increase the Q-table size. In addition, the agent will always make the same decision (i.e., either raising an alarm or not) when facing the same discretized local statistics $S_t =$

$(\beta_{1,t}, \beta_{2,t}, \dots, \beta_{M,t})$, whereas the continuous local statistics $(W_{1,t}, \dots, W_{M,t})$ might be different. This will lead to either more false alarms or longer detection delays. To address this, we only include the dynamic sampling decision in the Q-learning action space, while learning a global threshold for the alarm decision using continuous local statistics in Section 2.3.2. This separation of optimal policy learning and global threshold learning reduces the problem's complexity.

There are two unresolved questions: how to define the action space to reduce the Q-table size and whether the formulation satisfies the Markov property as in equation (2). We first focus on the first challenge. Technically, at each epoch, we can freely decide the data streams to be observed. Unfortunately, the cardinality of the action space will become very large $\binom{M}{m}$. Therefore, we introduce a strategy where only one resource, i.e., one data stream, can be adjusted by adding or removing it from the current sampling layout at each epoch. This defines our action space $\mathbb{A}$ as (i, j), where $i \in \{1,2, \dots, M\}$ represents the data stream to remove and $j \in \{1,2, \dots, M\}$ represents the data stream to allocate the resource to. The cardinality of the action space $\mathbb{A}$ is M^2, independent of m. Furthermore, the Q-table is sparse since most actions are not feasible for the current state. For example, given $S_t = (\beta_{1,t}, \beta_{2,t}, \dots, \beta_{M,t})$, if $\beta_{i,t} \geq n + 1$, indicating data stream i is not observed at time t, we cannot remove the resource from it. This significantly reduces the size of the Q-table and the number of parameters to be learned so that we can store and update the Q-matrix in a sparse matrix format. For convenience, we call the (x, y) entry of the Q-table *infeasible* if the yth action is infeasible for the xth state and set it to $-\infty$ ($Q(x, y) = -\infty$). These *infeasible* entries remain unchanged throughout the learning process. An illustrative example is provided in Section 2.3.1.5.

With the state and action spaces defined, we now review the Markov Decision Process framework and examine the Markov property. First, the state and action are closely linked, where

each state vector can only take two sets of mutually exclusive values based on whether the data stream is monitored or not. The validity of actions depends on the current state's specific values. Second, the agent should consider the discretized local statistic values to take actions, which reflect the level of suspiciousness that a mean shift has taken place. As the action selection relies on the current state, the process is indeed a Markov Decision Process (MDP). Given the current state $S_t = (\beta_{1,t}, \beta_{2,t}, \dots, \beta_{M,t})$ and the action $A_t = (i, j)$, $S_{t+1} = (\beta_{1,t+1}, \dots, \beta_{M,t+1})$ does not depend on the history information and is purely decided by S_t, A_t and future observations at $t + 1$. In particular, the continuous local statistic W_j^{t+1} of each data stream j at time epoch $t + 1$ is calculated only based on W_j^t and the data obtained at time epoch $t + 1$, $X_j(t + 1)$. Since A_t determines whether data stream j is being observed, β_j^{t+1} depends on W_j^{t+1} and A_t, following the discretization rule in equations (17) and (18). The reward R_t also only depends on whether the shifted data streams are selected to be monitored and the true system status at time t. Therefore, we would like to highlight that the Markov property is satisfied in the context of our proposed NCQ method and with this key underlying property, it can be proven that the policy learned through Q-learning converges to the optimal policy [17]. Limiting the action to reallocating only one resource per time epoch slows down the detection speed. To mitigate this issue, we introduce the Between-Time Sampling (BTS) algorithm in the next subsection, which aims to reduce the detection delay.

2.3.1.3 The BTS Algorithm

The intuition of the BTS algorithm is to mimic the behavior that one can freely switch resources on data streams while maintaining the current feasible size of the Q-table. Specifically, between the two adjacent data acquisition time epochs t and $t + 1$, we divide the time window into several

"sub" artificial timestamps $t^0, t^1 \ldots, t^\omega$ such that $t^0 = t$, and $t^\omega = t + 1$. At each of these "sub" timestamps t^k, the agent performs an action following the current policy with the discretized local statistics information acquired at t^0. In this way, the agent can perform a series of actions, i.e., switching resources among multiple data streams from t to $t + 1$. Note that, the continuous local statistics we use in these timestamps $t^1, \ldots, t^{\omega-1}$ are from time epoch t and fixed at these timestamps until we acquire new data information at $t + 1$. As the data streams being monitored change at each sub-time stamp, the discretized local statistics and the corresponding state also change. By repeatedly reallocating resources at these sub-time stamps, we can mimic the behavior of free resource reallocations, enabling prompt anomaly detection. The detection delay diminishes with an increasing ω, as a larger ω enhances the emulation of resource-free reallocation behavior. However, the precise correlation between ω and detection delay, as well as the optimal choice of ω, requires further exploration. Due to space constraints, we have deferred this investigation to future research. Typically, setting $\omega = m$ is sufficient because by reallocating a resource to one data stream at each timestamp, the algorithm allows the agent to reallocate in total m data streams from t to $t + 1$, which can lead to select an entirely different set of data streams. Computationally, since the policy remains fixed among the sub-timestamps, only value extraction from a sparse matrix is performed, rendering the increase in computational effort tolerable. A detailed example of the implementation of BTS is provided in Section 2.3.1.5.

2.3.1.4 Reward

Now, at any given time epoch t, we are at a state $S = (\beta_1, \beta_2, \ldots, \beta_M)$. Recall that during the offline training, in each generated episode we know the specific time and which data stream that the mean shift is introduced at, while the agent does not. Therefore, we are able to assign the

reward according to the true status of the system and whether the decision made by the agent is good (when it chooses to monitor the shifted data stream) or not. When the system is in control at $t = 0$ and a mean shift occurs in unknown data streams at time $t = T$, the agent should not have preference for selecting which data streams to observe until $t \geq T$. This prevents potential long detection delays when mean shifts happen in unfavored data streams. Hence, we assign a negative reward regardless of the state when the process is in control. This is necessary to suppress preferences because as rewards are accumulative, a positive reward will incentivize the agent to continuously select the same data stream for monitoring, regardless of whether a mean shift occurs or not.

Once the system is out of control, i.e., $t \geq T$, we provide different rewards based on whether the agent monitors the true shifted data streams. If the agent chooses to monitor the true shifted data streams, we assign a large positive reward that gradually decays over time. This encourages prompt anomaly detection. For example, at $t = T + \Delta$ where $0 \leq \Delta \leq \mathrm{p}$, we simply assign reward $e^{(p-\Delta)/2}$, where p is the time epochs within which we hope the mean shift to be detected. Note that to make the exponentially decreasing reward function smoother, we set the decay rate to be $e^{-0.5}$ per time epoch elapsed. One may also adjust p depending on the specific problem and practical need. Based on empirical studies, we found that $p = 16$ performs well enough. However, one issue is that the CUSUM may not be large enough in the first a few epochs after the shift occurs. Consequently, the agent struggles to differentiate between small CUSUM values resulting from the in-control or out-of-control process status, leading to difficulties in the convergence of NCQ. Therefore, we introduce a hyperparameter $epoch_{buffer}$ such that $[T, T + epoch_{buffer})$ will be treated indifferently as in control and receives the same in control reward. This approach still promotes exploration during $t \in [T, T + epoch_{buffer})$. Based on our empirical studies,

$epoch_{buffer} = 4$ performs well. Thus, when the agent chooses to observe the true shifted data stream, we modify the assigned reward to be $e^{(p+epoch_{buffer}-\Delta)/2}$ if $\Delta \in [epoch_{buffer}, p + epoch_{buffer}]$ and reward to be -1 if $\Delta \in [0, epoch_{buffer})$. For $\Delta > p + epoch_{buffer}$, we assign a constant reward 1 if the agent correctly monitors the true shifted data stream.

In contrast, when the process is out of control and the agent fails to observe the true shifted data stream, we assign a constant negative reward, such as -10, as a punishment. Following the exact same logic as above, we use $epoch_{buffer} = 4$ and at time $t = T + \Delta$, we assign reward -10 when $\Delta > epoch_{buffer}$ and reward -1 when $\Delta \in [0, epoch_{buffer})$.

The proposed reward scheme aims to encourage exploration when local statistics are small, indicating that the system is under control. Negative rewards are assigned during control periods to prevent the agent from developing a preference for monitoring specific data streams. Conversely, when the local statistic $W_{j,t}$ is large for data stream j, indicating a possible mean shift, the agent prioritizes exploitation to monitor the shifted data stream correctly. This is achieved through positive rewards for correct monitoring, which decay over time for prompt detection, and a constant negative reward for failure to monitor the true shifted data stream. Overall, the reward scheme promotes exploration during periods of small local statistics and exploitation during significant local statistics. In section 2.4.4, we will further examine the impact of reward assignment on the method's performance. It's worth noting that the reward function is only used during offline training to derive the optimal policy and is not employed during online implementation.

2.3.1.5 An Illustrative Example

We provide a detailed example to illustrate the proposed NCQ Stage-I algorithm. The visual representations and the corresponding Q-table are provided in Fig. 2.1. There are in total of 6 data streams, and 2 of them are observable at each acquisition time ($M = 6, m = 2$). For simplicity, we assume that all the data streams have the same percentile boundary, i.e., $W_{i,k}^{*} = W_{j,k}^{*}\ \forall k, i \neq j$. Further, we set $\omega = 2$ in the BTS algorithm, and $n = 6$ in the state space. In other words, we introduce two pseudo sub timestamps and only one resource will be reallocated at each sub timestamp. If the local statistics are in parentheses, it means the corresponding data stream is not observed at that time. Since we have in total of 6 data streams while only 2 are observable, each row of the local statistics representation has exactly 4 values in parentheses. Recall that if data stream j is unobserved at time t, $\boldsymbol{Y}_j(t)$ is randomly drawn from the most updated $\widehat{\boldsymbol{\theta}}_j(t)$. Thus, the four continuous local statistics in parentheses are randomly generated pseudo-observations while the other two are based on real observations. The discretized local statistics follow the same pattern, but they are discretized based on percentile boundaries from the continuous local. For example, we have the first continuous local statistics $3.1 \in (2.9, 4.8)$ and thus it is discretized to 3. We also provide the Q-table for the two intermediate states at t^0 and t^1, with blank entries for values smaller than 10^4. For example, the first row of the Q-table shows all possible actions and their corresponding Q-values given a specific state $S_t = (3,7,2,10,7,8)$.

The BTS algorithm keeps the continuous local statistics unchanged between consecutive acquisition times (t and $t+1$). At the "sub" timestamp $t = t^0$, action $(3,4)$ is taken, deselecting data stream 3 and selecting data stream 4 for monitoring. Similarly, for $t = t^1$, action $(1,6)$ is taken. Then at the next acquisition time $t+1$, observations are collected from data streams 4 and 6, updating their corresponding local statistics to 3.6 and 4.6, respectively. For unobserved data

streams, their local statistics are updated based on the sampled $\boldsymbol{Y}_j(t+1) \sim Cat\left(\widehat{\boldsymbol{\theta}}_j(t+1)\right)$. This example clearly highlights the difference between existing heuristic approaches [4], [13], [16] and the NCQ algorithm. While a heuristic approach would immediately allocate resources to monitor data streams 1 and 4 due to their relatively large local statistics at time t^1, NCQ chooses to monitor data streams 4 and 6, maximizing long-term rewards by exploring when local statistics are not yet significant. NCQ strikes a balance between exploiting data streams with large local statistics (indicating potential process out-of-control) and exploring data streams with small local statistics (indicating likely process in-control), unlike existing heuristic approaches that greedily monitor data streams with relatively larger local statistics regardless of the process status, resulting in a less effective monitoring scheme.

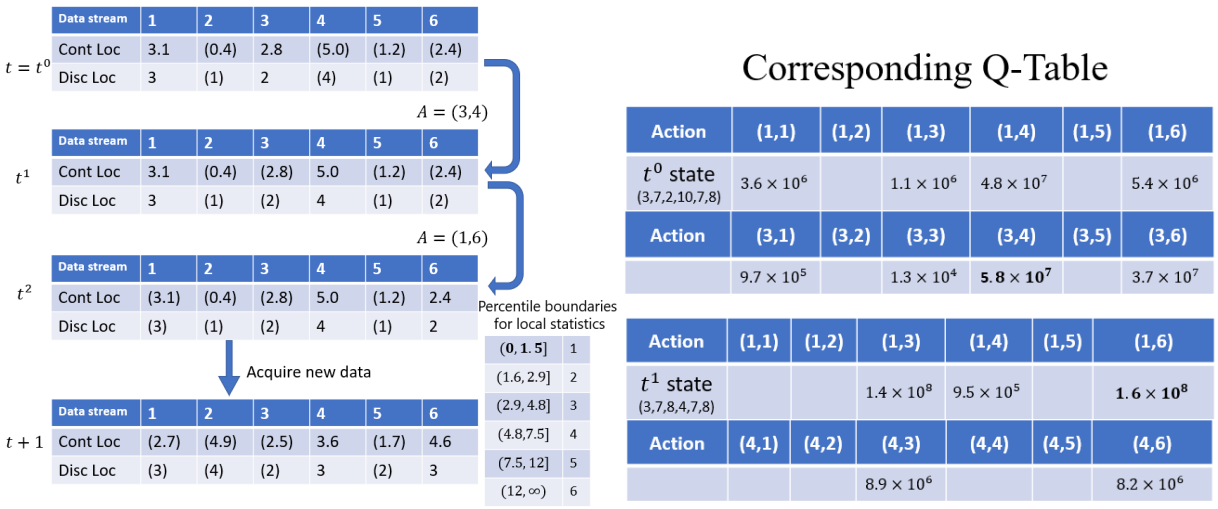


Fig. 2.1. Illustrative example of NCQ Stage-I algorithm between two consecutive acquisition times t and t+1 (cont loc: continuous local statistic; disc loc: discrete local statistic)

Lastly, note that in this example, at $t = t + 1$, NCQ treats data streams 4 and 6 equally based on their discretized states. However, as their continuous local statistics differ, a higher value of n is expected to yield better discretization and expedite the detection process. This aspect will be tested in section 2.4.3.

2.3.1.6 Overview of Stage-I Q-learning

With the Q-learning being fully introduced, we would like to draw some connections to the partially observable Markov decision process (POMDP). In our framework, the true underlying system status (in control/out of control) can be viewed as a hidden state that cannot be directly observed by the agent. Instead, the agent receives observations of the data streams. While maintaining a probabilistic representation of the agent's current knowledge about the underlying, unobservable states of the system is a common approach in POMDPs, it can be computationally demanding and may fail to converge [27]. The limitations of Q-learning in this context have been studied [28]. To address this, we utilize Q-learning to learn a dynamic sampling strategy and leave the inference of the true underlying system to the statistical process control (SPC) approach in the Stage-II algorithm. Therefore, given the discretized local statistics $S_t = (\beta_{1,t}, \beta_{2,t}, \dots, \beta_{M,t})$, the true underlying system status (i.e., the hidden state) can be in control or out of control depending on the current time epoch. Thus, the state transition in the MDP shifts following a mean shift in the process. Despite the non-stationary nature of the process, these dynamics change only once rather than constantly. As a result, Q-learning can still converge, enabling the agent to adeptly balance exploration and exploitation in the sampling layout without directly inferring the underlying system status.

Input: historical in-control data collected, hyperparameter $d, n, k, \omega, \alpha, \gamma, \epsilon, p, epoch_{buffer}$ **Output**: learned Q-table and corresponding optimal policy π^*
Initial setup: Initialize Q-table by setting *infeasible* entries to $-\infty$ and everywhere else 0.
Algorithm: While two consecutive policies π^* are not the same, repeat steps (II)-(IV):

I. Set $t = 0$. Let $W_{j,0} = W_{j,0}^{+} = W_{j,0}^{-} = 0, \boldsymbol{\alpha}_j = (1/d, 1/d, \ldots, 1/d), j = 1, \ldots, M$. Randomly select m out of M data streams to create the initial state S_0 and a data stream associated with the mean shift δ.

II. Bootstrap in-control data T^0 times and additional out-of-control data T^1 times, which makes up a dataset for one full episode with length $T^0 + T^1$.

III. For each time epoch $t = 1,2, \ldots, T^0 + T^1$:

a. Construct discretized local statistics for both observed and unobserved data streams and obtain the current state S_t.

b. Choose action A_t based on S_t using $\epsilon-$greedy policy as in (5) and take action A_t. Implement the BTS algorithm for ω times and calculate reward R_t. Next, proceed to $t + 1$ and collect new observations of data streams.

c. If $t > warm\_up\_epoch$, update Q-table as (3). Otherwise, do not update the Q-table.

IV. Derive the current optimal policy π^* given the current function values following (4).

End

Fig. 2.2. The overall flow chart of NCQ Stage-I algorithm.

When using RL to learn the optimal policy, a challenge arises in obtaining both in-control and out-of-control data for training, especially when limited out-of-control data is available. To address this, we generate out-of-control data by introducing a mean shift to the in-control data. For example, if we are interested in detecting a mean shift (δ) with at least magnitude 1, we simply add $\delta = 1$ to a randomly selected in-control data stream to obtain the out-of-control data in each training episode.

In Section 2.3.1, we have successfully addressed the challenges related to online monitoring of heterogeneous data streams with partial information using RL. Specifically, to incorporate the

information from both observed and unobserved data streams, we construct discretized local statistics and define the state space accordingly. The defined (i, j) action pairs effectively reduce the Q-table size for efficient RL. The state and action spaces fulfill the Markov Property, a crucial assumption of RL. Additionally, the BTS algorithm enhances the performance of NCQ by allowing multiple actions within consecutive data acquisition times. Lastly, we define a reward scheme to encourage quick anomaly detection. An overview of the Stage-I Q-learning process is provided in Fig. 2.2.

To enable the agent to learn the sampling layout for both in-control and out-of-control scenarios, an adequate number of in-control and out-of-control time epochs should be available for observation and learning in each training episode. Based on the empirical studies in which we tried different combinations of the number of time epochs for in-control and out-of-control scenarios, we found it is sufficient to generate around 150 time epochs for both in-control and out-of-control processes in each training episode, which can lead to a stable and efficient learning process. In particular, we set T^0 follows a uniform distribution: $T^0 \sim U(100,200)$ and $T^1 = 150$ in the numerical studies, which results in an expected number of time epochs to be 300. One thing we should highlight is that the key here is to accurately train the ranking of Q-values rather than the absolute Q-values, which helps us determine the optimal policy. Specifically, we train our agent on many episodes until the learned policy converges to ensure that the estimated Q value accurately reflects the ranking of the action given a state.

2.3.2 Offline Stage-II threshold-learning

In this subsection, we propose constructing global statistics based on the continuous local statistics of observed data streams. Given the local statistics of each **observed** data stream $j \in \mathcal{O}(t)$ at time epoch t, $W_{j,t}$, we first order the local statistics in a descending order such that $W_{(1),t} \geq$

$W_{(2),t} \geq \ldots \geq W_{(m),t}$, where $W_{(j),t}$ denotes the jth largest observed local statistic. Note that we only rank the local statistics for **observed** data streams since the inferred local statistics for unobserved data streams lack accuracy. We use the sum of top-l local CUSUM statistics as global monitoring statistic for the current time epoch t:

$$g(t) = \sum_{j=1}^{l} W_{(j),t}\,, \tag{19}$$

and raise an alarm at the stopping time $\tau = \inf\{t \geq 1: g(t) \geq threshold\}$, where the *threshold* is related to a prescribed in-control ARL, ARL_0. Existing studies have shown that the sum of top-l local CUSUM statistics is more sensitive and robust to detect mean shifts with suitable choices of l [29] (see Section 2.3.4 on how to set l). Regardless of whether the process exhibits a positive or negative mean shift in data stream i, the corresponding local statistic $W_{i,t}$ is expected to become large. As a result, our proposed approach is capable of detecting a wide range of potential shifts in all directions, without relying on assumptions about the spatial structure of variables or prior knowledge of the shift pattern.

In Stage-II, we follow the steps as in Stage-I and apply binary search to learn the *threshold.* An overview of the Stage-II threshold learning is summarized in Appendix A. We would like to point out that both Stage-I and Stage-II training are completed offline. In the numerical studies, $num\_episode$ and $in\_control\_epochs$ are set to 5000. Recall that in Stage-I, we follow the $\epsilon-$greedy policy as in equation (5) for exploration and exploitation purposes. However, in Stage-II since we have already obtained the optimal policy π^* from Stage-I, we simply follow the derived optimal policy closely.

2.3.3 Online Implementation

In this subsection, we discuss the online implementation of monitoring after learning the optimal policy and *threshold* in the offline phase. During online monitoring, we adhere to the learned optimal policy, update the local statistics for observed and unobserved data streams, and raise an alarm when $g(t)$ exceeds the *threshold*. The main computational complexity lies in calculating m local statistics $W_{j,t}$ $\left(j \in \mathcal{O}(t)\right)$, which can be done analytically. Since we calculate $\boldsymbol{S}_{j,t}^{+(1)}$ and $\boldsymbol{S}_{j,t}^{-(1)}$ recursively, memory usage remains constant over time. Therefore, the online implementation of NCQ operates efficiently in real time.

2.3.4 Parameter Setting

In this subsection, we discuss how to set the parameters involved in the NCQ algorithm, $d, l, k, \epsilon, \alpha, \gamma, n, \omega$.

1. d: from in Section 2.3.1.1, d refers to the number of categories for the quantile-based indicator. As suggested in [12], the performance of the CUSUM monitoring schemes does not change much when $5 \leq d \leq 15$. In this article, d is set to be 10 for the numerical studies.

2. l: l should be smaller than m to avoid including unobserved data streams in $g(t)$. In practice, the exact number of true shifted data streams is often unknown. [21] suggests that a small l value is preferred in such cases, as it is robust in detecting a wide range of possible shifts.

3. k: from equations (11) and (14), it is clear that k works as an allowance that restarts the CUSUM procedure when there is no significant out-of-control evidence. In particular, k should be less than $\min_{1 \leq j \leq M} \left(C_{j,1}^{+}, C_{j,1}^{-}\right)$. Otherwise, the CUSUM procedure will be restarted at each time t. Based on existing literature [13], [14], in this paper we set $k = 0.5$.

4. ϵ: ϵ in equation (5) controls the exploration vs. exploitation tendency in the policy. Following existing RL practices [30], [31], we set $\epsilon = 1$ for the first $10\% - 20\%$ of the episodes, then decrease it linearly to $\epsilon = 0.01$ until the agent runs through $70\% - 80\%$ of the episodes, and fix it to $\epsilon = 0.01$ afterwards.

5. α: α is the step size of Stage-I Q-learning of NCQ algorithm (equation (3)). According to [32], a common choice is $\alpha = 0.01$.

6. γ: γ is the discount factor in equation (1). In our numerical studies, we set $\gamma = 0.99$ based on empirical results and existing practices.

7. n: n represents the number of categories for discretized local statistics. Observations indicate diminishing marginal benefits for increasing n from 4 to 10, with minimal performance changes beyond $n = 10$. This finding will be further demonstrated in Section 2.4.3.

8. ω: ω represents the number of actions that we perform in the BTS algorithm and typically set $\omega = m$ is sufficient as discussed in Section 2.3.1.3.

2.4 Simulations

In this section, we thoroughly evaluate the performance of the NCQ method and compare it with four state-of-the-art methods: NAS [9], QH01 [11], QUANTS [13], and random sampling (RS) method. RS employs the NCQ method with the same local and global statistics, but with randomly selected $O(t)$ instead of using the policy obtained through Q-learning. All benchmark methods are nonparametric as NAS and QH01 are anti-rank based and QUANTS and RS are quantile based. In addition, NAS, QH01, and QUANTS are all heuristic approaches that greedily reallocate resources onto the data streams with the largest local statistics at each time epoch and thus their sampling schemes all ignore the long-term reward. Note that the comparison with QH01 is slightly unfair as

it requires full observations at each time epoch, unlike the other methods that only observe partial information. Throughout this section, we consider in total $M = 6$ independent distributed data streams and only $m = 1, 2, 3$ observable data streams at each time. In addition, we assume the 6 data streams follow 3 distributions, with each distribution corresponding to 2 data streams. For demonstration purposes, these distributions include standard t-distribution with degree of freedom 3, standard normal distribution, and log-normal distribution with parameters $\mu = 1$ and $\sigma = 0.5$. Furthermore, we standardize each data stream to attain zero mean and unit variance. This standardization is crucial for ensuring fair comparability of constructed local statistics across all data streams. It establishes a uniform standard for constructing global statistics, particularly when selecting the top-l largest local statistics. Without standardization, streams with larger variance would consistently dominate the selection process. The number of categories for discretized local statistics is $n = 8$. We also test different magnitude of mean shifts $|\delta| = 1,2,3$. All results are based on 5000 replications and the in-control ARL_0 is set to be 370.

2.4.1 Monitoring Heterogeneous Data with Single Mean Shifts

In this subsection, we investigate the performance of detecting a single mean shift in heterogeneous data streams. To simulate out-of-control scenarios, we introduce a random mean shift to the in-control data, determined by the magnitude we aim to detect. For instance, in a manufacturing context defining an anomaly as a mean shift with a magnitude of 1, our goal is to detect shifts with at least this magnitude. Thus, during training, we randomly introduce a mean shift chosen from {1, -1} at a uniformly distributed time between $t = 150$ and $t = 250$. The shift is applied to a randomly selected data stream to derive the optimal policy for NCQ. NCQ is capable of detecting both positive and negative mean shifts. We demonstrate the online monitoring results for true single mean shifts with magnitudes of 1, 2, and 3, randomly chosen from {1, -1}, {2, -2},

and {3, -3}. The parameter settings for the five methods are as follows: $l = 1, k = 0.1$ for the NCQ, QUANTS and RS methods; $k = 0.1$ and compensation parameter of 0.02 for the NAS method; and $k = 0.05$ for the QH01 method. The out-of-control ARL, ARL_1 is summarized in Table 2.1, where values in parentheses represent the standard deviation.

Table 2.1 Performance Comparison of ARL_1 and the corresponding standard deviation (in parentheses) when heterogeneous data with single mean shift

m	$\|\delta\|$	NCQ	NAS	RS	QH01	QUANTS
1	3.0	14.06 (0.2)	>100	36.17 (0.3)	12.19 (0.1)	14.75 (0.2)
	2.0	15.28 (0.2)	>100	46.95 (0.4)	16.27 (0.1)	18.35 (0.2)
	1.0	24.31 (0.2)	>100	87.26 (0.8)	37.00 (0.2)	42.09 (0.5)
2	3.0	8.70 (0.1)	>100	31.05 (0.3)	12.19 (0.1)	7.71 (0.1)
	2.0	10.09 (0.1)	>100	37.64 (0.3)	16.27 (0.1)	9.79 (0.1)
	1.0	19.18 (0.2)	>100	70.09 (0.4)	37.00 (0.2)	26.33 (0.3)
3	3.0	7.67 (0.1)	>100	29.16 (0.1)	12.19 (0.1)	5.81 (0.1)
	2.0	9.10 (0.1)	>100	33.83 (0.2)	16.27 (0.1)	7.52 (0.1)
	1.0	18.22 (0.2)	>100	56.72 (0.3)	37.00 (0.2)	21.60 (0.2)

According to Table 2.1, we can make the following observations:

(i) NAS malfunctions due to strong heterogeneity in the data, violating the exchangeable assumption in the algorithm.

(ii) Larger shift magnitudes ($|\delta|$) lead to shorter detection delays for all methods. This is because larger $|\delta|$ values indicate more severe out-of-control scenarios, making them easier to notice and detect promptly. Additionally, increasing the number of observable data streams (m) reduces the detection delay for all methods, except for QH01, which assumes all data streams are observable at all times.

(iii) The advantage of NCQ over RS stems solely from using the Q-learning algorithm for sampling scheme selection, resulting in significant improvement across all combinations of m and $|\delta|$. We can then see the clear advantages of using Q-learning versus without Q-learning, while all of the local statistics remain the same for the two methods.

(iv) NCQ outperforms the QUANTS method in most cases, with a more pronounced advantage when $|\delta|$ and m are small. The superiority of NCQ is attributed to multiple factors. Firstly, during NCQ training, a mean shift of magnitude 1 is introduced, aligning with small mean shifts in the system. Secondly, when $|\delta|$ is small, heuristic approaches like QUANTS immediately prioritize monitoring data streams with larger local statistics, regardless of the actual values. Intuitively, if all local statistics are small, i.e., there is no strong evidence of anomaly, the optimal action is supposed to keep exploring data streams. NCQ, on the other hand, considers the cumulative future rewards and avoids premature exploitation when local statistics are still small. Similarly, when m is small, it becomes challenging to detect changes with limited resources, making it crucial to consider long-term rewards in resource allocation, as NCQ does. The comparison of NCQ with QUANTS clearly demonstrates the advantages of the learned policy through Q-learning over the current state-of-the-art sampling method. A detailed sampling layout analysis is studied in Section 2.4.2.

(v) When m and $|\delta|$ are large, the performance of NCQ and QUANTS becomes comparable. In such cases, process changes are generally easier to notice, resulting in shorter detection delays. Consequently, the advantage of considering long-term benefits in NCQ becomes less significant, while the QUANTS method's focus on monitoring the largest local statistics improves its performance.

(vi) Surprisingly, NCQ outperforms QH01 in most cases, despite QH01 requiring full observations at all times. Two possible reasons contribute to this result. Firstly, NCQ's emphasis on long-term benefits enables effective resource utilization and prompt anomaly detection even with limited resources. Secondly, NCQ's use of the top-l scheme, where only the largest l local statistics contribute to the global statistic $g(t)$, enhances detection sensitivity. In contrast, QH01 considers all M summands, which is less effective in single mean shift scenarios.

2.4.2 Sampling Layout Analysis

In this subsection, we analyze the sampling layouts of QUANTS and NCQ to gain a better understanding of the unique advantages and behaviors of our proposed method. For simplicity, we focus on the case where the mean shift (δ) occurs on data stream 1 with $m = 1$. We conducted 5000 simulated episodes and observed the sampling frequency of each data stream for both QUANTS and NCQ during the time epoch $T - 40$ to $T + 50$, where T refers to the time that mean shift takes place. Figs. 3 and 4 illustrate the sampling frequency of each data stream during the period $T - 40 < t < T$, i.e., the system is in control, for NCQ and QUANTS, respectively. Figs. 5 and 6 depict the sampling frequency of data stream 1 and the other normal data streams during the period T to $T + 50$, i.e., after process changes, for NCQ and QUANTS, respectively.

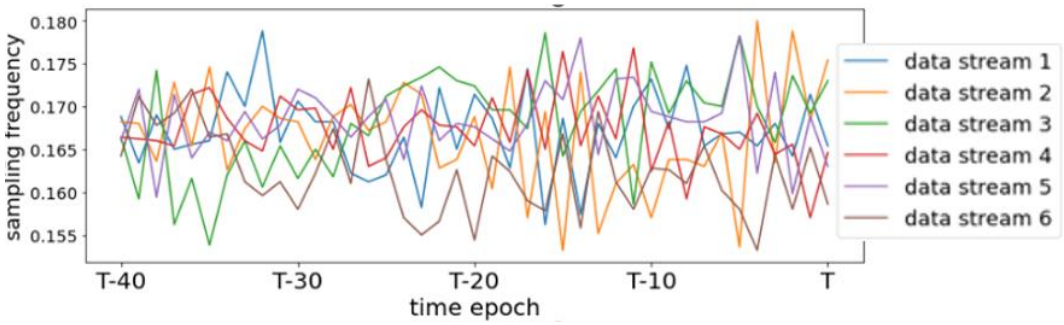


Fig. 2.3. Sampling frequency of data streams during in-control time epochs for NCQ.

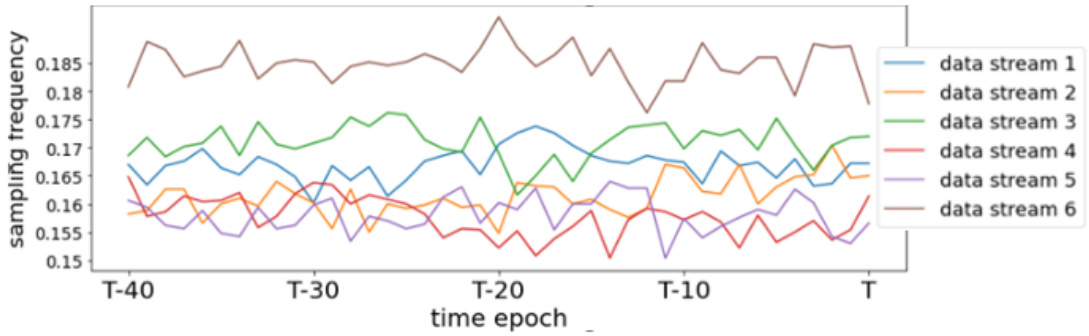


Fig. 2.4. Sampling frequency of data streams during in-control time epochs for QUANTS

In Fig. 2.3 and Fig. 2.4, it is evident that when the process is in control, NCQ samples each data stream approximately equally, while QUANTS exhibits uneven sampling frequencies. Fig. 2.5 demonstrates that when the process goes out of control and a mean shift occurs in data stream 1, the sampling frequency of NCQ increases at a faster rate compared to QUANTS, resulting in NCQ sampling data stream 1 nearly 80% of the time by $t = T + 50$. This highlights the advantage of NCQ in sampling the shifted data stream more frequently, enabling quicker anomaly detection. Additionally, Fig. 2.6 reveals that as time progresses during the out-of-control period, NCQ samples the other normal data streams less frequently compared to QUANTS, despite both methods having the same total resources available.

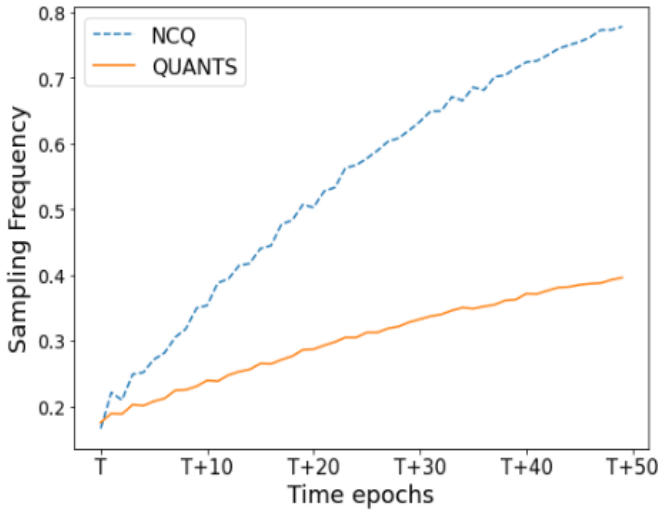


Fig. 2.5. Sampling frequency of data stream 1 during out-of-control time epochs

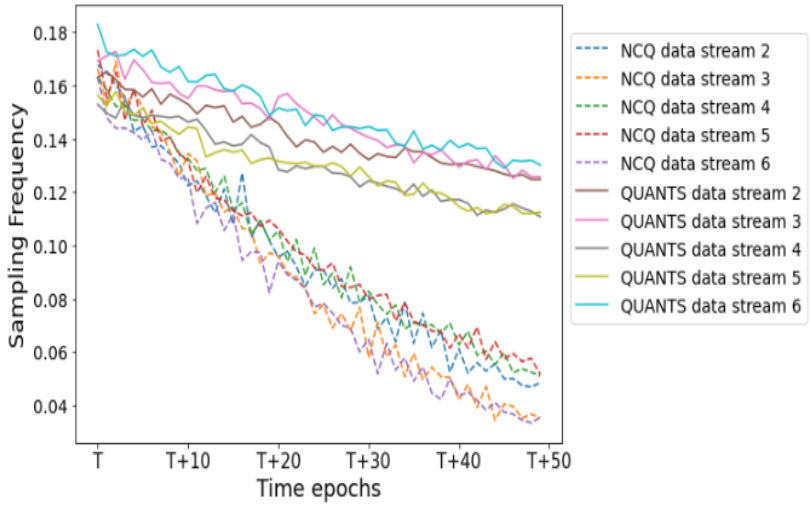


Fig. 2.6. Sampling frequency of other normal data streams during out-of-control time epochs

2.4.3 Choice of n

In this subsection, we investigate the effect of the hyperparameter n, which is a unique parameter in the proposed NCQ algorithm. Recall the parameter n represents the number of categories chosen for discretizing the local statistics. A larger n implies less information loss during the discretization process, while a smaller n results in a smaller state space size and improves the efficiency of learning the Q-table. We conducted simulation experiments using the settings from

Section 2.4.1, specifically testing the detection delay for $n = 4, 6, 8, and\ 10$, with $\delta = -1$ and $m = 1$. The results are displayed in Fig. 2.7. As we can see, the improvement in the detection delay decreases as n increases. When $n \geq 8$, the enhancement in the detection delay becomes marginal. This can be attributed to the fact that when n is already large, the discretized states contain sufficient information to effectively distinguish the level of suspicion for each data stream. Further increasing n becomes less beneficial while significantly increasing the computational complexity of the Q-table.

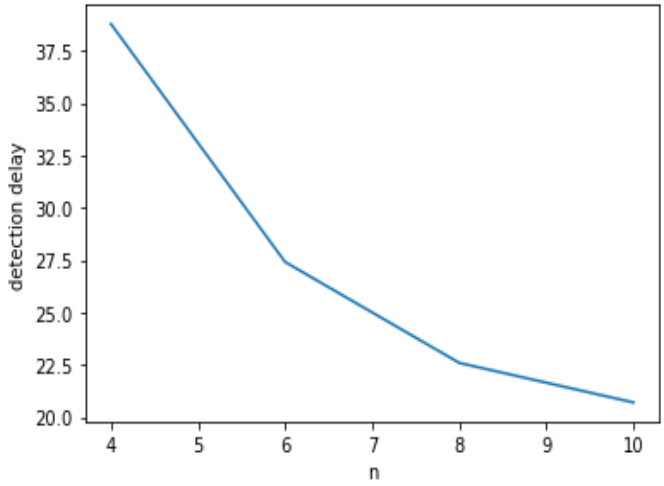


Fig. 2.7. Performance evaluations of NCQ with different hyperparameter n

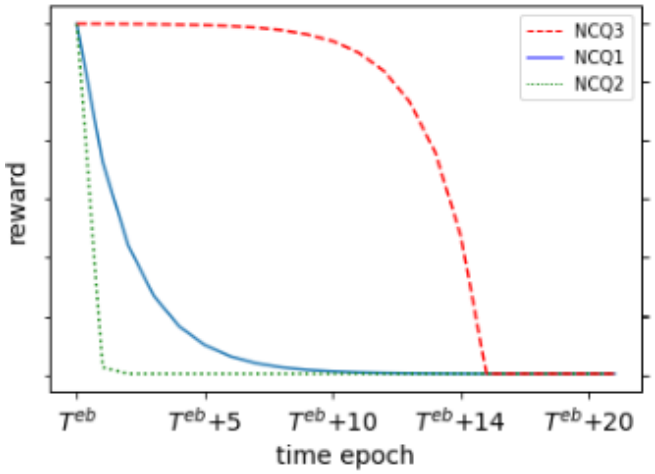


Fig. 2.8. Comparison of reward for transformed NCQ1 (solid), NCQ2 (dotted) and NCQ3 (dashed).

2.4.4 The Choice of Reward Assignment

In this subsection, we explore different reward assignment schemes. In Section 2.3.1.4, we proposed a smooth exponentially decayed reward function that can be easily modified to meet various detection requirements. For example, the reward can be redesigned to encourage greedier behavior of the agent. Specifically, we can assign $e^{(p-\Delta)/2}$ when the agent monitors the shifted data streams at $T + epoch_{buffer}$ as in Section 2.3.1.4, whereas assigning much smaller reward afterwards, e.g., $e^{(p-\Delta)/4}$ at $T + epoch_{buffer} + 1$, and $e^{(p-\Delta)/8}$ after time epoch $T + epoch_{buffer} + 2$, etc. We refer to this reward assignment scheme as NCQ2 for convenience. In this way, the agent cannot endure the delayed detection and behaves more greedily even when considering the long-term reward as a small detection delay will result in a significantly less reward and thus is no longer acceptable in NCQ2. As a result, the performance of NCQ2 will align more with the heuristic approaches such as QUANTS that has better advantage for detecting large shifts and when m is large. This congestion is validated in Table 2.2.

Unlike NCQ1 and NCQ2 where we assign an exponentially decayed reward function if the agent chooses to monitor the true shifted data streams, we can assign a constant reward of 10 when the agent monitors the true shifted data streams, but with an exponentially increasing punishment over time if the agent fails to monitor the shifted data streams. Specifically, the agent receives the same in-control reward at time $t = T + \Delta$, where $\Delta \in [0, epoch_{buffer})$, followed by exponentially increased punishment, $e^{(\Delta - epoch_{buffer})/2}$ where $\Delta \in \left[epoch_{buffer}, 16 + epoch_{buffer}\right]$. A fixed punishment $e^8$is assigned for $\Delta > 16 + epoch_{buffer}$. We refer to this reward assignment scheme

as NCQ3 with $epoch_{buffer} = 4$.

Table 2.2 Performance evaluation of ARL_1 for NCQ with different reward assignments

	$\|\delta\|$	NCQ1	NCQ2	NCQ3
m = 1	3.0	14.06	11.95	14.46
		(0.2)	(0.2)	(0.2)
	2.0	15.28	15.31	16.02
		(0.2)	(0.2)	(0.2)
	1.0	24.31	34.14	22.57
		(0.2)	(0.3)	(0.3)
m = 3	3.0	7.67	6.06	8.30
		(0.1)	(0.1)	(0.1)
	2.0	9.10	8.78	9.12
		(0.1)	(0.1)	(0.1)
	1.0	18.22	22.16	17.39
		(0.2)	(0.2)	(0.2)

For better comparison, we rescaled the reward values of NCQ1, NCQ2, and NCQ3 to have the same range when the agent monitors the true shifted data streams. Fig. 2.8 in shows a comparison of these three reward schemes after $t \geq T^{eb} = T + epoch_{buffer}$. NCQ3 (dashed line) has a slower decrease in reward compared to NCQ1 (solid line), indicating that the agent of NCQ3 receives a relatively consistent reward for a longer duration when monitoring the shifted data streams before it starts to decline rapidly. This suggests that the agent of NCQ3 behaves less greedily and performs better in detecting small shifts and when m is small. Conversely, NCQ2 (dotted line) drops more quickly than NCQ1, giving it an advantage in detecting large shifts and when m is large. This congestion is also validated in Table 2.2.

2.4.5 Monitoring Heterogeneous Data with multiple mean shifts

Recall that the optimal policy of NCQ in Section 2.4.1 was learned by assuming there is only a single mean shift in the system. However, in practice, multiple mean shifts may happen as well. In this subsection, we analyze the performance of NCQ in Section 2.4.1 while mean shifts occur in multiple data streams. For simplicity, we only consider the downward mean shift that is $\delta = -1, -2, -3$.

As we can see from Table 2.3, when NCQ in Section 2.4.1 is directly applied to multiple mean shifts scenarios, the detection delay uniformly decreases as the number of shifted data streams increases. Similarly, when the mean shift magnitude $|\delta|$ increases, or the number of observable data streams m increases, the detection delay becomes smaller as well. However, since the optimal policy of NCQ is derived based on the single mean shift with $|\delta| = 1$, the improvement in the detection delay becomes less significant as the mean shift magnitude $|\delta|$ increases or the number of shifted data streams increases. Generally, while NCQ is trained on a single mean shift scenario, it can still be directly applied and perform effectively in situations where simultaneous mean shifts occur in multiple data streams, even with a small size.

Table 2.3 Performance evaluations of ARL_1 and the corresponding standard deviation (in parentheses) when monitoring heterogeneous data with multiple mean shifts

m	δ	One data stream with a mean shift	Two data streams with mean shifts	Three data steams with mean shifts
1	-3.0	12.16 (0.2)	9.87 (0.1)	8.44 (0.1)
	-2.0	13.73 (0.2)	10.49 (0.1)	9.09 (0.1)
	-1.0	22.98 (0.3)	18.62 (0.2)	15.24 (0.2)
3	-3.0	6.91 (0.1)	5.74 (0.1)	5.18 (0.1)
	-2.0	8.25 (0.1)	6.72 (0.1)	6.07 (0.1)

-1.0	17.37 (0.2)	14.55 (0.2)	12.78 (0.2)

2.5 Case Study

In this section, we illustrate the real-world applicability of the proposed NCQ method through its implementation on the Tennessee Eastman Process (TEP), a scenario where the early detection of potential abnormalities is crucial. The TEP serves as an ideal testbed due to its status as a widely used dataset for algorithmic comparisons and benchmarks. Although the TEP is a computational model of an industrial process, it closely mimics real-world applications. The dataset is publicly available on Harvard Dataverse (https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/6C3JR1) and comprises 51 process variables (data streams) capturing various system properties, including flowrates, pressures, temperatures, levels, mole fractions, and compressor power outputs. The dataset includes both 'fault-free' and 'faulty' data files. The former represents the values of the TEP process during normal operation, while the latter incorporates 20 different simulated process faults. Notably, the dataset provides 250,000 labeled in-control training samples and 480,000 labeled in-control testing samples, contributing to a comprehensive evaluation of the proposed NCQ method in a real-world context. Out of the 51 data streams, we excluded those that are constant, as they are not meaningful for our analysis. To demonstrate our method's ability to monitor nonparametric data streams, we intentionally excluded those that follow normal distributions. This former selection was done through the Kolmogorov-Smirnov hypothesis tests at a significance level of 5% to verify whether in-control and out-of-control samples are from the same distribution. The latter selection was completed through D'Agostino-Pearson test that returns a p-value less than 0.05, which indicates that the underlying distribution of the data stream is not normal. Finally, we

selected data streams that have mean shifts in over 14 of the 20 fault types to ensure that the chosen data streams are indeed key indicators of the system out-of-control situations. This results in 10 remaining data streams for monitoring eventually. All selected data streams are then standardized before further analysis.

Table 2.4 Performance comparison of ARL_1 for NCQ and QUANTS based on the case study

	NCQ	QUANTS
$m = 1$	49.26	85.43
	(0.7)	(3.2)
$m = 2$	28.67	46.71
	(0.3)	(1.5)
$m = 3$	24.19	31.84
	(0.2)	(0.9)

Table 2.5 The average computational time of NCQ and QUANTS during online monitoring when calculating the updated sampling layout at each acquisition time

	NCQ	QUANTS
Time (sec)	1.73	0.28

To implement the NCQ algorithm, for offline training, we adopt the bootstrap procedure as discussed in Section 2.3.1.2 to sample with replacement from the 250000 in-control training samples to form the in-control data that consists of T time epochs in each episode, where T is uniformly distributed between 150 and 250. Then, to derive the optimal policy of NCQ, for each episode, a mean shift of either 1 or -1 is randomly added to one of the 10 data streams at time T. After the optimal policy is derived, we learn the threshold such that the in-control ARL is 370.

Next, for online monitoring, we substitute in-control training samples with in-control testing samples. The monitoring horizon is set as 5000 and the number of simulation runs is 5000. We consider $m = 1,2,3$ monitoring resources available in practice. The parameters are selected as follows: $l = 1, k = 0.5$ for the NCQ and QUANTS methods; and $n = 6$ for the NCQ method. The results of ARL_1 with different m are provided in Table 2.4 when a true mean shift of either 1 or -1 occurs at one of the 10 data streams during online monitoring. As we can see, NCQ outperforms QUANTS uniformly and the advantage of NCQ is more significant when there is a smaller number of monitoring resources, which is consistent with simulations in Section 2.4. The average computational time of NCQ and QUANTS during online monitoring when calculating the updated sampling layout at each acquisition time is presented in Table 2.5. The time gap results from looking for the state in the Q-table as well as the optimal action to take given the current state. However, the result shows that NCQ is still acceptable to be implemented in real time for most of the applications.

2.6 Conclusion

In this article, we have presented a novel framework for dynamic sampling and monitoring of heterogeneous processes with limited observations at each acquisition time. Our approach integrates Q-learning with a quantile-based nonparametric CUSUM procedure to construct local statistics and learn an optimal dynamic sampling strategy for quick anomaly detection. By summing up the top$-l$ local statistics, we obtain an effective global monitoring scheme that satisfies a prescribed in-control average run length (ARL). Compared to existing methods, our proposed NCQ algorithm offers several unique advantages: (i) NCQ considers long-term rewards and ensures that the dynamic sampling strategy asymptotically converges to the optimal policy

over time. This property is supported by the asymptotic convergence property of Q-learning [17]. (ii) NCQ naturally addresses the exploitation versus exploration trade-off without the need for heuristic compensations commonly used in other approaches (e.g., see [4]). (iii) NCQ tackles critical challenges in applying Q-learning to process monitoring by defining a systematic framework that incorporates partial observations, satisfies the Markov property, reduces the Q-table size, and improves detection performance using the BTS algorithm. The derived optimal policy together with the learned threshold helps define a dynamic sampling and monitoring strategy for quick mean shift detection. The superiority of the NCQ method has been demonstrated through simulations and a case study, where it outperformed state-of-the-art techniques, particularly in detecting challenging anomaly scenarios with small shift magnitudes and limited resources.

Several future research directions can be explored. Firstly, the application of NCQ can be extended beyond mean shift detection by incorporating practical costs into the reward function of RL, such as resource switching, late detection, and false alarm costs. Secondly, exploring data autocorrelation to relax the assumption of independent and identically distributed data streams would be beneficial. Third, when implementing the model online and encountering real process data with anomalous scenarios, continuously training the agent may help develop a policy more suited for actual anomaly situations. The extent and effectiveness of updating the Q-table in this context warrants further investigation. Finally, although BTS algorithm is proposed to improve the efficiency, NCQ still faces the scalability issue due to the slow process involved in learning the Q-table. Currently, the NCQ algorithm works best for no more than 15 data streams. The training time takes significantly longer when dealing with more data streams. In addition, when M is larger, n needs to be set to a smaller number to maintain a feasible size Q-table; therefore, it may

deteriorate the accuracy of the discretized local statistics and lead to a longer detection delay. One possible solution is to explore deep reinforcement learning with function approximation, enabling direct learning of the optimal policy based on continuous local statistics and free actions. This approach holds promise for handling a larger number of data streams, potentially reaching hundreds or thousands. We plan to explore these ideas in our future research endeavors.

2.7 Appendix A

Input: Prescribed in-control ARL and historical in-control data, learned Q-table and π^*, $num\_episode$, $in\_control\_epochs$, $threshold_{min}$, $threshold_{max}$, hyperparameter $l, d, n, k, \omega, \alpha, \gamma, \epsilon$, and percentile boundaries: $W_{1,j}^*, W_{2,j}^*, \dots, W_{n-1,j}^*$ for each data stream j

Output: *threshold* corresponding to the prescribed in-control ARL, ARL_0.

Initial setup:

Let $threshold = (threshold_{min} + threshold_{max})/2$, $\mu_0 = \infty$ and D be an empty list.

Algorithm:

While $\mu_0 \neq ARL_0$, repeat steps (I)-(III):

I. Let $t = 0, W_{j,0} = W_{j,0}^+ = W_{j,0}^- = 0, \boldsymbol{\alpha}_j = (1/d, 1/d, \dots, 1/d), j = 1, \dots, M.$ Randomly select m out of M data streams to create the initial state S_0. Bootstrap in-control data $in\_control\_epochs$ times.

II. For each time epoch $t = 1,2, \dots, in\_control\_epochs$:

a. Construct continuous and discretized local statistics for each data stream.

b. Order continuous statistics $\{W_{j,t}, j = 1, \dots, m\}$ for $j \in \mathcal{O}(t)$ from the largest to the smallest: $W_{(1),t} \geq \cdots \geq W_{(m),t}$ and calculate $g(t) = \sum_{j=1}^{l} W_{(j),t}$ as in (19).

c. If $g(t) < threshold$, obtain the current state S_t. Choose action A_t based on S_t following π^* and take action A_t. Implement the BTS algorithm for ω times. Proceed to time $t+1$ and collect new observations. Otherwise, raise an alarm, record the time epoch t and append it to D. Break the "For" loop and go to (III).

III. Repeat steps (i) and (ii) for $num\_episode$ times. Calculate the mean of D as μ_0. If $\mu_0 = ARL_0$, break the "While" loop. If $\mu_0 > ARL_0$, let $threshold_{max} = threshold$; Otherwise, let $threshold_{min} = threshold$. Update $threshold = (threshold_{min} + threshold_{max})/2$.

End

Fig. 2.9. The overall flow chart of NCQ Stage-II algorithm

Chapter 3 Online Monitoring of High-dimensional Data Streams with Deep Q-network

3.1 Introduction

The rapid development of modern sensor technology enables the possibility of real-time monitoring of vast measurements generated from diverse data streams. However, the substantial price associated with data acquisition, the limited bandwidth associated with data communication, and the limited processing capability restrict the full observability of the data streams during online monitoring. For instance, in healthcare surveillance, obtaining accurate real-time disease data during flu outbreaks is limited since it is impossible to conduct tests and receive results from every single person on a daily basis [33]. In some manufacturing processes, due to the limited processing capability and bandwidth constraint, only a fraction of data streams can be analyzed in real time for anomaly detection [1]. In energy systems, because of battery constraints, the sensors of power plants need to be turned on and off dynamically [2]. Thus, in all of these real-world problems, one can only observe selected partial data streams for decision-making. The research question is then how to adaptively determine the most informative data streams to monitor/observe and quickly detect anomalies in the system under resource constraints.

We frame the anomaly detection problem as follows. Assume there are M data streams and let $\boldsymbol{X}(t) = \big(X_1(t), \ldots, X_M(t)\big)$ denote the measurements of the M data streams at time epoch t. Each data stream can follow an arbitrary distribution. At time epoch t, only the observations of m out of M data streams can be acquired due to resource constraints. Define $\mathcal{O}(t)$ and $\mathcal{U}(t)$ to be the sets of observable and unobservable data streams at time epoch t. Then $\mathcal{O}(t) \cup \mathcal{U}(t) = \{1, 2, \ldots, M\}$,

$\mathcal{O}(t) \bigcap \mathcal{U}(t) = \emptyset$, and $|\mathcal{O}(t)| = m$, where $\emptyset$ denotes the empty set and $|\cdot|$ denotes the cardinality of a set. Besides, for $j = 1, \ldots, M$, we define $H_{j,t}$ as the time duration that data stream j has not been observed up to time epoch t, referred to as the *unattended time*. Below are our detailed assumptions: (i) By pre-centering and pre-scaling historical in-control data, we assume that the mean and standard deviation of $X_j(t), j = 1, \ldots, M$, are 0 and 1 when the process is in control. (ii) The process becomes out of control at some unknown time T, and the mean $\boldsymbol{\mu} = E(\boldsymbol{X}(t))$ shifts from $\mathbf{0}$ to an unknown value. Moreover, the number of shifted data streams is also unknown. (iii) $\boldsymbol{X}(t)$ is independent and identically distributed (i.i.d.) over time during either the in-control or the out-of-control process. (iv) The sets $\mathcal{O}(t)$ and $\mathcal{U}(t)$ can be dynamically updated at each time epoch t without incurring extra cost. (v) There are sufficient historical in-control data for offline training purposes. The research goal is to propose a generic monitoring scheme to decide which subset of data streams to observe at each acquisition time such that the mean shifts in high dimensional data streams can be detected as early as possible while maintaining a required system-wide in-control average run length (ARL), ARL_0.

In theory, online monitoring with partial observations can be interpreted as a problem that sequentially determines the occurrence of anomalies and identifies which data streams to observe at each acquisition time. Thus, this problem inherently incorporates the framework of Partially Observable Markov Decision Process (POMDP). However, due to the high-dimensional data streams and the unknown underlying distributions, the transition probability is challenging to be accurately described. To this end, conventional methods that focusing on solving POMDP cannot be directly applied [27]. This unique challenge motivates us to employ deep reinforcement learning (DRL) as a powerful tool to resolve the POMDP problem since DRL can well deal with unknown transition probabilities and complex underlying models. Furthermore, DRL considers the long-

term discounted rewards associated with the actions (i.e., which data streams to observe) while existing approaches [4], [9], [13], [14] derives the sampling strategy based on heuristic rules. In this way, DRL effectively balances between exploiting suspicious data streams and exploring potential mean shifts from all data streams. Although DRL has been used for anomaly detection in some literature [34]–[38], they mainly focus on diagnostic accuracy and anomaly localization instead of online monitoring and dynamically sampling of the most informative data streams. These facts then motivate us to develop a new DRL framework tailored to the online monitoring of high dimensional processes with partial observations. To the best of our knowledge, this is the first work that aims to integrate DRL into online monitoring of high-dimensional data streams given resource constraints. To develop an effective DRL framework, several challenging questions must be addressed: (i) How to effectively define the state and action spaces for the online monitoring problem with partial observations? It is crucial to ensure the state and action spaces satisfy the Markov property, which is the key underlying assumption of DRL. (ii) How to design an appropriate neural network structure given that DRL is typically hard to converge? (iii) How to establish an intelligent reward framework to minimize detection delay? We aim to bridge the research gap by answering these questions.

In this paper, we propose a DRL-based adaptive sampling and monitoring framework. We name this algorithm the Statistical Deep Q-network algorithm (denoted as the "SDQ" algorithm). Specifically, SDQ first transforms raw observations of each data stream into a nonparametric local statistic that indicates the likelihood of anomaly occurrences associated with each data stream. The nonparametric statistics and the unattended time form a state representation in the DRL framework. Then, a double-dueling extension of a carefully designed deep Q-network is established. This extension enhances the convergence speed of the algorithm and enables the

learning of an optimal dynamic sampling policy. At last, following the optimal policy suggested by the deep Q-network, we construct the global statistics using the sum of top-l local statistics to indicate whether the system is out of control. Consequently, the proposed SDQ algorithm has the following advantages: (i) Different from all previous algorithms [6][13][14] which act myopically and only focus on the short-term rewards of sampling strategy, SDQ considers long-term discounted rewards and directly incorporates the unattended time into decision making. In this way, SDQ better utilizes available information for smarter decisions. (ii) SDQ naturally handles the exploitation versus exploration trade-off, negating the need for any heuristic compensations to encourage exploration, as was emphasized in previous works [4], [41], [42]. (iii) The use of deep neural network structure addresses scalability issues and allows for the monitoring of hundreds of data streams.

The rest of this paper is organized as follows. In Section 3.2, we review the literature on DRL and multivariate process monitoring with partial information. In Section 3.3, we introduce the technical details of the SDQ algorithm. In Section 3.4, we conduct simulations to test the performance of different neural network architecture designs and parameter settings, and evaluate the proposed method by comparing it with benchmark methods. In Section 3.5, we conduct a case study on the Tennessee Eastman Process to further demonstrate the superiority of the SDQ algorithm. Finally, in Section 3.6, we provide concluding remarks and future research directions.

3.2 Literature Review

In Section 3.2.1, we review the basic framework of RL and DRL. Specifically, in Section 3.2.1.1 and Section 3.2.1.2, we introduce the fundamental concepts of reinforcement learning (RL) and Q-learning and discuss their extension to deep Q-learning. Then in Section 3.2.1.3, we examine

the application of DRL in anomaly detection, highlighting how DRL can improve anomaly detection in various fields. In Section 3.2.2, we review methods for online monitoring with partial observations.

3.2.1 RL and DRL

3.2.1.1 RL and Q-learning

The central idea of RL is to develop a deterministic or stochastic strategy, also called the ***policy***, based on the immediate reward and cumulative weighted rewards, also known as the ***value***, through the interaction with the environment. At each time epoch t, the agent is in some state S_t, facing n possible actions to take $A_t^1, A_t^2, \cdots, A_t^n$. Each action A_t^i will bring back an immediate reward R_t^i, and send the agent to a new state S_{t+1} under disturbance from the environment. A ***reward*** R_t indicates the performance level of the agent at time epoch t. The ***value*** is the discounted cumulative reward defined as $G_t \stackrel{\text{def}}{=} R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=1}^{\infty} \gamma^{k-1} R_{t+k}$, where γ is a discount rate. There are two types of value functions in RL: the state-value function $v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$ defined as the discounted reward starting from state s and following policy π afterward, and the action-value function $q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a]$ defined as the discounted return if we start from state s, take action a, and follow policy π afterward. A ***policy*** $\pi(a|s) = P[A = a | S = s]$ is a probability distribution of taking a certain action given a state and it is constantly adjusted by the agent based on the reward feedback from the environment. A policy π outperforms another π' if $v_\pi(s) \geq v_{\pi'}(s)$ for any state s. The agent aims to find the optimal policy that is better than or equal to any other feasible policies. Specifically, the optimal policy identifies the values $v^*(s), q^*(s,a)$ such that $q^*(s,a) = \max_\pi q_\pi(s,a)$ and $v^*(s) = \max_\pi v_\pi(s) = \max_a q^*(s,a)$.

Among all RL algorithms, Q-learning is especially powerful when the environment is complex and transition probabilities are unknown. Q-learning iteratively improves the policy π based on the action-value function $q_\pi(s,a)$. Specifically, given the initial policy and state, we repeatedly evaluate the value function following the specified policy and then find the optimal policy corresponding to the updated value functions, until the policy converges to the optimal one: $\pi_0 \xrightarrow{\text{Evaluate}} q_{\pi_0} \xrightarrow{\text{Improve}} \pi_1 \xrightarrow{\text{Evaluate}} q_{\pi_1} \xrightarrow{\text{Improve}} \cdots \xrightarrow{\text{Improve}} \pi^* \xrightarrow{\text{Evaluate}} q^*(s)$. For value evaluation, we implement the target policy which is a greedy search:

$$\pi(s) = \text{argmax}_a q_\pi(s,a), \tag{1}$$

Besides, for policy improvement, we implement the behavior policy, typically an ϵ-greedy search:

$$\pi(a|s) = \begin{cases} 1-\epsilon, & a^* = \pi(s) \\ \epsilon, & a^* = \text{Unif}(a \in A|s)\text{'} \end{cases} \tag{2}$$

where $a^* = \text{Unif}(a \in A|s)$ stands for a uniformly random selection of a feasible action given state s, and ϵ is a small positive value. In other words, with probability $1-\epsilon$ the agent selects the action that has the highest Q-value, and with probability ϵ the agent chooses a random feasible action. As a result, Q-learning is often regarded as an off-policy algorithm because it estimates the value function for state-action pairs based on a target policy, even though it actually follows a behavior policy.

3.2.1.2 Deep Q-network

For many problems, it is impractical to represent the Q-values as a table containing values for each state and action pair (s,a) giving the large state space and action space. Instead, we train a neural network with parameters $\boldsymbol{\theta}$ to parametrize and predict $Q(s,a)$. We will use $\boldsymbol{\theta}_i$ to refer to

the neural network parameters, and use s, a, r, s' to represent the realization of S_i, A_i, R_i, S_{i+1} at training step i. We can then minimize the following loss at each step:

$$L_i(\boldsymbol{\theta}_i) = \mathbb{E}_{s,a,r,s' \sim \rho(.)}[(y_i - Q(s, a; \boldsymbol{\theta}_i))^2] \quad (3)$$

where $y_i = r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\theta}_i)$ is called the temporal difference (TD) target and $y_i - Q(s, a; \boldsymbol{\theta}_i)$ is called the TD error. Each set of $\{s, a, r, s'\}$ is referred to as an *experience* and ρ represents the behavior distribution over experiences.

The final key component of the deep Q-network is the experience replay buffer. At each time step, we add the experience $\{s, a, r, s'\}$ to a circular buffer, which is called the replay buffer. During the training, instead of using only the latest experience $\{s, a, r, s'\}$ to compute the loss in (3) and update the parameters $\boldsymbol{\theta}_i$ accordingly, we sample a mini batch of experiences from the replay buffer and then compute the loss. In this way, we achieve better data efficiency by using each experience in multiple updates and better stability as the experiences in each batch are uncorrelated [43].

3.2.1.3 Anomaly Detection with Deep Reinforcement Learning

The use of DRL for anomaly detection has been increasingly embraced across a range of fields. For instance, DRL has been applied to detect and localize abnormalities in computer vision videos [34][35]. In the domain of cloud intrusion detection, DRL is used as a vital security solution to monitor and detect malicious activities in cloud-based applications, and to provide defense against cyber-attacks [10][11][22]. In organizations, Network Intrusion Detection Systems (NIDS) play a crucial role in ensuring the safety and security of communication and information. DRL has also been widely employed in NIDS, as demonstrated by [38], [45], and [46]. However, resource constraints often limit the ability to observe all data streams of interest in real time and none of the

above mentioned research considers resource limitations. [47] addressed this challenge by proposing a DRL-based algorithm that assumes an unknown number of abnormal processes at any given time, while the agent can only check with one sensor in each sampling step. Another common limitation of the existing DRL methods is that most of them focuses on diagnostic accuracy and anomaly localization, paying little attention to online monitoring and quick anomaly detection. [48] addressed this issue by proposing a low-latency detection approach that incorporates the trade-off between detection delays and detection accuracy based on a reward function. Nevertheless, this approach assumes full observations of state transitions and data, overlooking the challenges posed by heterogeneous data streams with partial observations. In summary, there is a pressing need for a generic DRL framework tailored to the online monitoring problem. This framework should enable quick anomaly detection in heterogeneously distributed high-dimensional data streams when only partial observations are available in real time due to resource constraints.

3.2.2 Online Monitoring with Partial Information

In order to monitor high-dimensional data streams, many existing literature use dimension reduction methods to simplify the feature space [27][28]. However, these works assume full observations of the data streams are available at each acquisition time and do not consider the resource constraints. To address this issue, [41] proposed a Bayesian framework by combining Thompson Sampling for sensor selection and a signal decomposition method for high-dimensional data. However, it is possible that the reconstructed feature space is not able to fully represent the original system and there is information loss compared to directly utilizing all observed data. Another approach in literature relies on constructing local statistics for each data stream, with heuristic compensation applied to unobserved data streams to encourage exploration. For example,

[4] proposed the first adaptive sampling algorithm for partially observed data streams with Gaussian distribution, and [9] extended the work to non-Gaussian but exchangeable processes. Later [14] further extended the method to monitor data streams with arbitrary distributions. However, all these works utilize heuristic approaches and do not take advantage of the unattended time. In other words, these existing works are not capable of tracking the time that a data stream has not been monitored and making intelligent decisions based on this additional information. In summary, the existing methods neither fully utilize the underlying structure of the online monitoring problem nor leverage the power of DRL. To fill the research gap, we propose a statistical deep Q-network algorithm in this paper.

3.3 Methodology

While it is promising to develop the DRL-based method for online monitoring of high-dimensional data streams, there are several unique challenges. First, we need to define an appropriate state space that can fully characterize the status of the system. We also need to decide on a reward assignment scheme for practical use. Second, incorporating decisions regarding which data streams to observe and whether to raise an alarm into the action space will inevitably pose significant computational challenges and potentially decrease detection speed. Therefore, we propose to separate the two decision processes and let DRL only determine the sampling strategy on which data streams to observe at each time. We then rely on the SPC technique to effectively decide whether to raise an alarm based on a global monitoring statistic. One advantage of this approach is that we can separate the training courses of DRL and SPC. Third, deep Q-network often fails to converge to the actual Q-value function $Q^*(s, a)$. To address this issue, we adopt the double extension [51] and the dueling extension [52] to boost computational effectiveness.

By tackling the above challenges, we propose the SDQ algorithm, which contains two parts: offline training and online implementation. For the offline training part, there are two stages: In Stage-I, we first construct a nonparametric local statistic for each data stream. The state representation of SDQ is formed based on the local statistics and the unattended time, to fully characterize the status of the system. Then a double dueling deep Q-network (DQN) is trained to parametrize the Q-value function and a sampling policy π^* can be derived accordingly. In Stage-II, following the learned policy, the threshold for raising an alarm is determined such that the in-control ARL equals a prespecified value. During online implementation, we apply policy π^* and the learned threshold for real-time process monitoring. Note that there are many existing methods to construct the nonparametric local statistics [4], [9], [13], [42] and some of them even consider the spatial structure of the data streams [39]. In this paper, we aim to develop a generic framework where various local statistics can be simply employed. In particular, we specify the types of nonparametric local statistics that can be incorporated into the proposed SDQ algorithm, as long as the following three properties are satisfied: (i) A larger value of the local statistic suggests that an anomaly is more likely to have occurred. (ii) When the process is in control, the value of the local statistic fluctuates, and when the system is out of control, the value continuously increases. (iii) The local statistic is memoryless, i.e., the future local statistic values only depend on the current value and are independent of the past statistic values. Indeed, there are many nonparametric local statistics available in the existing literature that satisfy the three properties above [].

The technique details of the SDQ algorithm are presented below. In Section 3.3.1, we introduce how to define the state and action space, the reward function, and the neural network design. Then the DRL framework is trained based on simulations. In Section 3.3.2, we construct a global statistic and select an appropriate threshold corresponding to a prespecified in-control ARL. After these

preparations, real-time online monitoring can then be conducted. In Section 3.3.3, the practical guidelines on the parameter settings of the SDQ algorithm are provided.

3.3.1 Stage-I SDQ algorithm

In this subsection, we initially define the state space, action space, and reward structure of the DRL framework. Following this, we detail the selection of an appropriate neural network architecture tailored to our specific problem. Subsequently, we delve into the intricacies of offline training methodologies. Lastly, we offer an exhaustive overview of the Stage-I SDQ algorithms.

3.3.1.1 State and Action Space

The state space should contain as much useful information about the current system status as possible, and yet has a simple form. In particular, we propose to construct the state space that consists of two pieces of information for each data stream j: the local statistics $W_{j,t}$ of the current time epoch t that represents the likelihood of a mean shift in the data stream j, and the unattended time $H_{j,t}$, which represents how long the data stream j has not been observed up to time epoch t. In this way, the state of data stream j at time t can be described as $(W_{j,t}, H_{j,t})$. Since we have in total of M data streams, the state of the whole system can be expressed as a $2 \times M$ matrix $S_t = \begin{pmatrix} \boldsymbol{W}_t \\ \boldsymbol{H}_t \end{pmatrix} = \begin{pmatrix} W_{1,t}, W_{2,t}, \ldots, W_{M,t} \\ H_{1,t}, H_{2,t}, \ldots, H_{M,t} \end{pmatrix}$. It is worth mentioning that including the information of unattended time in state representation is necessary for a smarter sampling policy. For example, consider two data streams j_1 and j_2, where $W_{j_1,t} = W_{j_2,t}$ while $H_{j_1,t} > H_{j_2,t}$. Then it is more reasonable to observe j_1 than j_2 since j_1 has not been observed for a longer time and a mean shift might already take place in j_1.

We then consider how to update the state. Given the definition of $H_{j,t}$, if we do not observe data stream j at time t, then $H_{j,t}$ should be incremented by one. Otherwise, if data stream j is being observed at time t, $H_{j,t}$ should be reset to 0. Thus, $H_{j,t}$ can be updated as

$$H_{j,t} = \begin{cases} 0 & \text{if } j \in \mathcal{O}(t) \\ H_{j,t-1} + 1 & \text{if } j \in \mathcal{U}(t) \end{cases}. \tag{4}$$

Besides, the update of the local statistic $W_{j,t}$ depends on the specific type of nonparametric local statistics employed by the practitioners. Without loss of generality, in this paper, we use the quantile-based statistic proposed in [7] as an illustration, where the details are provided in Chapter 2.

In our problem formulation, two decisions need to be made at each time epoch: which data streams to observe, and whether to raise an alarm. As mentioned above, incorporating both decisions into the action space will inevitably increase its cardinality, and potentially reduce the detection speed of the algorithm. Therefore, we train a DRL agent that only learns the dynamic sampling strategy, leaving the decision of when to raise an alarm to the Stage-II SDQ algorithm (i.e., the monitoring framework) in Section 3.3.2. To decide the observable streams, we define the action space as follows: each possible action is a free selection of m out of M data streams for monitoring, which leads to in total of $\binom{M}{m}$ possible actions. Specifically, we define $\boldsymbol{A}_t = (A_{1,t}, A_{2,t}, \dots, A_{M,t})$, where $A_{j,t} = \mathbb{I}\big(j \in \mathcal{O}(t)\big)$, and $\sum_{j=1}^{M} A_{j,t} = m$. Finally, we examine the Markov property associated with state and action space. Given the current state $S_t = \binom{\boldsymbol{W}_t}{\boldsymbol{H}_t}$ and the action A_t, $\boldsymbol{W}_{t+1}$ is purely determined by $\boldsymbol{W}_t$ and the new observations at $t+1$. Similarly, $\boldsymbol{H}_{t+1}$ only depends on $\boldsymbol{H}_t$ and $\boldsymbol{A}_t$. Thus, the Markov Property is satisfied.

3.3.1.2 Reward

In this section, we discuss the reward assignment during the offline training. Note that the agent should not have any preference in selecting the data stream to observe when the process is in control. This is because if any of the data streams is favored, it leads to a larger detection delay when a mean shift takes place in the unfavored ones. Consequently, we assign a constant negative reward r_1, such as -1, to suppress any uneven preference. Note that here we must have a negative reward, otherwise a positive one will accumulate and provoke the agent's inclination to choose the same data streams, causing a vicious cycle for training.

Additionally, when the system goes out of control at time T, we assign a large reward that decays exponentially as time elapses when the agent chooses to observe the shifted data streams, hence encouraging the agent to detect the anomaly as quickly as possible. Note that an exponentially decayed reward is one of the most common choices when designing the reward in DRL [34], [47], [53], [54]. However, one unique challenge here is that the local statistics take time to accumulate even if the shifted data stream is constantly being monitored, especially when the mean shift is small and just occurs. Thus, we introduce another hyperparameter λ such that the agent receives the same in control reward during the time interval $[T, T + \lambda - 1]$. Based on our numerical studies, we suggest to set $\lambda = 4$ by default and adjust its value accordingly when the mean shift is very large or very small. After a certain amount of time p, the decayed reward might be too small. Therefore, to avoid this issue and simplify the calculation, we assign a constant positive value r_2, such as 1 if the agent monitors the shifted data streams after p time epochs. In this way, if the agent monitors the shifted data streams when $t \geq T$:

$$R = \begin{cases} r_1 & t < T + \lambda \\ q \times \max\left(r_2, e^{\frac{p-(t-T)}{2}}\right) & T + \lambda \leq t \leq T + p \\ q \times r_2 & t > T + p \end{cases}, \quad (5)$$

where q is the number of shifted data streams that the agent chooses to monitor. Based on empirical results, setting $p = 16$ performs well enough. Meanwhile, if the agent chooses to monitor unshifted data streams, we assign a constant negative reward r_3 such as -10 to punish the agent regardless of the time.

3.3.1.3 Deep Neural Network

In most modern applications, DRL often faces the issue that the value function parametrized by the neural network is not accurate and stable enough, i.e., the TD error discussed in Section 3.2.1.2 does not converge. Without an accurate estimation of the value functions, the policy learned by DRL is hardly effective. We resolve this problem by adopting the double extension [51] and dueling extension [52] of deep Q-network, and carefully selecting the neural network architecture.

First, a common problem deep Q-network possesses is that it overestimates the Q-values [51]. Recall that $y_i = r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\theta}_i)$ is the TD target at each iteration i and the best action is often assumed to be the action corresponding to the highest Q-value. However, this is not necessarily true, especially at the early stage of training during which we do not have enough information about what action the agent has tried, and the neighboring states the agent has explored. To address this issue, [51] proposed the double deep Q-network which uses two separate networks with the same architecture to decouple the action selected from the target value estimation. Specifically, we construct two identical neural networks: the primary network and the target network. A graphical representation of the double architecture can be found in Fig. 3.1. Specifically, the target network selects the optimal action to take, and the primary network

calculates the Q-value of taking that action at that state. The weights of the primary network, $\boldsymbol{\theta}^{\text{primary}}$, get updated every $update\_primary$ time epochs following the loss function in (3), while the target network's weights, $\boldsymbol{\theta}^{\text{target}}$, is copied from $\boldsymbol{\theta}^{\text{primary}}$ every $update\_target$ time epochs to reduce the overestimation, where $update\_target > update\_primary$. We will further discuss the selection of $update\_primary$ and $update\_target$ in Section III.C. As a result, the loss in (3) becomes

$$L_i\left(\boldsymbol{\theta}^{\text{primary}}\right) = \mathbb{E}_{s,a,r,s'\sim\rho(.)}[(R + \gamma \max_{a'} Q(s', a'; \boldsymbol{\theta}^{\text{target}}) - Q\left(s, a; \boldsymbol{\theta}^{\text{primary}}\right)^2]. \quad (6)$$

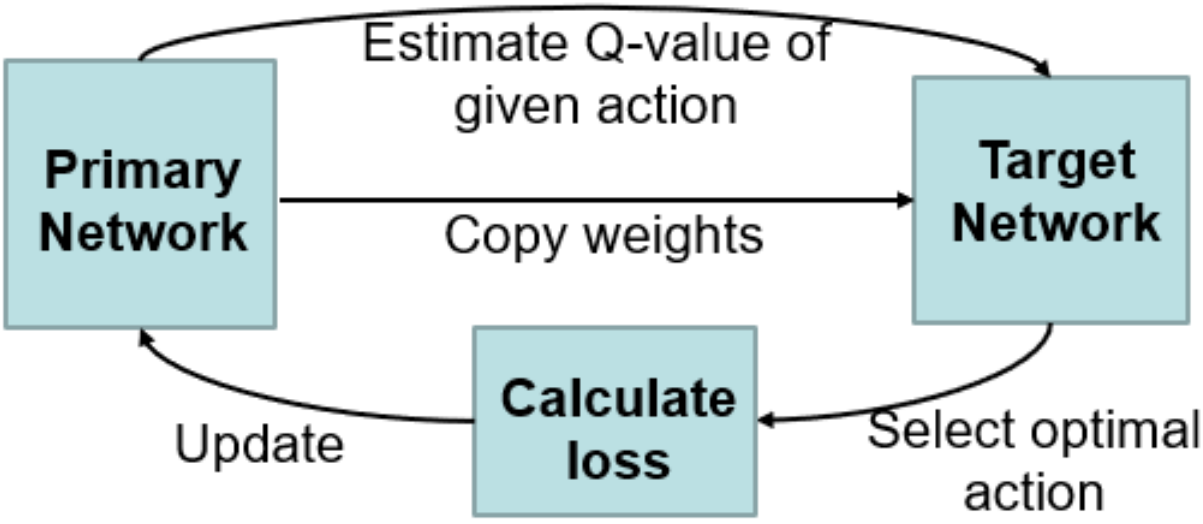


Fig. 3.1. Graphical representation of double architecture.

Second, updating the value of each action at every time step is unnecessary, especially when the cardinality of the action space is large, as it affects learning efficiency and stability. In the context of online monitoring, when the system is in control, the agent receives a constant reward regardless of the state and action and thus it is not necessary to learn which action to take until the occurrence of an anomaly. To address this issue, [52] proposed the dueling Q-network that can learn which states are valuable without learning the effect of each action for each state. Specifically, dueling

Q-network splits the deep Q-network into two separate streams, the state-value stream, parametrized by $V(s;\theta)$ and the action-value stream, parametrized by $A(s,a;\theta)$. The state-value stream learns the state-value function $v(s)$, which indicates how good it is to be in a particular state s regardless of the action taken. The action-value stream describes how much better or worse each action is compared to the average action in that state. Then the key idea of the dueling Q-network is to use the average value of all possible actions ($\frac{1}{|A|}\sum_{a'} A(s,a';\boldsymbol{\theta})$) as a baseline to estimate the Q-value:

$$Q(s,a;\boldsymbol{\theta}) = V(s;\boldsymbol{\theta}) + A(s,a;\boldsymbol{\theta}) - \frac{1}{|A|}\sum_{a'} A(s,a';\boldsymbol{\theta})\,. \tag{7}$$

A graphical representation of the dueling architecture can be found in Fig. 3.2.

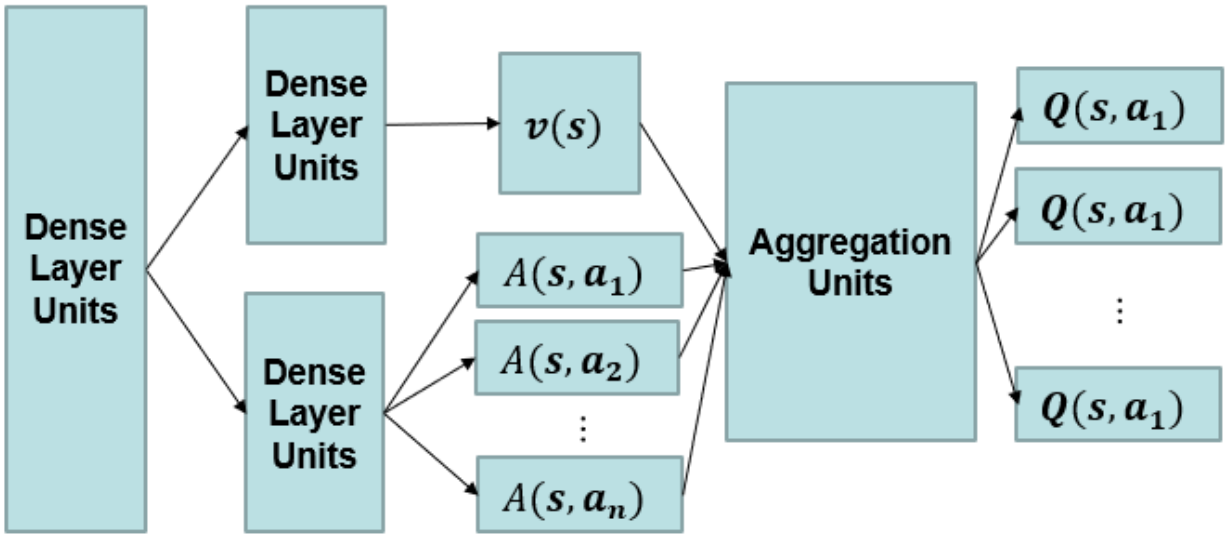


Fig. 3.2. Graphical representation of dueling architecture.

Note that to integrate double deep Q-network and dueling deep Q-network, both the prime network and the target network adopt the dueling structure. Besides, to ensure efficient learning, we create a mask so that only the Q-values corresponding to the sampled experiences (s,a,r,s') are being updated based on the loss function (6). Lastly, we also implement batch normalization,

which is a common technique applied in deep learning that accelerates the learning process. In Section 3.4, we present the simulation results that compare the vanilla deep Q-network with its double extension and the double dueling extension. The effects of varying neural network sizes are also studied in Section 3.4.

3.3.1.4 Offline training of DQN

A challenge that DRL frameworks often face is the lack of training data. In the problem of online monitoring, DRL frameworks require sufficient both in-control and out-of-control data such that we can explore different reward scenarios. To address this challenge, we adopt the bootstrap technique in existing literature to generate the in-control training data [4], [9], [13]. In particular, for each training episode, we bootstrap the available historical in-control data T^0 times as the in-control observations. Besides, it is common in practice that the out-of-control historical data is very sparse or even does not exist. Therefore, to simulate the out-of-control data in each training episode, we sample T^1 historical observations and introduce a mean shift to a randomly selected data stream. The magnitude of the mean shift introduced can be set as the minimum magnitude of mean shift that we are interested in detecting. As a result, each training episode has a total of $T^0 + T^1$ observations (time epochs). Based on empirical studies, we found it is sufficient to let $T^0 \sim \text{Unif}(100,200)$ and $T^1 = 150$ to ensure a stable and efficient training procedure, where $\text{Unif}(\cdot)$ is the uniform distribution. Finally, recall that an ϵ-greedy search is implemented for policy improvement. Since the agent is expected to engage in more exploration at the beginning of the training and gradually shift towards exploitation as the training iteration progresses, we define epi_{explore} such that the agent will purely explore the possible actions, i.e., $\epsilon = 1$, for the first epi_{explore} episodes, and further define epi_{greedy} such that after epi_{greedy} training episodes, the

agent will mainly exploit the estimated value function, i.e., $\epsilon = 0.1$. Otherwise, ϵ decreases linearly as training episode increases. In this way, ϵ can be expressed as

$$\epsilon = \begin{cases} 1, & \text{if } episode \leq epi_{\text{explore}} \\ 0.1, & \text{if } episode > epi_{\text{greedy}} \\ 1 - 0.9 * \dfrac{episode - epi_{\text{explore}}}{epi_{\text{greedy}} - epi_{\text{explore}}}, & \text{otherwise} \end{cases}. \quad (8)$$

As it typically takes some time for the local statistics to become stable at the beginning of each training episode, we introduce $warm\_up\_epoch$ such that we discard the data collected from the first $warm\_up\_epoch$ epochs in each training episode to ensure a more efficient and stable training procedure. Based on empirical results, we set it to be 60. In addition, we train the agent for fixed epi_{train}episodes. Empirically we found that setting $epi_{num} = 10^5$ is sufficient. We let α and γ denote the step size and the decay parameter in DRL, where the selection of these parameters is discussed in Section 3.3.

3.3.1.5 Overview of Stage-I SDQ algorithm

Now we have addressed all aforementioned challenges. Specifically, we defined a state space, which contains information of the unattended time that previous literature overlooked. Such information promotes smarter decisions on which data streams to observe. The defined state representation also satisfies the Markov property, which is the key underlying assumption of DRL. Secondly, we introduced a tailored reward assignment scheme, which encourages exploration when the process is in control and exploitation when the mean shift occurs. Thirdly, to make the trained DQN more accurate and stable, the vanilla deep Q-network is extended to a double dueling deep Q-network. Furthermore, some special designs of the neural network architecture are also employed to further boost convergence and accuracy, such as the mask and batch normalization.

An overview of the Stage-I SDQ algorithm is summarized in Fig. 3.3.

Input: historical in-control data, hyperparameter α, γ, δ, p, λ, $warm\_up\_epoch$, $epi_{num}, epi_{\text{explore}}$, and epi_{greedy}, $update\_primary$, $update\_target$, $batch\_size$.

Output: Learned optimal deep Q-network and corresponding policy π^*.

Initial setup: Randomly initialize the deep Q-network and the Target network. Initiate an empty replay buffer.

Algorithm:

For $episode = 1, \dots, epi_{num}$, repeat steps (I)-(III):

V. Sample $T^0 \sim U(100,200)$ and set $T^1 = 150$. Bootstrap in-control data T^0 times and then bootstrap additional in-control data T^1 times.

VI. Set $t = 0$, and ϵ based on (8). Randomly select a data stream as the shifted data stream and generate the out-of-control data by inserting mean shift δ to the data stream.

VII. For each time epoch $t = 1, 2, \dots, T^0 + T^1$, repeat steps (a)-(f):

a. Construct the local statistic for each data streams and obtain the current state $\boldsymbol{S}_t$.

b. Feed $\boldsymbol{S}_t$ into the target network and derive the optimal action. Take action A_t based on ϵ-greedy policy as in (8). Calculate reward R_t according to Section 3.3.1.2. Collect new observations of data streams corresponding to A_t.

c. If $t \leq warm\_up\_epoch$, go to next time epoch.

d. Store experience (s, a, r, s') into the replay buffer. If the current replay buffer size is greater than max_memory, remove the first experience in the replay buffer.

e. For every $update\_primary$ time epoch, sample $batch\_size$ experiences from the replay buffer, calculate the Q-value based on (7) using the primary network, and compute loss as in (6). Perform gradient descent.

f. For every $update\_target$ time epochs, copy the weights of the current deep Q-network to the target network.

End

Fig. 3.3 Overview of the Stage-I SDQ algorithm

3.3.2 Stage-II SDQ algorithm

In this subsection, we propose a monitoring procedure that decides when to raise an alarm. Specifically, we construct a global statistic to indicate whether a mean shift takes place in any data streams and compare it with a threshold H, which is chosen to achieve a pre-specified in-control ARL. To do so, given the local statistics $W_{j,t}$ of each observed data stream (i.e., for data stream j in set $\mathcal{O}(t)$) at time epoch t, we sort them in descending order. Mathematically, we let $W_{(1),t} \geq W_{(2),t} \geq \cdots \geq W_{(m),t}$ denoted the sorted local statistics. Then the global monitoring statistic can be constructed as the sum of the top l local statistics:

$$g(t) = \sum_{j=1}^{l} W_{(j),t}. \tag{9}$$

Note that we only sort the local statistics for the m observed data streams because the local statistics for unobserved data streams may not be accurate, especially when the data streams are not observed for a long time. Here, l is a hyperparameter, and its selection is discussed in Section 3.3.3. During the online monitoring, we raise an alarm if this global statistic is larger than the threshold H. In other words, the stopping time is defined as $\tau = \inf\{t \geq 1: g(t) \geq H\}$. The advantages of constructing this global statistic are twofold. First, as mentioned in Section 3.3.1.1, separating the decision to raise an alarm from the DRL framework reduces the complexity of the

action space. Second, the global statistic can detect a wide range of possible shifts in all directions without knowing any potential shift pattern.

During online monitoring, the reward assignment is no longer needed since we have already learned the optimal policy π^*. We simply keep updating the local statistics of both observed and unobserved data streams to form a state, raise an alarm when $g(t)$ is larger than the threshold H. Otherwise, we feed the state into the DQN and take the optimal action following learned π^*, and then proceed to the next acquisition time. Thus, the main computational complexity lies in two parts. The first part is the calculation of M local statistics $W_{j,t}$. Since the local statistics can be calculated recursively, the memory usage will not grow as time goes by. The second part is the search for the action with the maximum value according to the deep Q-network, which can be done in constant linear time for fixed number of data streams. Therefore, the online implementation of SDQ runs efficiently in real time.

In practice, the threshold H can be learned by bootstrap. Specifically, we generate in-control data again by bootstrapping the historical data. Then for a given value of threshold H, we can perform the online monitoring by following the learned optimal policy π^*, which takes the action with the largest value according to the deep Q-network. In this way, the in-control ARL can be estimated. Therefore, we can adjust the threshold H such that the actual in-control ARL equals a pre-specified value ARL_0.

3.3.3 Parameter Setting

In this subsection, we discuss how to set the parameters involved in the SDQ algorithm. We examine the following hyperparameters: l, α, γ, *batch_size*, *update_primary*, *update_target*, *max_memory*, epi_{explore}, epi_{greedy}, and $warm\_up\_epoch$.

1. l: l is the number of aggregated local statistics in (9). In practice, l should be smaller than m to prevent the inclusion of unobserved data streams in the calculation of $g(t)$. Ideally, l should be set as the number of shifted data streams. However, the exact number of shifted data streams is often unknown. In such cases, [21] suggests that a small l value is preferred as it is more robust in detecting a wide range of possible shifts.

2. α: α is the step size when training the DQN in Stage-I SDQ algorithm based on the gradient descent of the loss function (6). According to [32], a common choice of α is 0.01.

3. γ: γ is the discount factor in (3). In our numerical studies, we set $\gamma = 0.99$ based on empirical results and existing practices.

4. $batch\_size$: Each time we train the deep Q-network, we sample a batch of past experiences from the replay buffer to update the network weights. The batch size used in DRL is greater than 1 and less than the size of the training set. We set it to be 20 based on empirical results and existing practices.

5. $update\_primary$: We record the new experience (s, a, s', r) at each time epoch. However, the primary network is only updated every $update\_primary$ time epochs. This approach enhances training stability and efficiency by minimizing the overall frequency of weight updates in the neural network. Based on empirical results and established practices, we set $update\_primary$ to 4.

6. $update\_target$: As discussed in Section 3.3.1.3, the double Q-network involves concurrent training of the DQN network and periodic copying of its weights to the target network every $update\_target$ epoch. The influence of this parameter on monitoring performance will be explored in Section 3.4.3.

7. max_memory: In DRL, the replay buffer needs to have a proper memory length to make sure there are enough past experiences to sample from, and the past experiences should not be too outdated. Consequently, a good choice of max_memory is important, and we further investigate the selection of max_memory in Section 3.4.3.

8. epi_{explore}: In DRL, it is common to allow the agent to completely explore the states and actions at the initial stage of the training process, i.e., set $\epsilon = 1$. Typically, the agent spends 10% of the entire training episodes in purely exploring phase. Since we set $epi_{\text{num}} = 10^5$, we let $epi_{\text{explore}} = 10000$.

9. epi_{greedy}: In DRL, at the final stage of the training procedure, the agent should already know its task well and thus should only exploit what it has learned, i.e., set $\epsilon = 0.1$. This setting leads to the final convergence of the optimal policy and the parametrized value functions. To make sure the agent has enough time to exploit, it is common to allow the agent to have 30% of the entire time in this purely exploiting phase for the DRL framework. Since we set $epi_{\text{num}} = 10^5$, we let $epi_{\text{greedy}} = 30000$.

10. $warm\_up\_epoch$: Recall that we introduced this hyperparameter in Section 3.3.1.4 since the local statistics are not stable at the beginning of each training episode. Besides, we have also explained that when the process is in control, there is not much valuable information that the agent needs to learn, which suggests that discarding more time epochs when the process is in control might be helpful. Therefore, we will further study the effect of this parameter on the monitoring performance of the proposed method in Section 3.4.3.

3.4 Simulations

In this section, we thoroughly evaluate the performance of the SDQ algorithm. To make the construction of the state space complete, we adopt the quantile-based local statistics in [7] due to its simplicity and effectiveness in detecting mean shifts (please see Appendix A for more details). In Section IV.A, we first demonstrate the advantages of utilizing the double dueling extension of the deep Q-network by comparing the performance of a vanilla deep Q-network, a double Q-network, and a double dueling Q-network, all under the same hyperparameters. In Section IV.B, we contrast the performance of different neural network architectures, including four schemes with different network widths and depths. In Section IV.C, we investigate the effects of hyperparameters related to deep Q-network. We then demonstrate the advantage of including the unattended time information, $\boldsymbol{H}_t$, in Section IV.D. With the final optimal architecture, we compare its performance with several benchmark methods including the QH01 method [38], the QUANTS method [7], and the Random Sampling method in Section IV.E. In Section IV.F, we further deep dive the effectiveness of the SDQ algorithm and in Section IV.G, we test the power of SQD in monitoring multiple shifted data streams. All results are based on 5000 simulation replications and the prescribed ARL_0 is set to be 370.

3.4.1 The Vanilla Deep Q-network and Its Extensions

In this subsection, we demonstrate the advantages of the double dueling Deep Q-network extension by comparing its performance to the vanilla Deep Q-network and the double Deep Q-network extension. Specifically, we assume there are 15 independent data streams with 4 being observable at each time epoch, i.e., $M = 15$ and $m = 4$. Without loss of generality, we assume the 15 data streams follow 5 distributions, with each distribution corresponding to 3 data streams.

These distributions include standard normal distribution, standard t distribution with degree of freedom 3, exponential distribution with $\mu = 1$, log-normal distribution with $\mu = 1$ and $\sigma = 0.5$, and poison distribution with $\lambda = 10$. Then, each data stream is standardized to achieve zero mean and unit variance when the process is in control. To conduct offline training, we employ the bootstrap method as discussed in Section III.A.4 by randomly sampling $T^0 + T^1$ samples with replacement from the historical in-control data in each training episode, where $T^0 \sim U(100,200)$ and $T^1 = 150$. The first T^0 samples are used as the in-control training data, while a mean shift of 1 is randomly inserted to one of the 15 data streams in the last T^1 samples. Then the deep Q-network is trained according to Fig. 3.3 and the optimal policy can be obtained accordingly. With the learned policy, the monitoring threshold of the global statistic is determined such that the in-control ARL is 370. To evaluate the effectiveness of the proposed algorithm during the online monitoring, we repeat the aforementioned simulation procedures. Note that although the SDQ is able to detect both positive and negative mean shifts, here for simplification and demonstration purposes, we only examine the positive mean shift detection and illustrate SDQ's capability of negative mean shift detection in Section IV.E. In this subsection, we fix $update\_target = 1000$ and $warm\_up\_epoch = 60$. Besides, we evaluate the deep Q-networks and the extensions with max_memory set at both 10^5 and 1000. This approach is taken because variations in max_memory can much affect performance. The ARL_1 values of the three network structures are summarized in Table 3.1, where the values in the parentheses represent the standard deviation, and the best results for each scenario are in bold.

Table 3.1 Performance evaluations of ARL_1 for SDQ with different DQN

	Vanilla	Double	Double Dueling
$max\_memory = 10^5$	73.51	44.25	**39.37**

	(2.41)	(1.67)	**(0.98)**
$max\_memory = 1000$	69.23	43.17	**38.54**
	(2.37)	(1.65)	**(1.02)**

As we can see in both cases, the performance of double Q-network is significantly better than the vanilla deep Q-network. This demonstrates the advantage of using two separate networks to separate action selection from Q-value estimation, thereby avoiding the overestimation of Q-values. In addition, the separation of the state-value stream and the action-value stream in the dueling architecture also brings additional benefits. This indicates that the double dueling structure of the deep Q-network outperforms its counterparts.

3.4.2 Neural Network Architectures

In this subsection, we compare the performance of different neural networks with different depths and widths. Specifically, we apply the double dueling deep Q-network and adopt all the settings in Section IV.A. In particular, the following four network structures in Table 3.2 are considered. The numerical values represent the number of neurons in each hidden layer, where the leftmost layer is the one directly connects with the input layer. For example, in NN_{large} configuration, there are three hidden layers with the number of neurons set to 256, 256, and 512, respectively. Again, we test the performance of SDQ when $max\_memory = 10^5$ and $max\_memory = 1000$. The results of ARL_1 is shown in Table 3.3, where the values in the parentheses represent the standard deviation.

Table 3.2 Neural Networks

	Architecture
NN_{regular}	256 – 256

NN_{large}	256 – 256 – 512
NN_{medium}	128 – 256 – 256
NN_{deep}	128 – 256 – 256 – 256

Table 3.3. Performance evaluations of ARL_1 for SDQ with different networks

	NN_{regular}	NN_{large}	NN_{medium}	NN_{deep}
$max\_memory = 10^5$	44.25 (1.67)	**40.25** **(1.58)**	44.03 (1.66)	46.31 (1.64)
$max\_memory = 10^3$	43.17 (1.65)	**39.03** **(1.51)**	44.60 (1.57)	44.81 (1.63)

From Table. 3.3, we observe that different neural network structures yield different performances. Among them, NN_{large} consistently outperforms the rest. Thus, we will use NN_{large} in the following studies.

3.4.3 SDQ Hyperparameter Selection

In this subsection, we test the effect of the following three hyperparameters that are crucial and unique in SDQ algorithm: $update\_target$, max_memory, and $warm\_up\_epoch$. We have also investigated the influence of $update\_primary$ and $batch\_size$, which are also unique parameters in SDQ algorithm. However, these two parameters do not have much impact on the monitoring performance, thus the results are omitted in this paper. Again, the simulation settings in Section IV.A are adopted. Besides, we use the double deep Q-network with structure NN_{large} according to the previous subsections. The results of different values of max_memory, $update\_target$,

and $warm\_up\_epoch$ are shown in Table 3.4, Table 3.5, and Table 3.6, where the values in the parentheses represent the standard deviation.

Table 3.4. ARL_1 for SDQ with $warm\_up\_epoch = 60$, $update\_target = 10^4$, and different max_memory.

	5000	1000	300	100	20
ARL_1	43.20 (1.04)	41.08 (1.01)	**39.16 (0.98)**	40.22 (1.02)	51.49 (1.27)

Table 3.5. ARL_1 for SDQ with $max\_memory = 300$, $update\_target = 10^4$, and different $warm\_up\_epoch$.

	0	30	60	100
ARL_1	44.86 (1.09)	41.23 (1.04)	**39.60 (0.84)**	40.54 (0.98)

Table 3.6. ARL_1 for SDQ with $max\_memory = 300$, $warm\_up\_epoch = 60$, and different $update\_target$.

	1	5000	10^4	1.5×10^4	2×10^4
ARL_1	62.33 (2.31)	44.09 (0.70)	39.60 (0.84)	**36.24 (1.03)**	42.27 (1.05)

From Table 3.4, we find that with $warm\_up\_epoch$ and $update\_target$ fixed, the performance of SDQ improves when max_memory decreases until around 300 and then degrades when max_memory further decreases. This suggests that when the replay buffer size is too large or too small, the learning is not efficient. In particular, when max_memory is too large, the sampled

experiences may be from a long time ago, which are not helpful for the agent to learn the sampling strategy, and when max_memory is too small, the sampled experiences may not be independent and thus make the learning process biased. From Table 3.5, we observe that since local statistics are not stable at the start of each training episode, discarding approximately 60 epochs is helpful. However, further increasing the $warm\_up\_epoch$ may deteriorate the performance of the SDQ algorithm, as it results in the discarding of some useful data. From Table 3.6, we can see that letting $update\_target = update\_primary = 4$ is equivalent to the vanilla deep Q-network without the double structure which performs poorly. In addition, the ARL_1 performance is improved by increasing $update\_target$ up to 1.5×10^4, but then degrades if $update\_target$ is further increased. This is because a delayed update of the target network can lead to a significant disparity between the estimated value of taking an action and the actual value, resulting in misleading and erroneous updates. Overall, we select the hyperparameters for the following sections: $max\_memory = 300$, $warm\_up\_epoch = 60$, and $update\_target = 1.5 \times 10^4$. However, it is worth noting that in general, the SDQ algorithm is insensitive to hyperparameter selections as long as they remain in a reasonable range.

3.4.4 The Necessity of Unattended Time

In this subsection, we demonstrate the necessity of including the unattended time information, $\boldsymbol{H}_t$, by comparing the SDQ algorithm with SDQ-II, where the unattended time is excluded from the state space, while applying the same reinforcement learning algorithm. Specifically, at time t, the state of data stream j is denoted as $W_{j,t}$. With a total of M data streams, the state of the entire system can be represented as a $1 \times M$ vector $S_t = (\boldsymbol{W}_t) = (W_{1,t},\ W_{2,t}, \dots, W_{M,t})$. The rest of the setup follows the SDQ algorithm. Both SDQ and SDQ-II utilize a double dueling Q-network, with

the optimal neural network design and hyperparameters as detailed in Sections IV.B and IV.C. We consider a total of 15 data streams, as described in Section IV.A, with$m = 2$ observable data streams for demonstration. The result of ARL_1 is shown in Table 3.7. As shown, the inclusion of unattended time information leads to a significant improvement, as the detection delay ARL_1, drastically decreases in all cases.

Table 3.7. Performance evaluations of ARL_1 for SDQ with different networks

	$\delta = 1$	$\delta = -1$	$\delta = 3$	$\delta = -3$
SDQ	57.26	56.98	30.91	30.18
	(1.24)	(1.24)	(0.97)	(0.97)
$SDQ - II$	104.63	**99.07**	39.50	38.64
	(4.12)	**(4.14)**	(1.06)	(1.3)

3.4.5 Performance Comparison of the State-of-the-Art Methods

In this subsection, we thoroughly evaluate the performance of the SDQ algorithm and compare it with the state-of-the-art methods: the QH01 method, the QUANTS method, and the RS (random sampling) method. Here the SDQ algorithm follows a double dueling deep Q-network with the optimal design of the neural network and hyperparameters discussed in Sections 3.4.2 and 3.4.3. The QH01 method assumes all observations are available and thus it is unfair to directly compare QH01 with our SDQ algorithm. However, QH01 serves as a good benchmark for assessing the detection performance when all data streams can be observed. The QUANTS method follows the exact same procedure for constructing local statistics as SDQ, but it implements a heuristic sampling strategy rather than relying on the RL approach. The RS method is also based on the same local statistics as SDQ but at each time epoch, it randomly selects m data streams to observe. Note that the QH01 method is based on the anti-rank and all the other algorithms are based on the

same quantile-based approach to construct local statistics. Thus, all the benchmark methods considered here are nonparametric. In addition, both QH01 and QUANTS methods are heuristic approaches and are not capable of utilizing the information of unattended time for decision making. The parameters for the four methods are selected as follows: l = 1, $k = 0.5$ for the SDQ, QUANTS, and RS methods, and k = 0.05 for the QH01 method. We test the ARL_1 results when the true mean shift takes value of 1, 2, 3, -1, -2, and -3 and occurs in a data stream. We still consider a total of 15 data streams distributed as in Section IV.A, and with $m = 2, 4, 6$ observable data streams at each time. Please note that when training the SDQ algorithm, we still learn the optimal policy based on only a mean shift of 1 that is randomly inserted to one of the 15 data streams, as in Section IV.A, and then we evaluate SDQ's performance based on different mean shift scenarios during online monitoring. The result of ARL_1 is shown in Table 3.8.

Table 3.8. Performance comparisons of ARL_1 when monitoring heterogeneous data with a single mean shift

	δ	SDQ	RS	QH01	QUANTS
m = 2	3	30.91 (0.97)	>100	16.32 (0.81)	45.65 (1.28)
	2	39.52 (1.02)	>100	21.72 (1.12)	57.64 (1.47)
	1	57.26 (1.24)	>100	73.74 (2.36)	107.94 (2.15)
	-1	56.98 (1.24)	>100	70.33 (2.32)	103.17 (2.06)
	-2	39.27	>100	20.46	54.21

		(1.01)		(1.09)	(1.43)
	-3	30.18 (0.97)	>100	15.75 (0.81)	45.83 (1.36)
m = 4	3	19.70 (0.84)	58.49 (0.37)	16.32 (0.81)	22.67 (0.30)
	2	22.66 (0.89)	74.08 (0.46)	21.72 (1.12)	27.77 (0.76)
	1	36.24 (1.03)	>100	73.74 (2.36)	63.46 (1.24)
	-1	35.66 (1.02)	>100	70.33 (2.32)	61.74 (1.18)
	-2	21.79 (0.87)	74.88 (0.46)	20.46 (1.09)	25.79 (0.78)
	-3	19.41 (0.84)	58.43 (0.37)	15.75 (0.81)	21.83 (0.28)
m = 6	3	13.06 (0.73)	38.61 (0.32)	16.32 (0.81)	10.40 (0.26)
	2	15.29 (0.77)	43.80 (0.33)	21.72 (1.12)	15.00 (0.32)
	1	27.46 (0.96)	>100	73.74 (2.36)	45.77 (0.75)
	-1	27.39 (0.94)	>100	70.33 (2.32)	44.60 (0.73)

-2	15.27	43.83	20.46	13.98
	(0.77)	(0.33)	(1.09)	(0.38)
-3	13.07	38.55	15.75	10.56
	(0.73)	(0.32)	(0.81)	(0.25)

We can make the following observations from Table 3.8:

(i) When the magnitude of the mean shift magnitude becomes larger or the number of available resources m becomes higher, the detection delay is shorter for all methods. This result is expected as a larger $|\delta|$ value is easier to be noticeable in general. Similarly, more information becomes available when m increases. Thus, the detection delay becomes smaller as we have better knowledge about the system status, except for QH01 which assumes all data streams are observable all the time.

(ii) The advantage of SDQ over RS purely results from employing the DRL for deciding the sampling scheme. In all combinations of m and δ, there is a significant improvement by SDQ, indicating the clear advantage of our proposed dynamic sampling strategy.

(iii) We also find that SDQ outperforms QH01 in some cases. We provide two possible reasons. First, SDQ focuses on the long-term benefits as described above. So even when only limited resources are available, SDQ can still effectively utilize the available resources to decide the appropriate sampling scheme for quick anomaly detection. Second, SDQ uses the top-l scheme, i.e., only the largest l local statistics are used to form the global statistic $g(t)$, whereas QH01 takes all M summands. This improves the detection sensitivity and reduces detection delay.

(iv) The proposed SDQ algorithm outperforms the QUANTS method in most cases, and the advantage of SDQ is more significant when $|\delta|$ and m are small. The first possible reason is that when training SDQ, we introduced a mean shift of magnitude 1. Therefore, when the system indeed

has a mean shift that is close to 1, SDQ should perform the best. The second reason is that SDQ also incorporates and utilizes the information of unattended time, while QUANTS cannot consider this important information. Lastly, heuristic approaches such as QUANTS will immediately reallocate resources to monitor whichever data streams that have larger local statistics. Intuitively, if there is no strong evidence of anomaly, i.e., all local statistics are small especially when $|\delta|$ is small, the optimal action is supposed to keep exploring data streams. SDQ tackles this problem by focusing on the optimal behavior in the long term, and thus it can avoid getting into exploitation early when local statistics are still small. Similarly, when m is small, it becomes very challenging to notice the changes with limited resources and thus it is more important to consider the long-term consequence of the sampling strategy.

(v) When m and $|\delta|$ are both large, the performance of SDQ and QUANTS become comparable. This is because in this case, the process change is easy to be noticed in general (i.e., the detection delay is expected to be very small). As a result, it is less beneficial to consider the long-term rewards, and thus the heuristic approach used by QUANTS can also perform well.

3.4.6 Sampling Strategy Analysis

In this subsection, we illustrate the effectiveness of the proposed SDQ algorithm in leveraging information from unattended time to strike a balance between exploration and exploitation, thus resulting in reduced detection delays. Specifically, for the case when $m = 4$ and $\delta = 1$ in Section IV.E, we record the values of $H_{j,t}$ and $W_{j,t}$ whenever a data stream is not observed. Then we compare the sampling strategy of SDQ with RS and QUANTS by visualizing the combinations of $H_{j,t}$ and $W_{j,t}$ values collected during the implementation of each method.

When the process is in control, Fig. 3.4. shows the scatter plot of $H_{j,t}$ and $W_{j,t}$ for unobserved data streams using SDQ and RS. Similarly, the plot of $H_{j,t}$ and $W_{j,t}$ for comparing SDQ and QUANTS is shown in Fig. 3.5. From Fig. 3.4., we observe that the scatter points of SDQ are clustered towards the lower left corner of the figure. This matches our expectation that when SDQ does not observe a data stream, it is due to either a low local statistic or short unattended time. However, the scatter points of RS are widely dispersed because RS makes random decisions on which data stream to observe. Consequently, it is possible for data streams with a large local statistic and a long unattended time to remain unobserved by the RS algorithm. Additionally, when comparing SDQ and QUANTS in Fig. 3.5, it is evident that some data streams with a large unattended time are not selected by QUANTS for observation. This is because QUANTS chooses observable data streams based on local statistics alone, failing to effectively utilize the information from unattended times. As a result, when the process goes out of control, it is highly probable that some data streams with mean shifts remain unobserved by QUANTS for extended periods, leading to a large detection delay.

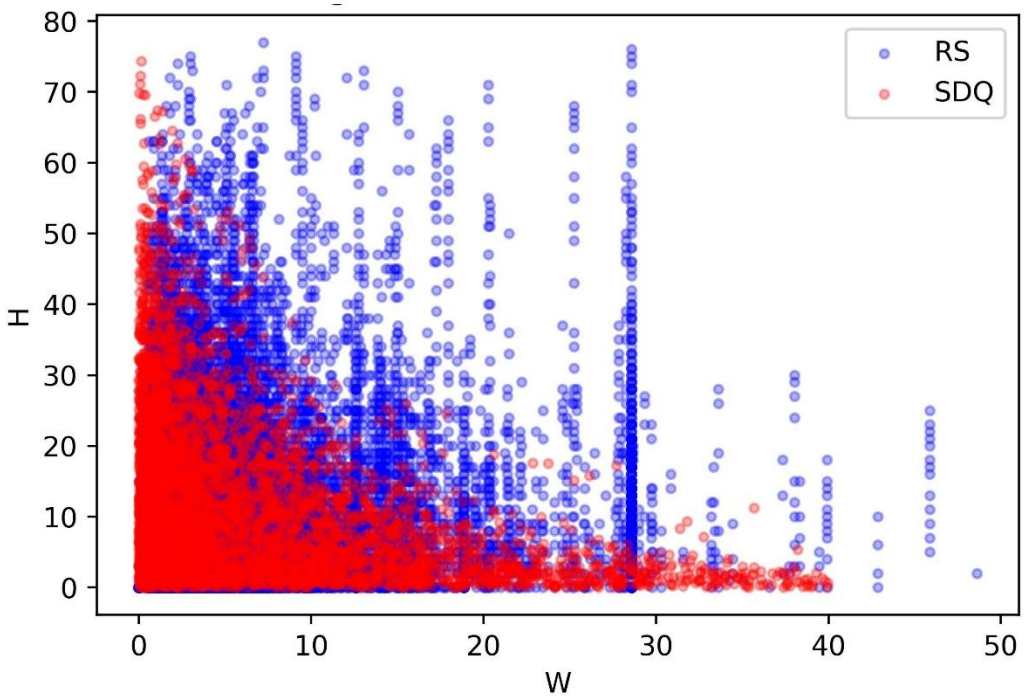


Fig. 3.4. Scatter plot of $H_{j,t}$ and $W_{j,t}$ for unobserved data streams by SDQ and RS when the process is in control.

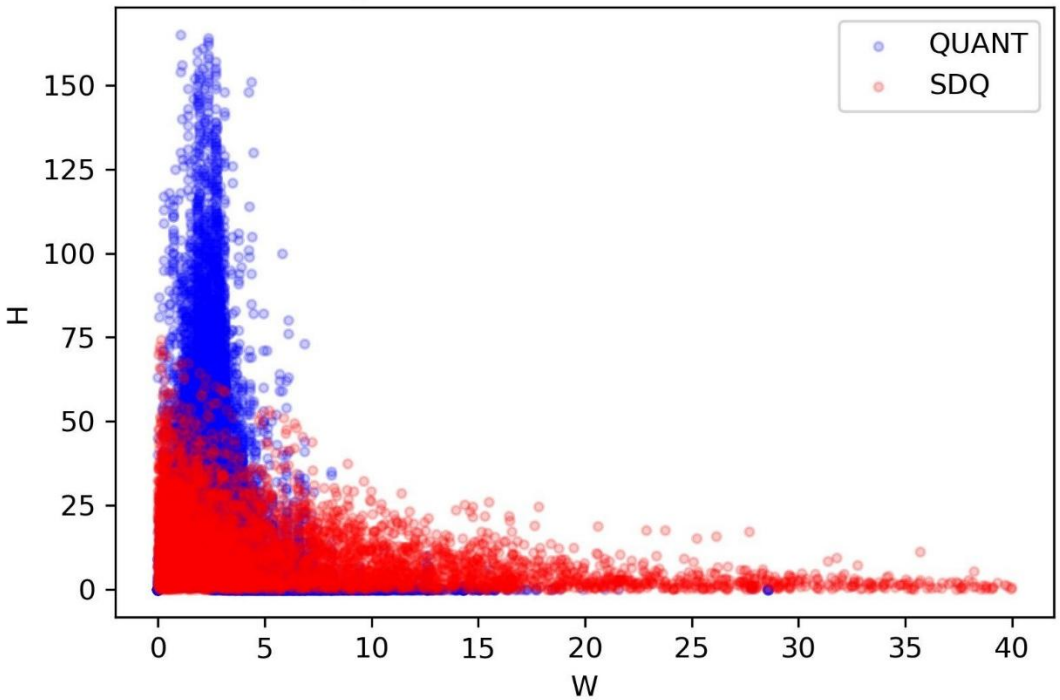


Fig. 3.5. Scatter plot of $H_{j,t}$ and $W_{j,t}$ for unobserved data streams by SDQ and QUANTS when the process is in control.

3.4.7 Monitoring Heterogeneous Data with multiple shifted data streams

In Section IV.E, we initially established the optimal policy for SDQ under the premise that a single mean shift occurs in the system. However, real-world applications often encounter multiple shifted data streams, necessitating further evaluation of SDQ's efficacy under such conditions. Therefore, this subsection explores the performance of SDQ in the context of multiple shifted data streams. In particular, all settings in Section IV.E are adopted. Furthermore, we examine downward shifts with $\delta = -1, -2,$ and -3, and evaluate scenarios where the number of true shifted data streams is 1, 2, and 3. The results are summarized in Table 3.9.

Table 3.9 reveals that applying the SDQ to scenarios with multiple shifted streams results in a consistent reduction in detection delay as the number of shifted data streams increases. These results indicate that though the SDQ assumes a single shifted data stream when learning the optimal policy, it can still effectively handle the case when multiple data streams have a mean shift. One possible reason is that SDQ always try to identify the m most informative data streams to collect data from. Similarly, Table 3.9 also indicates that an increase in the magnitude of the mean shift, $|\delta|$, or in the number of observable data streams leads to a decrease in detection delay, which aligns with our intuition.

Table 3.9. ARL_1 with the standard errors (in parentheses) under multiple shifted data streams

m	δ	One shifted data stream	Two shifted data streams	Three shifted data streams
2	-1.0	56.98	45.34	38.71

		(1.24)	(1.15)	(1.01)
	−2.0	39.27	31.89	27.46
		(1.01)	(0.98)	(0.96)
	−3.0	30.18	24.26	22.14
		(0.97)	(0.93)	(0.89)
6	−1.0	27.39	21.60	17.88
		(0.94)	(0.87)	(0.82)
	−2.0	15.27	11.31	9.47
		(0.77)	(0.71)	(0.68)
	−3.0	13.07	10.11	8.75
		(0.73)	(0.70)	(0.67)

3.5 Case Study

In this section, we demonstrate the real-world applicability of the proposed SDQ algorithm. In particular, we use the Tennessee Eastman Process as an example, a chemical plant where early detection of potential anomalies is crucial for hazard prevention. The dataset is publicly available in Harvard Dataverse [39]. There are in total 53 data streams with 250000 labeled in-control training samples, and 480000 labeled in-control testing samples. We preprocess the data by standardizing each data stream such that each data stream has a mean of 0 and a standard deviation of 1 when the process is in control. For offline training, we bootstrap the data as discussed in Section III.A.4 to from the training samples in each training episode, where a mean shift of either 1 or -1 is randomly inserted to one of the 53 data streams. Since the original out-of-control data

comprises 20 different, unknown fault types, our approach ensures the maintenance of a controlled environment with a known ground truth. This setup facilitates a more transparent evaluation and understanding of our method's performance relative to benchmarks. SDQ then trains the agent and derives an optimal policy. Next, we follow the optimal policy and determine the threshold such that the in-control ARL is 370. For each setting, we conduct 5000 simulation replications. We consider $m = 5$, 10, and 15, and the parameters are selected as l = 1, $k = 0.5$ for both SDQ and QUANTS methods. The result of ARL_1 is shown in Table 3.10 when there is a single mean shift in the system with magnitude of 1. As we can see from Table 3.10, SDQ outperforms QUANTS in all cases and the advantage of SDQ is more significant when m is small, i.e., when the number of monitoring resources is small. This is consistent with the observations in Section IV.

Table 3.10. Comparisons of ARL_1 with the standard errors (in parentheses) for the Tennessee Eastman Process

	QUANTS	SDQ
m = 5	170.33	80.71
	(1.84)	(1.67)
m = 10	151.28	69.03
	(1.82)	(1.55)
m = 15	138.40	61.44
	(1.80)	(1.48)

Table 3.11 shows the average computational times for SDQ and QUANTS during online monitoring as they update the sampling layout at each time step. The time difference primarily results from processing the neural network output. Despite these additional computations, the

results demonstrate that SA2C-I operates efficiently enough to be deployed in real-time applications.

Table 3.11. Average computation time during online monitoring for SDQ and QUANTS

	QUANTS	SDQ
Time (sec)	0.28	0.63

3.6 Conclusion

In this article, we propose a DRL-based adaptive sampling and monitoring framework tailored to high-dimensional processes when only dynamic partial observations are available at each acquisition time. Specifically, an extension of deep Q-network, the double dueling Q-network, is seamlessly integrated with SPC. We first construct a local statistic for each data stream and the double dueling Q-network is employed to learn a dynamic sampling strategy. A global monitoring statistic is then constructed based on the sum of top-l local statistics to indicate whether the system is out of control. At last, following the optimal policy suggested by the DRL, an appropriate threshold for maintaining a prescribed in-control ARL is determined. Compared to the existing literature, the proposed SDQ algorithm has the following unique advantages: (i) In contrast to previous algorithms that prioritize short-term rewards and adopt a myopic approach when deriving the sampling strategy, SDQ considers long-term discounted rewards and explicitly accounts for unattended time. As a result, SDQ makes smarter decisions, which leads to improved detection delays. (ii) Unlike previous methods that rely on heuristic compensations to encourage exploration, SDQ naturally handles the tradeoff between exploitation and exploration due to its efficient DRL framework. (iii) By utilizing a deep neural network structure, SDQ tackles scalability challenges

and enables the monitoring of numerous data streams simultaneously. This architecture empowers SDQ to a variety of monitoring applications in practice.

There are several future research directions. First, while SDQ successfully handles nonparametric data distributions, we assume that each data stream is independent and identically distributed across time. Thus, further studies are needed to extend SDQ's applicability to autocorrelated data streams. Furthermore, while SDQ effectively integrates SPC with value-based DRL, it still encounters scalability limitations when applied to environments with a large number of data streams. The training time and computational complexity increase significantly as the action space expands exponentially with the number of streams. One promising direction is to employ the Actor-Critic algorithm. This method combines the benefits of both policy-based and value-based approaches, allowing for the direct modeling and optimization of the policy function. This could significantly enhance the efficiency of the system, enabling it to handle thousands of data streams without a substantial increase in computational demand. We aim to explore this and other policy-based methods in our future research to address the scalability challenges.

Chapter 4 Cost-Effective Dynamic Sampling in High Dimensional Online Monitoring with A2C

4.1 Introduction

The rapid advancement of sensor technology, fundamental to Industry 4.0 and further evolved in Industry 5.0, has significantly enhanced our capability to monitor extensive data streams from myriad interconnected devices [55]. However, challenges such as the high cost of data collection, limited data transmission bandwidth, and restricted processing capabilities may hinder the complete observation of all data streams in real time. For example, in military applications, only a limited number of unmanned aerial vehicles are often available to conduct surveillance and rescue tasks [3]. In the realm of energy, sensors in power plants must be managed dynamically, toggling between the ‘on’ and ‘off’ modes for performance monitoring, owning to battery limitations [2]. Similarly, in certain manufacturing scenarios, limited processing power restricts the real-time analysis to only a subset of data streams for timely anomaly detection [1]. These scenarios underscore a central challenge of Industry 5.0: the need for methodologies that not only adaptively select the most critical data streams for observation based on system requirements but also enhance the synergy between human operators and automated systems. As a result, the key research challenge is to devise methods for adaptively choosing the optimal quantity and the most critical data streams to observe, enabling swift anomaly detection within the constraints of resource limitations. Such methods ensure that decision-making processes are not only efficient but also aligned with human-centric and sustainable industrial practices, facilitating a balance between technological capabilities and the intrinsic limitations of physical and economic resources. In

conclusion, our work is directly applicable and crucial to scholars and practitioners striving to enhance productivity and sustainability in various production environments, marking it as a pivotal contribution to the broader field of production research.

Despite the significance of the subject, the existing body of literature scarcely addresses the challenge of prompt anomaly detection, which involves dynamically selecting both the optimal quantity and specific data streams for observation at each time step. For instance, while some studies have applied Deep Reinforcement Learning (DRL) to the field of anomaly detection, their emphasis predominantly lies on diagnosis accuracy and anomaly localization, rather than on online monitoring for the earliest possible anomaly detection [34]–[36], [38], [56], [57]. Moreover, within the subset of research tackling prompt anomaly detection, a pervasive assumption is the complete observability of all data streams, which overlooks the practical constraints of limited resources [47], [48]. To enable dynamic selection of data streams, heuristic methodologies have been proposed for prompt anomaly detection (Liu et al., 2015; Xian et al., 2018; Ye et al., 2022; Ye & Liu, 2022). Nonetheless, these heuristic strategies exhibit two principal shortcomings: (i) they prioritize immediate advantages in selecting data streams, contrary to DRL's orientation towards maximizing long-term rewards, thus resulting in ineffective monitoring performance; and (ii) they presuppose a fixed number of observable data streams always available, thus failing to dynamically adjust the optimal quantity for observation. These considerations motivate us to propose a novel DRL framework designed for the online monitoring challenge under partial observation. This framework aims to strike a balance between exploiting suspicious data streams and exploring potential shifts across all streams, while autonomously determining the most informative and minimal requisite number of data streams for observation at each time epoch.

We define the anomaly detection problem as follows. There are in total of M data streams and $\boldsymbol{X}(t) = \big(X_1(t), \ldots, X_M(t)\big)'$ denotes the measurements of the data streams at time t. Each data stream can follow an arbitrary distribution. At each time epoch t, it is feasible to selectively acquire/sample measurements from m_t out of M available data streams, where $m_t \leq M$. Define $\mathcal{O}(t)$ and $\mathcal{U}(t)$ to be the sets of observable and unobservable data streams at time epoch t. Then $\mathcal{O}(t) \cup \mathcal{U}(t) = \{1, 2, \ldots, M\}$, $\mathcal{O}(t) \cap \mathcal{U}(t) = \emptyset$, and $|\mathcal{O}(t)| = m_t$, where $\emptyset$ denotes the empty set and $|\cdot|$ denotes the cardinality of a set. Besides, for $j = 1, \ldots, M$, we define $H_{j,t}$ as the duration for which data stream j remains unobserved up to time step t, a period we term as the "unattended time". Based on the above problem formulation, we make the following assumptions: (i) Without loss of generality, we assume the mean and standard deviation of $X_j(t), j = 1, \ldots, M$, are 0 and 1, when the process is in control. This can be done by pre-centering and pre-scaling the historical in-control data. (ii) The process becomes out of control at some unknown time T, and the mean $\boldsymbol{\mu} = E(\boldsymbol{X}(t))$ shifts away from $\mathbf{0}$ to an unknown value $\boldsymbol{\delta}$. The number of shifted data streams is also unknown. (iii) For $j = 1, \ldots, M$, $X_j(t)$ is independent and identically distributed (i.i.d.) over time in either the in-control or the out-of-control process. (iv) The sets $\mathcal{O}(t)$ and $\mathcal{U}(t)$ can be updated at each time epoch t without incurring extra cost. (v) There are sufficient partially observed data streams from the in-control process for offline training purposes. The research goal is to propose a generic monitoring framework that determines the most informative and minimally necessary quantity of data streams for observation at each time interval, which will facilitate the earliest possible detection of mean shifts in high-dimensional data streams, while ensuring a specified in-control average run length (ARL).

Considering that our interested problem involves sequential decision makings (i.e., determining the occurrence of anomalies and identifying how many and which data streams to

observe at each acquisition time), DRL emerges as a promising tool, especially when the underlying problem is complex due to high dimensionality and the unknown transition probability. However, to develop an effective DRL framework, several challenging questions must be addressed: (i) Which DRL algorithm to employ tailored to the problem context? (ii) How to effectively define the state and action spaces for the online monitoring problem with partial observations? It is also crucial to ensure the state and action spaces satisfy the Markov property, which is the key underlying assumption of DRL. (iii) How to design an appropriate neural network structure given that DRL is typically hard to converge? (iv) How to establish an intelligent reward framework to minimize detection delay? In this work, we aim to answer these questions. To the best of our knowledge, this is the first work that specifically addresses the dynamic selection of an optimal number and the most informative data streams for observation by establishing a novel method integrating DRL and SPC schemes.

In particular, we name the proposed algorithm the Statistical Advantage Actor-Critic algorithm (denoted as the "SA2C" algorithm). Specifically, the SA2C algorithm initially converts the raw observations from each data stream into a nonparametric local statistic, reflecting the likelihood of anomaly occurrences. These nonparametric statistics, alongside the duration of unattended time, constitute a state representation that informs the input to the deep neural network underpinning the Advantage Actor-Critic (A2C) algorithm. The neural network's output specifies the optimal number of data streams, m_t, to observe, as well as the probabilities p_t that characterize the likelihoods of observing each data stream at each time step. Subsequently, m_t data streams are sampled according to this probability distribution, facilitating the effective identification of the most critical data streams for observation. Ultimately, leveraging the optimal policy generated by A2C, we aggregate the top-l local statistics to construct a global statistic, thereby assessing if the

system is out of control. The proposed SA2C algorithm offers the following distinct advantages: (i) Different from existing heuristic approaches (Cheng & Thaga, 2005; Liu et al., 2015; Xian et al., 2018; Ye et al., 2022; Ye & Liu, 2022) that predominantly target the short-term gains resulting from sampling strategies in a myopic manner, the SA2C algorithm adopts a perspective that values the long-term discounted rewards. This approach effectively navigates the balance between exploitation of suspicious data streams and exploration of unobserved data streams for quick anomaly detection. (ii) Unlike earlier studies, SA2C is equipped to dynamically determine not only which specific data streams to observe but also the optimal number of observable data streams. (iii) The incorporation of a deep neural network framework addresses challenges related to scalability, enabling the monitoring of hundreds of data streams simultaneously.

The structure of this paper is outlined as follows: Section 4.2 delves into a review of the literature concerning Deep Reinforcement Learning (DRL) and monitoring of multivariate processes under resource constraints. In Section 4.3, we introduce the technical details of the SA2C algorithm. In Section 4.4, we conduct simulations to perform comparative analysis of the proposed method against existing benchmarks. Section 4.5 presents a case study on a semiconductor manufacturing process to elucidate the merits of the SA2C algorithm further. Finally, in Section 4.6, we provide concluding remarks and future research directions.

4.2 Literature Review

In Section 4.2.1, we review the basic framework of DRL. Specifically, we introduce the fundamental concepts of DRL including its two main categories of algorithms [17], [58], [59], as well as its applications in anomaly detection [53], [60], [61]. Then in Section 4.2.2, we review statistical methods for online monitoring of high-dimensional data streams with partial

information.

4.2.1 DRL and Its Applications in Anomaly Detection

DRL combines deep learning and reinforcement learning principles to enable algorithms to learn from their actions' outcomes in complex environments. In anomaly detection, DRL helps efficiently identify irregular patterns or unusual behaviours efficiently, optimizing detection mechanisms over time through trial and error. Section 4.2.1.1 outlines the foundational concepts of DRL. Section 4.2.1.2 presents the two primary types of algorithms: value-based and policy-based. Lastly, Section 4.2.1.3 delves into the real-world applications of DRL for anomaly detection.

4.2.1.1 Foundational Concepts of DRL

Deep Reinforcement Learning (DRL) is a synthesis of reinforcement learning (RL) and deep learning (DL), leveraging deep neural networks to approximate certain functions essential for making decisions in RL. This integration addresses the challenge of high-dimensional input spaces, which are common in complex decision-making problems.

In the RL framework, an agent interacts with an environment over discrete time steps. At each time step t, the agent observes a state s_t from the state space S, takes an action a_t from the action space A, receives a scalar reward R_{t+1}, and transitions to a new state s_{t+1} according to the environment's dynamics. The objective of the agent is to learn a policy $\pi(a|s)$ that maximizes the expected sum of discounted rewards $G_t \stackrel{\text{def}}{=} R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=1}^{\infty} \gamma^{k-1} R_{t+k}$, where γ is a discount rate. The policy function π is the strategy that the agent employs to determine actions given states. In DRL, a deep neural network is often used to parameterize the policy

function, $\pi_\theta(a|s)$, where θ represents the network's weights. There are two types of value-based functions in RL: the value function $v_\pi(s)$, which estimates the expected return from state s following policy π; and the action-value function $q_\pi(s,a)$, which estimates the expected return from taking action a in state s and thereafter following policy π. These functions are often approximated by deep neural networks in DRL as well.

4.2.1.2 DRL Algorithms

At the core of DRL are two primary classes of algorithms: value-based and policy-based methods. Value-based algorithms focus on estimating the value of each action in a given state. The most notable value-based method is the Deep Q-Network (DQN), which utilizes a deep neural network to approximate the action-value function, $q(s,a)$. The core update equation in DQN is derived from the Bellman equation for Q-learning:

$$q_\pi(s_t,a_t) \leftarrow q_\pi(s_t,a_t) + \alpha\left(R_{t+1} + \gamma \max_a q(s_{t+1},a) - q(s_t,a_t)\right), \quad (1)$$

where α is the learning rate. DQN and its variants (such as Double DQN [51], Dueling DQN [52]) have shown remarkable success in domains like playing Atari games directly from pixels, demonstrating the power of combining deep learning with Q-learning, especially in environments with small discrete action spaces.

In contrast, policy-based methods directly learn the policy that determines the agent's actions, without the intermediary step of value estimation. The goal is to find the optimal θ that maximizes the expected return $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)]$, where τ denotes a trajectory, i.e., a sequence

of states, actions, and rewards, and $R(\tau)$ is the reward of the trajectory. To optimize the policy, we calculate the gradient of $J(\theta)$ with respect to the policy parameter θ:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) q_\pi(s_t, a_t)], \quad (2)$$

where $q_\pi(s, a)$ is the action-value function under policy π_θ. We can then update θ using the gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta), \quad (3)$$

where α is the learning rate. Policy-based methods, such as REINFORCE [62] and its more sophisticated successors like Proximal Policy Optimization [63] are particularly adept at handling environments with continuous or high-dimensional action spaces.

Actor-Critic methods represent a hybrid approach, leveraging the strengths of both value-based and policy-based learning. Here, the "actor" learns to select actions based on a policy model, while the "critic" evaluates those actions using a value function, which estimates how good a particular state or action is, given the current policy. This combination allows for more stable and efficient learning by reducing the variance of policy updates and providing more precise feedback on the quality of actions taken [64]. Advantage Actor-Critic [65], referred as A2C, uses the advantage function and parallel environments to update policies. The advantage is defined as

$$A(s_t, a_t) = q(s_t, a_t) - v(s_t), \quad (4)$$

which measures the benefit of taking a specific action a_t over the average action as suggested by the policy in state s_t. This helps in reducing the variance of updates and focusing on actions that are better than average.

4.2.1.3 Anomaly Detection with Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) has been widely adopted for anomaly detection across several fields. For example, it has been proven effective in identifying and locating anomalies in videos [34], [35]. In the area of cloud security, DRL plays an essential role in monitoring cloud-based applications for malicious behavior, thereby offering a robust defense against cyber threats [36], [44], [47]. Additionally, DRL has been instrumental in enhancing the efficacy of Network Intrusion Detection Systems (NIDS) within organizations, ensuring secure communication and data protection [38], [45], [46].

However, all of the research previously discussed primarily concentrates on assessing the accuracy of anomaly diagnosis or identifying anomalies using the value-based DQN method. Even though some work acknowledges an unknown number of abnormal processes at any given time, the limitation arises from the agent's capability to inspect only one sensor at each time step [47]. In such scenarios, DQN is suitable due to its effectiveness in handling discrete and relatively small action spaces, where the number of available actions increases linearly with the number of data streams. However, this approach falls short in our interested problem context of online dynamic monitoring with limited resources. For instance, when selecting m_t from M available data streams for observing, the set of possible actions expands to $\binom{M}{m_t}$, exhibiting exponential growth. Moreover, the complexity escalates if we further need to decide the optimal number of data streams to observe, leading to 2^M possible actions. This characteristic highlights a pressing need for a more advanced DRL framework tailored for online dynamic monitoring, capable of detecting anomalies as early as possible in high-dimensional settings where we can freely select the number and which data streams to observe.

4.2.2 Online Monitoring of High-dimensional Data Streams with Partial Information

To monitor high-dimensional data streams, conventional studies have applied dimension reduction techniques to transform original observations into different features [3], [49], [50], [66]. Yet, these approaches often assume the complete observations of data streams available at every time point, overlooking the practical limitations of resources. In response, [41] introduced a Bayesian model that integrates Thompson Sampling for choosing observable sensors and a method for decomposing signals in the high-dimensional contexts. This solution, however, may lead to a partial depiction of the original system with potential loss of information.

Another direction taken by researchers involves creating local statistics for individual data streams (Celano & Chakraborti, 2021; Graham et al., 2017; Hou & Yu, 2021; Li et al., 2016; Liu et al., 2023; Liu et al., 2014). However, these studies often overlook constraints related to resource limitations. Subsequent research has introduced methods that compensate for unobserved data streams when constructing local statistics. These approaches can be divided into two main categories: one that involves sampling across the time domain, and another that focuses on the spatial domain. In the time-domain category, efforts have been made to enhance the efficiency of multivariate control charts by adaptively adjusting the sampling intervals across all data streams over time [73]–[76]. However, these methods often assume uniform sampling intervals across all data streams at any specific moment, which limits their applicability in practice. In the spatial-domain category, decisions are made on which subsets of data streams to monitor at each time step, with a fixed number of data streams being observable. A pioneering algorithm (Liu et al., 2015) addressed sampling under resource constraints for Gaussian-distributed data streams, and it was extended later by [9] to deal with non-Gaussian yet exchangeable processes. This was further

expanded by (Ye et al., 2023) to accommodate data streams of any distribution with Thompson Sampling. Similarly, [39] integrate Thompson Sampling with sequential change-point detection methods such as the Shiryaev-Roberts-Pollak procedure. [6] also leverage the spatial structure among data streams to enhance monitoring performance. Nevertheless, these methods are predominantly heuristic, emphasizing immediate gains rather than long-term benefits when updating sampling strategy. Furthermore, they restrict observation to a predetermined number of data streams at each time, limiting flexibility in adjusting the number of data streams to observe. To address these shortcomings, this paper introduces a novel Statistical Advantage Actor-Critic algorithm.

4.3 Methodology

While DRL appears to be a viable approach for overseeing high-dimensional data streams under resource limitations, there are several unique challenges. First, there is a necessity to establish a state space that accurately reflects the system's current condition. Second, given the potential for the action space to expand exponentially, reaching a size as large as 2^M when freely selecting among data streams to observe, a scalable action space is essential to clearly indicate which and how many streams are observed. Third, a sophisticated reward mechanism is crucial that helps the agent promptly identify anomalies while also considering the cost associated with observing data streams. Ultimately, establishing a suitable DRL algorithm, along with crafting the right framework and design, remains to be a great challenge for online dynamic monitoring.

To tackle the previously mentioned challenges, we present the Statistical Advantage Actor-Critic (SA2C) Algorithm. Recognizing the limitations of value-based algorithms, our approach centers on policy-based DRL methods, which inherently represent the probability distribution of

actions. We chose to adapt one of the leading-edge algorithms, the Advantage Actor-Critic [65], which enhances the original Actor-Critic method by offering a steadier training experience. This is achieved through the utilization of advantage function, and the numerous parallel environments for calculating the gradient estimate. SA2C is structured around two main phases: an offline training phase and an online implementation phase. The offline phase includes two critical stages: initially, in Stage-I, we develop a nonparametric local statistic for each data stream, utilizing these statistics along with unattended time to form a comprehensive state representation of the system for SA2C. Subsequently, we train a neural network to process this state representation, producing an optimal monitoring policy π^* that specifies which and how many data streams to observe at every time step. In Stage-II, following the learned policy, we set a monitoring threshold for alarm activation to achieve a predetermined ARL. During online implementation, the established policy π^* and threshold are applied to monitor the processes in real time.

The specifics of the SA2C algorithm are outlined as follows. In Section 4.3.1, we detail the process of defining the state space, action space, and reward function, followed by training the neural network through simulations. Section 4.3.2 discusses the creation of a global statistic and the selection of a threshold that aligns with a predetermined in-control ARL, setting the stage for real-time online monitoring. Lastly, Section 4.3.3 offers practical advice on configuring the parameters of the SA2C algorithm for optimal performance.

4.3.1 Offline Training of SA2C

In this subsection, we define the state space in Section 4.3.1.1, outline the action space in Section 4.3.1.2, and detail the reward assignment within the SA2C framework in Section 4.3.1.3. We then further discuss the neural networks training in Section 4.3.1.4. Lastly, we offer a detailed overview

of Stage-I SA2C.

4.3.1.1 State Space

The state space must comprehensively capture the system's status, and thus we propose that the state space includes two important pieces of information for each data stream j at time t: the local statistic value $W_{j,t}$ which characterizes the likelihood of a change occurring in this data stream; and the unattended time $H_{j,t}$, indicating the duration this data stream has remained unobserved up to time t. Incorporating $H_{j,t}$ is beneficial as it allows for prioritizing the observation of data streams that have been neglected for a long period. For instance, between two data streams j_1 and j_2, where $W_{j_1,t} = W_{j_2,t}$ while $H_{j_1,t} > H_{j_2,t}$, it is more desired to monitor j_1 over j_2 as it is more likely that a shift has occurred in j_1 but we do not know yet due to lack of observations. Furthermore, motivated by the concept of the advantage function in (3), we introduce a revised version of the local statistics: $W'_{j,t} \coloneqq W_{j,t} - \bar{W}_t$, where $\bar{W}_t$ represents the average of local statistics at time step t: $\bar{W}_t \coloneqq \sum_j W_{j,t}$. Consequently, $W'_{j,t}$ quantifies the extent to which a variation is more pronounced in data stream j relative to the collective variations in all data streams at the same time step t. This helps the SA2C agent prioritize observing data streams with positive $W'_{j,t}$ values as they indicate a higher level of suspicion relative to the others. With M total data streams, the complete state representation of the system at any given time t can be expressed as $S_t = (\boldsymbol{W}'_t, \boldsymbol{H}_t) = \left(W'_{1,t},\ W'_{2,t}, \dots, W'_{M,t}, H_{1,t}, H_{2,t}, \dots, H_{M,t}\right)$.

We now describe how to update the state space S_t. While numerous methodologies exist for nonparametrically constructing the local statistics (Liu et al., 2015; A. Wang et al., 2018; Xian et al., 2018, 2021; Ye & Liu, 2022; Zan et al., 2022), our aim is to create a versatile framework

that accommodates a variety of statistical methods. Such an approach ensures that the SA2C framework is adaptable and effective for a broad spectrum of anomaly detection scenarios. To this end, we specify three essential criteria for the local statistics: (i) A larger statistic value suggests that an anomaly is more likely to have occurred, (ii) When the process is in control, the statistic value oscillates around 0, and when the system is out of control, the statistic value continuously increases; (iii) The statistic is memoryless, i.e., given the current statistics, future values depend solely on the current state, not past values. Without loss of generality, in this paper, we exemplify the construction and update of the local statistics by adopting the quantile-based statistic method [14], as detailed in Chapter 2 which satisfy the above three criteria. For the update of $H_{j,t}$, if data stream j is not observed at time t, then $H_{j,t}$ is increased by one. Conversely, if data stream j is observed at time t, $H_{j,t}$ is reset to zero. Thus, $H_{j,t}$ can be updated as

$$H_{j,t} = \begin{cases} 0 & \text{if } j \in \mathcal{O}(t) \\ H_{j,t-1} + 1 & \text{if } j \in \mathcal{U}(t) \end{cases}. \tag{5}$$

4.3.1.2 Action Space

In the structure of our problem, each time epoch requires making two critical decisions: selecting the data streams for observation and determining if an alarm should be triggered. Merging these two decisions into the action space would inevitably increase its cardinality, and potentially rendering the algorithm more challenging to converge. Therefore, we propose to divide the decision-making into two stages. In Stage-I, the SA2C algorithm focuses solely on devising a sampling strategy on which data streams to observe at each time epoch. In Stage-II, we rely on the SPC technique, which will be discussed in Section 4.3.2, to effectively decide whether to raise an alarm based on a global monitoring statistic.

Since A2C generates a stochastic policy, we define the action space as a $1 \times (M+1)$ vector $\boldsymbol{A}_t$, where $\boldsymbol{A}_t = (A_{1,t}, A_{2,t}, \dots, A_{M,t}, m_t)$. Each of the first M entries, $A_{j,t}$, denote the likelihood of selecting data stream j for observation and the sum of the probability equals 1: $\sum_{j=1}^{M} A_{j,t} = 1$. The final element, $m_t \in \mathbb{Z}$, specifies the desired number of data streams to be observed at time step t and ranges from $[0, M+1)$. Accordingly, we observe m_t data streams based on the probability distribution $p_t = (A_{1,t}, A_{2,t}, \dots, A_{M,t})$ for the next time step. This method allows for a linear increase in complexity with the total number of data streams M, ensuring the algorithm remains scalable. Finally, we examine the Markov property associated with the state and action space. Given the current state $S_t = (W_t', H_t)$ and the action $\boldsymbol{A}_t$, the subsequent state $\boldsymbol{W}_{t+1}'$ is solely determined by $\boldsymbol{W}_t$ and the new data at $t+1$. Similarly, $\boldsymbol{H}_{t+1}$ only depends on $\boldsymbol{H}_t$ and $\boldsymbol{A}_t$.

4.3.1.3 Reward

This subsection elaborates on how rewards are designed during the offline training phase, which is a crucial component for guiding the agent's learning process. The reward structure is bifurcated: one part punishes the agent for not being able to observe the anomalous data streams, while the other part imposes a cost relative to the number of data streams the agent chooses to observe. Specifically, when the system is in control, since there is no anomaly, the agent receives 0 punishment. However, the agent incurs a cost of C for each data stream it selects for observation and thus the reward is set as follows when $t < T$:

$$R_t = -m_t \times C, \qquad \text{for } t < T. \tag{6}$$

For simplicity, here we assume that the cost increases linearly with the number of observed data streams. However, a more intricate cost formula could be devised in different problem context.

When the system goes out of control at time T, we allocate a substantial punishment to the agent, which increases exponentially over time, if the agent fails to observe the shifted data streams. In addition, we let the punishment to be proportional to the number of shifted data streams that the agent fails to monitor, q_t. In this way, the reward formula can be expressed as $-q_t \times e^{\frac{t-T}{\beta}}$, where β is a hyperparameter determining the increasing rate for the punishment. The choice of an exponentially increasing punishment, which is widely recognized in deep reinforcement learning literature as an effective mechanism for reward structuring [34], [47], [53], [54], aims at incentivizing rapid anomaly detection here. To prevent the punishment from becoming unmanageably large, a constant value $r_1 = -q_t \times e^{\frac{p-T}{\beta}}$ is applied to the agent after an extended duration, p, if the agent still fails to observe the shifted data streams beyond this period. Similar to the in-control scenario, the cost of observing m_t data streams at each time step is represented as $C * m_t$. In this way, we can set the reward as follows if the agent observes q_t shifted data streams when $t \geq T$:

$$R_t = -q_t \times e^{\frac{\min(t,p)-T}{\beta}} - m_t \times C, \quad \text{for } t \geq T. \tag{7}$$

Based on our empirical results, a p value of 50 has been found to be satisfactory.

In summary, the reward scheme is designed to encourage the agent to maintain the number of observed data streams at an essential minimum. However, observing too few data streams significantly increases the risk of overlooking an anomaly when it occurs. Hence, the agent needs to well balance the trade-off between the exploration of more data streams vs. the exploitation of suspicious data streams by constantly interacting with the system. Moreover, we should highlight that from the practical standpoint, this reward mechanism can be directly linked to the actual costs,

including both fixed and variable expenses associated with setting up observing data streams and the financial consequences of failing to detect anomalies.

4.3.1.4 Neural Network Training

Once the state, action, and reward definitions are established, the next step is to train the A2C agent using a neural network framework. However, there are three key challenges to consider. First of all, it is a standard practice to jointly train the actor and critic networks using a unified neural network model [65]. This unified approach facilitates efficient learning through shared layers and streamlines the training process. Nevertheless, given the complexity of our action space comparing to its standard usage [54], [65], [77], adopting a distinct neural network architecture for the actor, separate from the critic, will be more advantageous. By training the Actor and Critic networks separately, we allow each network to focus on its specific objective without interference. This decoupling can prevent the propagation of suboptimal gradients that might occur when training both networks simultaneously. This separation also allows for tailored actor design, enhancing its capability to effectively navigate the intricate action space, reducing the risk of getting trapped in local optima. Moreover, assigning different learning rates to the actor and critic helps prevent one component from lagging too far behind the other, ensuring more balanced learning progress.

The second challenge in our problem context is that we require to observe m_t data streams. This means we not only need to output the probability distribution of sampling each data stream, but also make sure in total m_t data streams are observed. To address this issue, we structure the output layer so each neuron correlates to the likelihood of observing a particular data stream, with the final neuron indicating the total number of data streams observed at that timestep. Inspired by

the dueling network architecture [52], we split the action neural network into two distinct pathways: one learning the probability distribution associated with selecting each data stream, which concludes with a softmax layer for the M neurons; and the other one which estimates the single neuron m_t, shares the first couple of layers with the first pathway, end with a sigmoid function multiplied by $M + 1$ to ensure $m_t \in [0, M + 1)$. The combined output of these pathways forms a $1 \times (M + 1)$ vector, where last element is also floored so that $m_t \in \mathbb{Z}$. A graphical representation of the neural network is shown in Appendix.

Lastly, traditional A2C implementations [78]–[80] often accumulate experiences throughout an entire episode, updating the model post-episode. This approach suits scenarios with distinct start and end points, where rewards are significantly influenced by the concluding state (such as solving a puzzle). However, in our specific problem context, it is essential to incorporate continuous feedback at each time step because it provides valuable information about the number of shifted data streams observed and the total cost of resource deployment. Recognizing that each action triggers an immediate reward, the existing literature suggests training at each time step rather than after each episode [65]. We adopt this approach in our model, SA2C, which continuously trains at every time step and thereby accelerating convergence.

4.3.1.5 Overview of Stage-I SA2C

One of the key challenges in DRL is the scarcity of training data, especially when it comes to online monitoring tasks. These tasks necessitate a substantial amount of both in-control and out-of-control data to effectively navigate through various reward scenarios. To overcome this hurdle, we employ the bootstrap method (Liu et al., 2015; Xian et al., 2018; Ye & Liu, 2022) to generate necessary in-control data. Specifically, we use bootstrapping to replicate historical in-control data

T^0 times, thereby creating T^0 in-control observations for each training episode. Furthermore, considering that the out-of-control historical data may be sparse or even non-existent, we simulate T^1 out-of-control samples by introducing a mean shift in a randomly chosen in-control data stream. The magnitude of the mean shift should be set based on the smallest significant shift we aim to detect. If the mean shift introduced during training exceeds that during online monitoring, the algorithm might mistakenly identify the process as being in control. Although it is advisable for the direction of the mean shift to align with the actual shift direction, it is not mandatory. This flexibility is due to the calculation of local statistics $W_{j,t} = max\left(W_{j,t}^+, W_{j,t}^-\right)$, where $W_{j,t}^+$ helps detect positive shifts and $W_{j,t}^-$ helps detect negative ones. By taking the maximum value of the two, the algorithm effectively identifies significant shifts, regardless of whether the training and actual data shifts align, thus indicating the presence of a suspicious mean shift. Consequently, each training episode comprises $T^0 + T^1$ observations. Our empirical study suggests that setting T^0 to follow a uniform distribution between 100 and 200 and fixing T^1 at 150 is sufficient to ensure a stable and efficient training experience. In addition, we let α_c and α_a denote the learning rate of the critic network and the actor network, respectively. We will further discuss the selection of the parameters involved in Section 4.3.3.

Now we have addressed all aforementioned challenges. Specifically, we defined a comprehensive state space, which includes the local statistics and the duration that each data stream has remained unattended, to fully capture the system's status at any given moment. Secondly, we defined the action space that can effectively decide the number and which data stream to observe. Such an approach is vital for optimizing resource allocation and ensuring that critical data streams receive attention when they need. Thirdly, we introduced a generic reward assignment scheme, which strikes a delicate balance between the necessity of exploring new data

streams and the need to exploit suspicious data streams effectively. It addresses the critical challenge of managing the trade-offs between the cost associated with missed detection of anomalies and the costs incurred from observing unnecessary additional data streams. Lastly, we have enhanced the actor-critic algorithm (A2C) to make it tailored to our problem context. We separated the actor and critic network, designed a specialized actor network, and trained the agent at each time step instead of updating the model post-episode. An overview of the Stage-I SA2C is summarized in Figure 4.1.

Input: historical in-control data, hyperparameter $\gamma, \alpha_a, \alpha_c, \delta, p, \beta, C, epi_{num}$.

Output: Learned actor and critic network that generates the optimal policy π^*

Initial setup: Initialize the actor and critic network with the Xavier initialization.

Algorithm:

For $episode = 1, \dots, epi_{num}$, repeat steps (I)-(III):

III. Randomly select a data stream as the shifted data stream and simulate the out-of-control data by inserting mean shift δ to that data stream.

IX. Sample T^0 from a uniform distribution $U(100,200)$ and set $T^1 = 150$. Bootstrap in-control data T^0 times followed by bootstrapping the out-of-control data T^1 times.

X. Randomly select data streams for observation and initialize the state S_1.

XI. For each time epoch $t = 1, 2, \dots, \ T^0 + T^1$:

a. Input the current state S_t into the actor network, obtain the probability distribution p_t which indicates the likelihood of observing each data stream, as well as m_t, the number of data streams to observe. Sample m_t data streams to observe according to p_t without replacement to get action A_t.

b. Take action A_t. Construct local statistics for both observed and unobserved data streams. Determine the next state S_{t+1} as outlined in Section 4.3.1.1.

c. Calculate reward R_t as described in Section 4.3.1.3.

d. Update the actor and critic networks using the A2C algorithm defined in III.A.4.

End

Fig. 4.1 Overview of the SA2C stage-I algorithm.

4.3.2 Monitoring Framework

Recall in Section 4.3.1.2 we separated the decision processes for choosing a sampling layout and determining when to issue an alarm. In this subsection, we propose a monitoring procedure that decides the latter. Specifically, we construct a global statistic to detect the occurrence of a mean shift in any data stream and compare its value against a predetermined threshold H, which is set to achieve a pre-specified ARL. Specifically, we take the local statistics $W_{j,t}$ for each observed data stream (i.e., data streams in set $\mathcal{O}(t)$) at time step t, and arrange them in a descending order. Mathematically, we let $W_{(1),t} \geq W_{(2),t} \geq \cdots \geq W_{(m_t),t}$ denote the sorted local statistics for observed data streams. The global monitoring statistic is then formulated by summing the top l local statistics:

$$g(t) = \sum_{j=1}^{l} W_{(j),t}\,. \tag{8}$$

Note that here we only sort the local statistics for the m_t observed data streams as the local statistics constructed for unobserved data streams may lack accuracy. The parameter, l, which represents the number of top local statistics to sum for creating the global statistic, is a hyperparameter whose setting is further discussed in Section 4.3.3.

During online monitoring, an alarm is triggered whenever this global statistic exceeds the threshold H, and thus the stopping time is expressed as $\tau = \inf\{t \geq 1: g(t) \geq H\}$. The introduction of this global statistic offers two main benefits: Firstly, as outlined in Section 4.3.1.1, decoupling the alarm-raising decision from the DRL framework reduces the complexity of the action space. Secondly, this global statistic is capable of detecting a broad spectrum of potential shifts in any direction, without the prerequisite of knowing the specific pattern of the shift.

During the online monitoring phase, the critic network and the process of reward assignment become useless because we have already acquired the optimal policy, denoted as π^*. In other words, we are only interested in executing the policy, i.e., taking the learned optimal actions based on the current state. Thus, the actor network is sufficient as it is responsible for directly mapping states to their corresponding actions. The online process then includes continuously updating the local statistics for both observed and unobserved data streams to establish the state and raise an alarm when $g(t)$ is larger than the threshold H. If no alarm is triggered, the current state is fed back into the actor network to determine the best course of action. Consequently, the computational effort primarily splits into two segments. The first involves the computation of M local statistics $W_{j,t}$, which, due to their recursive nature, do not increase memory demands over time. The second segment focuses on establishing the sampling layout based on the actor network's output which specifies the probability of sampling each data stream, p_t and the total number of data streams to be observed, m_t. This procedure involves sampling m_t times according to p_t without replacement and can be executed in linear time. Therefore, the SA2C algorithm's online execution is highly efficient and suited for real-time applications.

4.3.3 Parameter Setting

In this subsection, we discuss how to set the parameters involved in the SA2C algorithm. We examine the following hyperparameters: $l, \alpha_a, \alpha_c, \gamma, \beta, C$ and H.

1. l: The parameter l represents the count of aggregated local statistics in equation (8). Practically, l should be smaller than m_t to avoid incorporating statistics from unobserved data streams when calculating $g(t)$. While the optimal setting for l would equate to the actual number of data streams experiencing shifts, this information is often unavailable. Under these circumstances, [21] suggests a small l as it enhances the robustness of the system in identifying a broad spectrum of potential shifts.

2. α_a, α_c: In Stage-I of the SA2C algorithm, α_a and α_c refer to the learning rates for the actor and critic networks, respectively. These parameters guide the gradient descent optimization process in equation (3). Since we train the network at each time step and use the Xavier initialization to set the initial network weights, we set $\alpha_a = 5 * 10^{-8}$ and $\alpha_c = 5 * 10^{-4}$ based on the existing research [32] and practical experiments.

3. γ: γ is the discount factor in equation (1). In our numerical studies, we set $\gamma = 0.99$ based on the empirical results and existing practices.

4. β: β relates to the punishment rate when the agent fails to observe the shifted data streams as described in equation (7). When β increases, the punishment for missed detections over time diminishes. Consequently, the cost associated with resource usage plays a more important role within the reward framework, incentivizing the agent to adopt a more cautious strategy to conserve resources. This approach, in turn, may result in a longer detection delay. We will further study the effect of β in Section 4.4.2.

5. C: C denotes the cost incurred by the agent for each additional data stream it decides to observe, as specified in equation (7). A larger C leads the agent to be more conservative in choosing data streams to observe, whereas a lower C incentivizes the agent to observe more data streams. We will further study the effect of C in Section 4.4.2.

6. H:the threshold H can be determined through a bootstrapping process (Liu et al., 2015; Xian et al., 2018; Ye & Liu, 2022). This involves generating in-control datasets by bootstrapping from historical data. With a specified threshold value H, online monitoring can proceed according to the learned optimal policy π^* from the actor network. This approach allows for the estimation of the ARL. Consequently, the threshold H can be fine-tuned to ensure that the actual ARL matches a predetermined target value ARL_0.

4.4 Simulations

In this section, we conduct a comprehensive evaluation of the SA2C algorithm's performance. Section 4.4.1 offers a detailed comparison of the SA2C algorithm with established benchmark methods, including QH01 [11], QUANTS [13], and Random Sampling (RS). These benchmark methods monitor a fixed number of data streams m, while SA2C is the only algorithm in the current literature that has the flexibility to adjust m at each time step t. To ensure a fair comparison, we align the average number of data streams observed by SA2C, $\bar{m} = \frac{\sum_t m_t}{t}$, with the fixed number of data streams, m, observed by the benchmark methods. Given the difficulty in ensuring that $\bar{m}$ exactly matches m, we round $\bar{m}$ up to m. This approach guarantees that SA2C consistently utilizes resources no more than benchmark methods on average, and we refer to our SA2C algorithm under this approach as SA2C-I. While using fewer resources can put SA2C-I at a disadvantage, it ensures a convincing comparison across methodologies. Furthermore, we additionally evaluate SA2C's

performance with a fixed number of observable data streams to highlight the benefits of adaptively selecting m_t, and we refer to our SA2C algorithm under this approach as SA2C-II. In Section 4.4.2, we explore how the parameters β and C influence the balance between anomaly detection speed and resource utilization. In Section 4.4.3, we further diagnose the performance of SA2C-I by analysing the average number of data streams $\bar{m}$ observed by the agent during both in-control and out-of-control scenarios as well as the sampling layout. Lastly, in Section 4.4.4, we examine the ability of SA2C to monitor multiple shifted data. All evaluations are based on 5,000 simulation runs with a target in-control ARL, ARL_0, set at 370.

4.4.1 Comparative Analysis of Algorithm Performance

In this subsection, we thoroughly evaluate the performance of the SA2C algorithm and compare it with the state-of-the-art methods: the QH01 method, the QUANTS method, and the RS method. The QH01 method is an anti-rank based nonparametric monitoring technique which assumes access to all observations of data streams. Thus, it is unfair to directly compare QH01 with other benchmarks. However, this method provides a reference point for ideal situation under complete observable information. The QUANTS method employs the same approach for generating local statistics as SA2C but utilizes a heuristic sampling strategy. Note that QUANTS is a state-of-the-art algorithm derived from control chart methods, tailored for resource-limited conditions and capable of handling nonparametric data distributions. Meanwhile, the RS method, similar to SA2C and QUANTS, uses identical local statistics but opts for a random selection of m data streams for observation at each time step. In addition, both RS and QUANTS fail to leverage the information of how long each data stream has not been observed in decision-making processes. Last but not least, unlike these benchmarks which observe a fixed number of data streams at each time step,

SA2C dynamically adjusts both the quantity and layout of the observed data streams.

To compare fairly, we examine two versions of SA2C. The first, SA2C-I, maintains the average number of observed data streams, calculated as $\bar{m} = \frac{\sum_t m_t}{t}$, consistent with the fixed number of observable data streams, m, in the benchmark methods. The second version, SA2C-II, evaluates SA2C when the number of observable data stream is always fixed at m for each time step. In order to do so, we adjust the actor's network to only learn the probability distribution $p_t = (A_{1,t}, A_{2,t}, \dots, A_{M,t})$ through a neural network and a softmax layer, and omit the estimation of the single neuron m_t. Additionally, this version removes the cost $m_t * C$ from the reward structure.

In our simulation studies, we assume that there are 15 independent data streams, i.e., $M = 15$. We further diversify these data streams by dividing them into 5 groups, each following a distinct statistical distribution: standard normal distribution, standard t-distribution with 3 degrees of freedom, exponential distribution with $\mu = 1$, log-normal distribution with $\mu = 1$ and $\sigma = 0.5$, and poison distribution with $\lambda = 10$. We also standardize each data stream to achieve zero mean and unit variance when the process is in control. For offline training, we adopt a bootstrap approach as outlined in Section 4.3.1.5. This involves randomly drawing $T^0 + T^1$ samples with replacement from a pool of historical in-control data, with T^0 sampled from a uniform distribution between 100 and 200 and $T^1 = 150$. The initial T^0 samples are used as the in-control training data, while a mean shift of δ is randomly inserted to one of the 15 data streams in the last T^1 samples. This process facilitates the training of both SA2C-I and SA2C-II models as depicted in Figure 4.1, leading to the derivation of an optimal policy. With the learned policy, the monitoring threshold of the global statistic is determined such that the in-control ARL is 370. To assess the system's performance in online monitoring, we replicate these simulations, generating new testing samples by inserting a mean shift of δ to one of the 15 data streams from time step 0. The parameters for

all of the methods are selected as follows: $l = 1, k = 0.5$ for the SA2C-I, SA2C-II, QUANTS, and RS methods, and $k = 0.05$ for the QH01 method. The efficacy of these algorithms is then evaluated across varying m values and for mean shifts of $\delta = \pm 1, \pm 2$, and ± 3. The out-of-control ARL, ARL_1, are detailed in Table 4.1, with standard deviations provided in parentheses.

Table 4.1 Performance comparisons of ARL_1 when monitoring a single mean shift

	δ	SA2C-I	SA2C-II	RS	QH01	QUANTS
$m = 2$	3	27.52 (1.61)	38.08 (1.78)	>100	16.32 (0.81)	45.65 (1.28)
	2	32.91 (1.76)	44.73 (1.25)	>100	21.72 (1.12)	57.64 (1.47)
	1	41.03 (1.79)	55.21 (1.31)	>100	73.74 (2.36)	107.94 (2.15)
	-1	40.22 (1.79)	53.86 (1.30)	>100	70.33 (2.32)	103.17 (2.06)
	-2	30.80 (1.76)	42.80 (1.24)	>100	20.46 (1.09)	54.21 (1.43)
	-3	28.04 (1.62)	36.72 (1.77)	>100	15.75 (0.81)	45.83 (1.36)
$m = 4$	3	19.44 (1.23)	24.73 (0.84)	58.49 (0.37)	16.32 (0.81)	22.67 (0.30)
	2	24.09 (1.55)	30.60 (1.12)	74.08 (0.46)	21.72 (1.12)	27.77 (0.76)
	1	35.77 (1.81)	42.90 (1.23)	>100	73.74 (2.36)	63.46 (1.24)
	-1	33.79 (1.82)	41.03 (1.22)	>100	70.33 (2.32)	61.74 (1.18)
	-2	23.19 (1.59)	28.95 (1.07)	74.88 (0.46)	20.46 (1.09)	25.79 (0.78)
	-3	19.36 (1.11)	22.76 (0.86)	58.43 (0.37)	15.75 (0.81)	21.83 (0.28)
$m = 6$	3	14.18 (1.22)	16.03 (0.74)	38.61 (0.32)	16.32 (0.81)	10.40 (0.26)
	2	19.61 (1.30)	23.11 (0.83)	43.80 (0.33)	21.72 (1.12)	15.00 (0.32)
	1	31.32 (1.85)	36.47 (1.18)	>100	73.74 (2.36)	45.77 (0.75)
	-1	30.92 (1.85)	34.14 (1.18)	>100	70.33 (2.32)	44.60 (0.73)
	-2	18.68	21.88	43.83	20.46	13.98

		(1.26)	(0.81)	(0.33)	(1.09)	(0.38)
	-3	14.66	15.35	38.55	15.75	10.56
		(1.17)	(0.74)	(0.32)	(0.81)	(0.25)

From Table 4.1, we can draw several interesting conclusions:

(i) For all comparison methods, a larger mean shift magnitude $|\delta|$ or an increased number of available resources m leads to quicker detections. This is attributed to the fact that a larger δ indicates a more noticeable mean shift, making it easier to be detected. Additionally, as more information becomes available when m increases, the detection delay also becomes smaller except for QH01 which assumes all data streams are observable all the time.

(ii): SA2C-I consistently outperforms SA2C-II across all values of δ and m. This superiority stems from SA2C-I's flexibility in adjusting the number of data streams observed at each step, unlike SA2C-II whose resource use is fixed. Essentially, SA2C-II can be regarded as a special case of SA2C-I by fixing $m_t = m, \forall\, t$. Indeed, SA2C-I can explore more data streams when no anomalies are detected, and exploit resources more effectively when observing suspicious data streams. This strategy becomes particularly advantageous when facing challenging conditions of small $|\delta|$ and m values. Specifically, SA2C-I effectively allocates resources by dispersing them broadly across data streams when the system is regarded in control. This strategy enables quicker anomaly detection when a mean shift does occur. Once an anomaly is identified in a particular data stream, SA2C-I focuses its resources on that data stream while confidently reducing resource allocation to others to maintain the average resource use, $\bar{m}$, the same as the fixed resource level, m, in SA2C-II. These dynamics and impacts on performance will be discussed in greater detail in Section 4.4.3.

(iii): The advantages of both SA2C-I and SA2C-II over random sampling (RS) stem from their use of the A2C algorithm to decide the optimal sampling layout, leading to significant improvements in detection speed under various conditions.

(iv): Comparing SA2C-I and SA2C-II with QH01, which requires continuous observation of all data streams, it is evident that full observability usually yields the best results. However, in challenging scenarios such as when $|\delta|$ is low, SA2C-I and SA2C-II occasionally outperform QH01. There are two possible reasons. First, SA2C focuses on the long-term benefits as described above. So, even when only limited resources are available, SA2C can still effectively utilize the available resources to decide the appropriate sampling scheme for quick anomaly detection. In this way, the full observability, especially after SA2C has identified the highly suspicious data streams, does not help much. Furthermore, unlike QH01, which uses all local statistics to form the global statistic $g(t)$, SA2C uses the top-l scheme, focusing only on the summands of the largest l local statistics, thereby enhancing detection sensitivity when $|\delta|$ is small.

(v): When comparing SA2C-I and SA2C-II with QUANTS, which employs the same local statistics but uses a heuristic sampling approach, it is clear that SA2C-I and SA2C-II generally outperform QUANTS, except in cases where both $|\delta|$ and m are large ($|\delta| = 3, m = 6$). Indeed, the superiority of SA2C is more pronounced when $|\delta|$ and m are small. The first possible reason is that SA2C is conducted with a training scenario that includes a mean shift of magnitude 1, and thus it achieves the optimal performance when real-world shifts closely match this scenario. The second reason is that SA2C effectively utilizes the information of unattended time, while QUANTS does not consider this important information. Lastly, heuristic approaches such as QUANTS tend to always reallocate resources quickly to data streams with higher local statistics even though when the magnitudes of the local statistics are relatively small. In contrast, SA2C

focuses on long-term monitoring strategies, which prevents heuristic resource allocation strategy based solely on transient or minor data fluctuations. SA2C-I further enhances this approach by dynamically adjusting the number of data streams observed, allowing for a more extensive and flexible response to emerging anomalies. When m is small, it becomes very challenging to notice the changes with limited resources and thus it is more important to consider the long-term consequence of the sampling strategy as in SA2C.

4.4.2 Hyperparameter Selection for SA2C Reward Scheme

In this subsection, we evaluate the impact of hyperparameter settings for β and C in the reward scheme. We specifically utilize SA2C-I for this analysis, adhering to the simulation parameters outlined in Section 4.4.1 and set $\delta = 1$.

Table 4.2 ARL_1 with $\beta = 5$ and different C

C	20	100	500	1000	1500
ARL_1	23.18 (1.30)	31.32 (1.85)	34.29 (1.75)	38.00 (1.63)	46.01 (1.93)
$\bar{m}$	11.60	5.95	4.17	2.20	1.69

Table 4.3 ARL_1 with $C = 100$ and different β

β	3	5	8	11
ARL_1	24.07 (1.30)	31.32 (1.85)	37.25 (1.71)	>100
$\bar{m}$	11.43	5.95	2.59	0.70

From Table 4.2, it is evident that when β is held constant and the cost of observing an additional data stream rises, the average number of observed data stream $\bar{m}$ decreases and the detection delay increases. This is a logical outcome as the increasing costs of additional monitoring resources make it less economically feasible compared to the risk of missing an anomaly. Consequently, the agent adjusts its strategy by using less resources, which unfortunately extends

the detection delay. On the other hand, in Table 4.3, when C remains fixed and β increases, the punishment for missing an anomaly detection is reduced. This characteristic encourages the agent to prioritize conserving resources over rapid anomaly detection.

4.4.3 SA2C-I Sampling Layout Analysis

In this subsection, we explore the sampling layout of SA2C-I and investigate how m_t, the number of data streams observed by the agent, varies before and after a mean shift occurs. In particular, we conduct test episodes where $T^0 + T^1$ samples are randomly selected with replacement from historical in-control data. T^0 is chosen from a uniform distribution between 100 and 200, and $T^1 = 150$. The first T^0 samples serve as the in-control training dataset, while a mean shift is randomly introduced into one of the 15 data streams during the final T^1 samples. We then assess the average resource usage, $\bar{m}$, during both in-control and out-of-control phases. Additionally, we compare the performance of SA2C-I against QUANTS under conditions where $\bar{m}$ during the in-control phase matches the m value for QUANTS. We set $\delta = 1$ and $\delta = 3$, and conduct our analysis based on 5,000 replications.

Table 4.4 Resource utilization and ARL_1 comparison during in-control and out-of-control phases by SA2C-I and QUANTS

	SA2C-I			QUANTS		
δ	In control $\bar{m}$	Out of control $\bar{m}$	ARL_1	In control $\bar{m}$	Out of control $\bar{m}$	ARL_1
1	4.0	1.9	42.87 (1.79)	4	4	63.46 (1.24)
	5.0	3.6	35.16 (1.82)	5	5	51.80 (1.39)
	6.0	4.9	32.80 (1.78)	6	6	45.77 (0.75)
3	4.0	2.5	24.39 (1.50)	4	4	22.67 (0.30)
	5.0	3.9	19.45	5	5	15.64

		(1.23)			(0.32)
6.0	5.1	15.76 (1.28)	6	6	10.40 (0.26)

As we can see in Table 4.4, SA2C-I consistently outperforms QUANTS in challenging scenarios when $|\delta|$ and m are small, whereas maintaining comparable performance when $|\delta|$ and m are large. Recall that the comparison was conducted while controlling the average resource usage, $\bar{m}$ by SA2C-I to match the fixed resource usage, m by QUANTS during the in-control stage. On the contrary, during the out-of-control stage, SA2C-I spent much less resources compared to QUANTS. Therefore, the superiority of SA2C-I over QUANTS is evident.

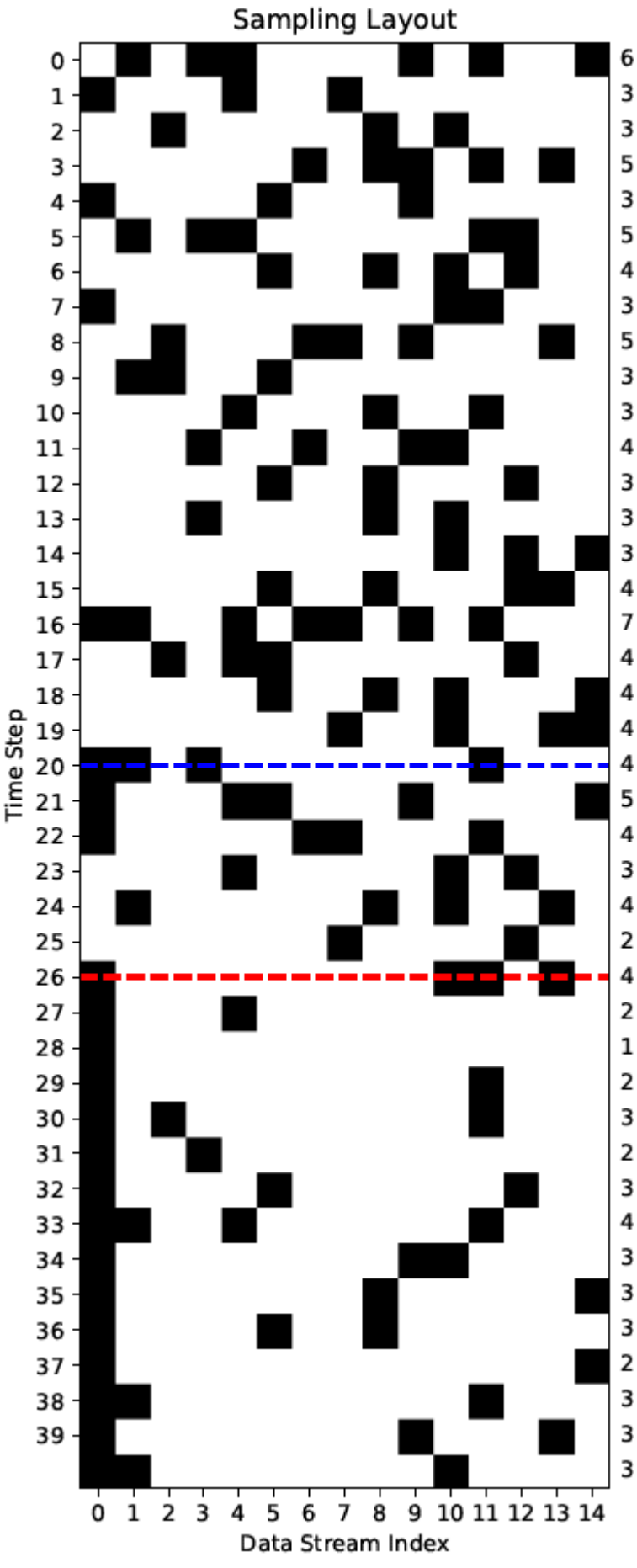

Sampling Layout
Time Step
Data Stream Index
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
6 3 3 5 3 5 4 3 5 3 3 4 3 3 3 4 7 4 4 4 4 5 4 3 4 2 4 2 1 2 3 2 3 4 3 3 3 2 3 3

Fig. 4.2 Sampling layout of the SA2C-I algorithm.

In Figure 4.2, we further present the sampling layout for one test episode before and after a mean shift occurs in data stream 1. Specifically, the mean shift occurs at time step $T = 126$, indicated by a blue dashed line. An alarm is triggered at time step 132, indicated by a red dash line. We analyze the sampling layout for 20 time steps before the shift, from $106 \leq t < 126$ (i.e., in control), for 20 time steps after the shift, from $126 < t \leq 146$ (i.e., out of control). The data streams selected for observation are highlighted in black, and the rightmost column provides a summary of the number of data streams observed at each time step. Like Table 4.4, Figure 4.2 reveals that SA2C-I allocates more resources when the system is in control compared to when it is out of control in general. This result is logical because in the absence of perceived mean shifts, the agent tends to explore extensively using more resources, enabling it to quickly detect an anomaly when it occurs. Conversely, when the system transitions to an out-of-control state and the agent is increasingly certain of a mean shift in a specific data stream (i.e., data stream 1 in this case), it will consider observing other data streams unnecessary and continue exploiting the shifted data stream, as shown in Figure 4.2.

4.4.4 Monitoring Multiple Shifted Data Streams

In Section 4.4.1, we initially established the optimal policy for SA2C, assuming the occurrence of a single mean shift in the system. However, real-world scenarios often involve multiple shifted data streams, so we are interested in reassessing the effectiveness of SA2C-I in these more complex situations. Specifically, we employ the same settings described in Section 4.4.1 to train the optimal policy and then apply it to monitor the scenarios with multiple shifted data streams. The findings are summarized in Table 4.5, where for comparison purposes $\bar{m}$ is rounded up to m.

Table 4.5 shows that applying SA2C-I to situations with multiple shifted streams consistently reduces detection delay as the number of shifted data streams increases. This result suggests that although SA2C-I is trained for monitoring a single shifted stream, it is adept at managing scenarios where multiple data streams exhibit mean shifts. This adaptability may stem from SA2C-I's strategy of identifying and monitoring the m_t most informative data streams. Additionally, in comparison to QUANTS, SA2C-I sometimes underperforms, which is expected given that SA2C-I is trained under the assumption of a single shift, whereas QUANTS does not rely on such an assumption. Despite this, SA2C-I can still effectively detect multiple shifts, even when trained for a different scenario. It's important to note that in practice, shifts are often sparse, making the detection of a single shift particularly challenging, as observed in Table 4.1. Therefore, the decision to train A2C with the assumption of single shifts is justified. If domain knowledge regarding the occurrence of multiple shifts is available, adjusting the A2C algorithm during the training process to account for multiple shifts could enhance its performance in detecting them. Lastly, Table 4.5 demonstrates that an increase in the magnitude of the mean shift, $|\delta|$, or in the number of observable data streams, leads to a decrease in detection delay, which is consistent with our expectations.

Table 4.5 Comparisons of ARL_1 with the standard errors (in parentheses) under multiple shifted data streams

		Two shifted data streams		Three shifted data streams	
$\bar{m}$	δ	SA2C-I	QUANTS	SA2C-I	QUANTS
2	1.0	30.13 (1.28)	41.77 (1.23)	24.91 (1.17)	25.34 (0.40)
	2.0	25.79 (1.16)	19.56 (0.37)	21.80 (1.07)	14.88 (0.33)
	3.0	24.93 (1.14)	12.83 (0.31)	20.44 (1.01)	10.17 (0.24)
6	1.0	24.48	20.91	20.09	12.01

	(1.14)	(0.34)	(1.10)	(0.26)
2.0	16.11	10.07	14.60	8.66
	(0.87)	(0.23)	(0.92)	(0.20)
3.0	13.69	8.74	12.87	7.73
	(0.83)	(0.20)	(0.89)	(0.18)

4.5 Case Study

In this section, we illustrate the real-world application of the proposed SA2C algorithm through a case study in semiconductor manufacturing, where it is critical to detect potential assignable root causes promptly. This application exemplifies the convergence of human-machine collaboration central to Industry 5.0. By enabling operators to interact more effectively with automated monitoring systems, our approach enhances both the efficiency and sustainability of production processes. Operators can prioritize interventions based on the critical insights provided by our algorithm, reducing downtime and resource waste. The dataset used can be accessed from the UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/datasets/SECOM). Following data preprocessing and standardization, the dataset comprises 1,294 in-control samples and 99 out-of-control samples, each consisting of measurements from 393 data streams.

To implement the SA2C-I algorithm, we utilize the bootstrap method as outlined in Section 4.3.1.5 for offline training. We preprocess the data by standardizing each data stream to have a mean of 0 and a standard deviation of 1 when the process is in control. For offline training, as discussed in Section 4.3.1.5, we bootstrap the data to form the training samples for each training episode. During this process, a random mean shift of either +1 or -1 is introduced into one of the 393 data streams. Our analysis confirmed that the mean shift of the out-of-control (OC) data is at least 1 compared to the in-control (IC) data, validating our use of a +1/-1 mean shift in the original

IC data to simulate real OC conditions. This approach allows us to maintain a controlled environment with a known ground truth, facilitating a clearer evaluation and understanding of our method's performance against benchmarks. Using actual OC data would complicate the analysis due to the variety of shift types and limited data samples (only 99 OC instances), hindering comprehensive performance evaluation. Therefore, our method uses a consistent, simulated shift to demonstrate its effectiveness and versatility in anomaly detection. Following training, the SA2C-I algorithm learns an optimal policy and determines the monitoring threshold for the global statistic such that the in-control ARL is 370. To evaluate performance in online monitoring, we generate new testing samples by inserting a mean shift of +1/-1 to one of the 393 data streams from time step 0. We set $l = 1, k = 0.5$ for both the SA2C-I and QUANTS method. Similar to Table 4.1, for fair comparison purposes, $\bar{m}$ is rounded up to m. Then, the result of ARL_1 with different m values are detailed in Table 4.6, with standard deviations provided in parentheses. According to Table 4.6, SA2C-I consistently outperforms QUANTS across all scenarios, particularly when $\bar{m}$ is small. This finding aligns with the results from Section 4.4.

Table 4.7 shows the average computational times for SA2C-I and QUANTS during online monitoring as they update the sampling layout at each time step. The time difference primarily results from processing the neural network output and sampling without replacement according to the probability distribution p_t for N_t times by the SA2C-I algorithm. Despite these additional computations, the results demonstrate that SA2C-I operates efficiently enough to be deployed in real-time applications.

Table 4.6 Comparisons of ARL_1 with the standard errors (in parentheses) for the real case study

m	QUANTS	SA2C-I
10	48.11	31.60

	(0.49)	(1.24)
20	35.72	27.53
	(0.41)	(1.18)
30	30.49	25.09
	(0.40)	(1.17)

Table 4.7 Average computation time during online monitoring for SA2C-I and QUANTS

	SA2C-I	QUANTS
Time (sec)	0.91	0.28

4.6 Conclusion

In this article, we introduce a DRL-based dynamic sampling and monitoring framework designed for high-dimensional processes, particularly when only partial observations are available at each acquisition time. We propose to seamlessly integrate an advantage actor-critic (A2C) algorithm with statistical process control (SPC). Initially, we construct a local statistic for each data stream, employing the A2C agent to develop a dynamic sampling strategy aimed at prompt anomaly detection. Subsequently, a global monitoring statistic is formulated by summing the top-l local statistics to determine if the system is out of control. Finally, guided by the optimal policy derived from DRL, we determine an appropriate threshold to manage the in-control ARL. Compared to existing methods, our proposed SA2C algorithm offers several unique advantages: (i) In contrast to previous heuristic approaches (Cheng & Thaga, 2005; Liu et al., 2015; Xian et al., 2018; Ye et al., 2022; Ye & Liu, 2022) that primarily focus on short-term gains from sampling strategies in a myopic manner, the SA2C algorithm adopts a strategy that emphasizes long-term discounted rewards and incorporates unattended time into its decision-making process, which effectively balances exploitation and exploration. (ii) Unlike prior research, SA2C is capable of dynamically determining not only the optimal number of data streams to observe but also identifying which specific data streams require surveillance. This enhancement marks a significant boost in

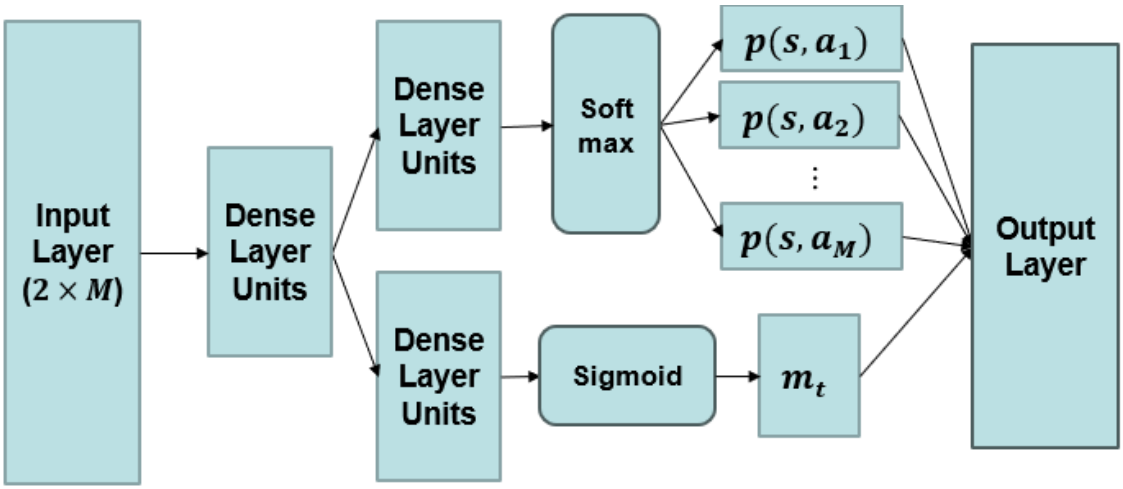
adaptability and efficiency in monitoring strategies. (iii) The integration of a deep neural network framework addresses scalability issues, enabling the simultaneous monitoring of high-dimensional data streams. This advancement significantly enhances the adaptability and efficiency of industrial monitoring strategies, which is crucial for managing large-scale data environments robustly.

In conclusion, the implementation of our decision aid model presents significant managerial implications for industry leaders seeking to optimize production systems. This model not only facilitates the prioritization and real-time monitoring of critical data streams but also enables managers to make informed, proactive decisions that can significantly reduce operational costs and improve efficiency. Furthermore, the adaptability of our approach allows for its application across various sectors, providing managers with a versatile tool tailored to meet specific operational needs and challenges. We encourage industry practitioners to consider the potential of such advanced monitoring solutions in strategic planning and operational management to harness the full benefits of Industry 4.0 and 5.0 technologies.

There are several future research directions. Firstly, while SA2C has proven effective in managing nonparametric data distributions, it operates under the assumption that each data stream is independent and identically distributed over time. Further research is required to adapt the SA2C framework to handle autocorrelated data streams. Additionally, the current framework does not consider spatial correlation between data streams and ignores the cost associated with reallocating resources. Such improvements will address the oversight of resource reallocation costs and enhance the model's applicability to complex industrial scenarios where human decision-making and machine efficiency are intertwined. Such advancements will further align with Industry 5.0's emphasis on sustainable, resilient, and human-centric manufacturing systems, enabling more nuanced and effective control of industrial processes.

4.7 Appendix

Graphical Representation of Actor Network



Chapter 5 Summary and Future Work

Online monitoring has been considered an important topic in data analytics and decision making in Internet of Things, such as smart manufacturing systems. The contribution of the research work to date can be summarized as follows: (1) Established an adaptive monitoring strategy in process monitoring under resource constraints based on Q-learning; (2) Further established a high dimensional monitoring strategy utilizing deep reinforcement learning to achieve continuous state space, specifically double dueling deep Q network, with partial observations; (3) Further leveraged the scalability of our high-dimensional monitoring strategy by introducing a continuous action space that can dynamically selecting both an optimal number and the most informative data streams for observation, utilizing the Advantage Actor Critic Network algorithm.

Looking ahead, I aim to address the limitations of our current models, which are confined to 1-D sensor reallocation. Our existing framework fails to consider the spatial correlation among data streams and neglects the costs of resource reallocation. To overcome these challenges, developing a pathwise online monitoring system using Convolutional Deep Reinforcement Learning would be advantageous. This system would not only account for path movements but also integrate the spatial structures and regional influences of anomalies, thereby enhancing the accuracy and efficiency of our anomaly detection capabilities. Adopting this method promises to revolutionize the field of intelligent monitoring, leveraging advanced technologies to foster a deeper understanding and more effective management of complex systems in real time.

References

[1] P. K. Rao, O. F. Beyca, Z. Kong, S. T. S. Bukkapatnam, K. E. Case, and R. Komanduri, “A graph-theoretic approach for quantification of surface morphology variation and its application to chemical mechanical planarization process,” *IIE Trans. (Institute Ind. Eng.*, vol. 47, no. 10, pp. 1088–1111, 2015, doi: 10.1080/0740817X.2014.1001927.

[2] I. Ahmed, A. Dagnino, and Y. Ding, “Unsupervised Anomaly Detection Based on Minimum Spanning Tree Approximated Distance Measures and its Application to Hydropower Turbines,” *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 2, 2019, doi: 10.1109/TASE.2018.2848198.

[3] R. J. Meuth, E. W. Saad, D. C. Wunsch, and J. Vian, “Adaptive task allocation for search area coverage,” 2009. doi: 10.1109/TEPRA.2009.5339643.

[4] K. Liu, Y. Mei, and J. Shi, “An Adaptive Sampling Strategy for Online High-Dimensional Process Monitoring,” *Technometrics*, vol. 57, no. 3, pp. 305–319, 2015, doi: 10.1080/00401706.2014.947005.

[5] K. Liu, X. Zhang, and J. Shi, “Adaptive sensor allocation strategy for process monitoring and diagnosis in a bayesian network,” *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 2, 2014, doi: 10.1109/TASE.2013.2287101.

[6] M. Nabhan, Y. Mei, and J. Shi, “Correlation-based dynamic sampling for online high dimensional process monitoring,” *J. Qual. Technol.*, vol. 53, no. 3, pp. 289–308, 2021, doi: 10.1080/00224065.2020.1726717.

[7] X. Xian, J. Li, and K. Liu, “Causation-Based Monitoring and Diagnosis for Multivariate Categorical Processes with Ordinal Information,” *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 2, 2019, doi: 10.1109/TASE.2018.2873365.

[8] J. Li, K. Liu, and X. Xian, “Causation-based process monitoring and diagnosis for multivariate categorical processes,” *IISE Trans.*, vol. 49, no. 3, 2017, doi: 10.1080/0740817X.2016.1241455.

[9] X. Xian, A. Wang, and K. Liu, “A Nonparametric Adaptive Sampling Strategy for Online Monitoring of Big Data Streams,” *Technometrics*, vol. 60, no. 1, pp. 14–25, 2018, doi: 10.1080/00401706.2017.1317291.

[10] P. Qiu and D. Hawkins, “A rank-based multivariate CUSUM procedure,” *Technometrics*, vol. 43, no. 2, 2001, doi: 10.1198/004017001750386242.

[11] P. Qiu and D. Hawkins, “A nonparametric multivariate cumulative sum procedure for detecting shifts in all directions,” *J. R. Stat. Soc. Ser. D Stat.*, vol. 52, no. 2, pp. 151–164, 2003, doi: 10.1111/1467-9884.00348.

[12] P. Qiu and Z. Li, “On nonparametric statistical process control of univariate processes,” *Technometrics*, vol. 53, no. 4, 2011, doi: 10.1198/TECH.2011.10005.

[13] H. Ye and K. Liu, “A Generic Online Nonparametric Monitoring and Sampling Strategy for High-Dimensional Heterogeneous Processes,” *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 3, pp. 1503–1516, 2022, doi: 10.1109/TASE.2022.3146391.

[14] H. Ye *et al.*, “Online nonparametric monitoring of heterogeneous data streams with partial observations based on Thompson sampling,” *IISE Trans.*, vol. 55, no. 4, pp. 392–404, 2023, doi: 10.1080/24725854.2022.2039423.

[15] G. J. Ross, D. K. Tasoulis, and N. M. Adams, “Nonparametric monitoring of data streams for changes in location and scale,” *Technometrics*, vol. 53, no. 4, 2011, doi: 10.1198/TECH.2011.10069.

[16] A. Wang, X. Xian, F. Tsung, and K. Liu, “A spatial-adaptive sampling procedure for online

monitoring of big data streams," *J. Qual. Technol.*, vol. 50, no. 4, pp. 329–343, 2018, doi: 10.1080/00224065.2018.1507560.

[17] R. S. Sutton and A. G. Barto, "Reinforcement Learning, Second Edition: An Introduction - Complete Draft," *MIT Press*, 2018.

[18] A. L. Strehl and M. L. Littman, "An analysis of model-based Interval Estimation for Markov Decision Processes," *J. Comput. Syst. Sci.*, vol. 74, no. 8, 2008, doi: 10.1016/j.jcss.2007.08.009.

[19] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM SIGART Bull.*, vol. 2, no. 4, 1991, doi: 10.1145/122344.122377.

[20] E. S. PAGE, "CONTINUOUS INSPECTION SCHEMES," *Biometrika*, vol. 41, no. 1–2, 1954, doi: 10.1093/biomet/41.1-2.100.

[21] Y. Mei, "Efficient scalable schemes for monitoring a large number of data streams," *Biometrika*, vol. 97, no. 2, pp. 419–433, 2010, doi: 10.1093/biomet/asq010.

[22] H. M. Ngai and J. Zhang, "Multivariate cumulative sum control charts based on projection pursuit," *Stat. Sin.*, vol. 11, no. 3, 2001.

[23] K. L. Tsui, S. W. Han, W. Jiang, and W. H. Woodall, "A review and comparison of likelihood-based charting methods," *IIE Transactions (Institute of Industrial Engineers)*, vol. 44, no. 9. 2012. doi: 10.1080/0740817X.2011.582476.

[24] M. Riaz, M. Abid, A. Shabbir, H. Z. Nazir, Z. Abbas, and S. A. Abbasi, "A non-parametric double homogeneously weighted moving average control chart under sign statistic," *Qual. Reliab. Eng. Int.*, vol. 37, no. 4, 2021, doi: 10.1002/qre.2812.

[25] C. Zhang and S. C. H. Hoi, "Partially observable multi-sensor sequential change detection: A combinatorial multi-armed bandit approach," 2019. doi: 10.1609/aaai.v33i01.33015733.

[26] S. W. Cheng and K. Thaga, “Max-CUSUM chart for autocorrelated processes,” in *Statistica Sinica*, 2005, vol. 15, no. 2, pp. 527–546.

[27] S. W. Hasinoff, “Reinforcement Learning for Problems with Hidden State,” *Univ. Toronto, Tech. Rep.*, 2003.

[28] M. L. Littman, “Memoryless policies : theoretical limitations and practical results,” *From Anim. to Animat. 3 Proc. Third Int. Conf. Simul. Adapt. Behav.*, no. April, 1994.

[29] Y. Mei, “Quickest detection in censoring sensor networks,” 2011. doi: 10.1109/ISIT.2011.6034390.

[30] H. G. Svensson, E. Bjerrum, C. Tyrchan, O. Engkvist, and M. H. Chehreghani, “Autonomous Drug Design with Multi-armed Bandits.,” *arXiv*, 2022. https://doi.org/10.48550/arXiv.2207.01393

[31] A. Maroti, “RBED: Reward Based Epsilon Decay.,” *arXiv.*, 2019. https://doi.org/10.48550/arXiv.1910.13701

[32] C. L. Beck and R. Srikant, “Error bounds for constant step-size Q-learning,” *Syst. Control Lett.*, vol. 61, no. 12, pp. 1203–1208, 2012, doi: 10.1016/j.sysconle.2012.08.014.

[33] S. Yang, M. Santillana, and S. C. Kou, “Accurate estimation of influenza epidemics using Google search data via ARGO,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 112, no. 47, 2015, doi: 10.1073/pnas.1515373112.

[34] S. Aberkane and M. Elarbi, “Deep Reinforcement Learning for Real-world Anomaly Detection in Surveillance Videos,” 2019. doi: 10.1109/ISPA48434.2019.8966795.

[35] R. F. Mansour, J. Escorcia-Gutierrez, M. Gamarra, J. A. Villanueva, and N. Leal, “Intelligent video anomaly detection and classification using faster RCNN with deep reinforcement learning model,” *Image Vis. Comput.*, vol. 112, 2021, doi:

10.1016/j.imavis.2021.104229.

[36] K. Sethi, R. Kumar, N. Prajapati, and P. Bera, “Deep Reinforcement Learning based Intrusion Detection System for Cloud Infrastructure,” 2020. doi: 10.1109/COMSNETS48256.2020.9027452.

[37] L. Zhang *et al.*, “A Hidden Attack Sequences Detection Method Based on Dynamic Reward Deep Deterministic Policy Gradient,” *Secur. Commun. Networks*, vol. 2022, 2022, doi: 10.1155/2022/1488344.

[38] H. Alavizadeh, H. Alavizadeh, and J. Jang-Jaccard, “Deep Q-Learning Based Reinforcement Learning Approach for Network Intrusion Detection,” *Computers*, vol. 11, no. 3, 2022, doi: 10.3390/computers11030041.

[39] W. Zhang and Y. Mei, “Bandit Change-Point Detection for Real-Time Monitoring High-Dimensional Data Under Sampling Control,” *Technometrics*, vol. 65, no. 1, pp. 33–43, Apr. 2023, doi: 10.1080/00401706.2022.2054861.

[40] X. Zan, D. Wang, and X. Xian, “Spatial Rank-Based Augmentation for Nonparametric Online Monitoring and Adaptive Sampling of Big Data Streams,” *Technometrics*, vol. 65, no. 2, pp. 243–256, Nov. 2023, doi: 10.1080/00401706.2022.2143903.

[41] J. Guo, H. Yan, and C. Zhang, “A Bayesian Partially Observable Online Change Detection Approach with Thompson Sampling,” *Technometrics*, vol. 65, no. 2, pp. 179–191, Oct. 2023, doi: 10.1080/00401706.2022.2127914.

[42] X. Xian, C. Zhang, S. Bonk, and K. Liu, “Online monitoring of big data streams: A rank-based sampling algorithm by data augmentation,” *J. Qual. Technol.*, vol. 53, no. 2, pp. 135–153, 2021, doi: 10.1080/00224065.2019.1681924.

[43] W. Fedus *et al.*, “Revisiting fundamentals of experience replay,” in *37th International*

Conference on Machine Learning, ICML 2020, 2020, vol. PartF168147-4.

[44] K. Sethi, R. Kumar, D. Mohanty, and P. Bera, "Robust Adaptive Cloud Intrusion Detection System Using Advanced Deep Reinforcement Learning," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2020, vol. 12586 LNCS, pp. 66–85. doi: 10.1007/978-3-030-66626-2_4.

[45] Y. F. Hsu and M. Matsuoka, "A Deep Reinforcement Learning Approach for Anomaly Network Intrusion Detection System," 2020. doi: 10.1109/CloudNet51028.2020.9335796.

[46] C. Kim and J. S. Park, "Designing online network intrusion detection using deep auto-encoder Q-learning," *Comput. Electr. Eng.*, vol. 79, 2019, doi: 10.1016/j.compeleceng.2019.106460.

[47] C. Zhong, M. C. Gursoy, and S. Velipasalar, "Deep actor-critic reinforcement learning for anomaly detection," 2019. doi: 10.1109/GLOBECOM38437.2019.9013223.

[48] Y. Li and J. Wu, "Low Latency Cyberattack Detection in Smart Grids with Deep Reinforcement Learning," *SSRN Electron. J.*, 2022, doi: 10.2139/ssrn.4019864.

[49] G. I. Allen, L. Grosenick, and J. Taylor, "A Generalized Least-Square Matrix Decomposition," *J. Am. Stat. Assoc.*, vol. 109, no. 505, pp. 145–159, Jan. 2014, doi: 10.1080/01621459.2013.852978.

[50] C. Zhang, H. Yan, S. Lee, and J. Shi, "Weakly correlated profile monitoring based on sparse multi-channel functional principal component analysis," *IISE Trans.*, vol. 50, no. 10, pp. 878–891, 2018, doi: 10.1080/24725854.2018.1451012.

[51] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-Learning," 2016. doi: 10.1609/aaai.v30i1.10295.

[52] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Frcitas, "Dueling

Network Architectures for Deep Reinforcement Learning," in *33rd International Conference on Machine Learning, ICML 2016*, 2016, vol. 4.

[53] J. Schmidhuber, "Deep Learning in neural networks: An overview," *Neural Networks*, vol. 61. pp. 85–117, 2015. doi: 10.1016/j.neunet.2014.09.003.

[54] V. M. K. K. D. S. A. G. I. A. D. W. M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *Deep Reinf. Learn. Fundam. Res. Appl.*, 2020.

[55] D. Ivanov, "The Industry 5.0 framework: viability-based integration of the resilience, sustainability, and human-centricity perspectives," *Int. J. Prod. Res.*, vol. 61, no. 5, pp. 1683–1695, 2023, doi: 10.1080/00207543.2022.2118892.

[56] M. A. Hayes and M. A. Capretz, "Contextual anomaly detection framework for big sensor data," *J. Big Data*, vol. 2, no. 1, 2015, doi: 10.1186/s40537-014-0011-y.

[57] F. Esmaeili, E. Cassie, H. P. T. Nguyen, N. O. V. Plank, C. P. Unsworth, and A. Wang, "Anomaly Detection for Sensor Signals Utilizing Deep Learning Autoencoder-Based Neural Networks," *Bioengineering*, vol. 10, no. 4, 2023, doi: 10.3390/bioengineering10040405.

[58] H. Dong, Z. Ding, and S. Zhang, *Deep reinforcement learning: Fundamentals, research and applications*. 2020. doi: 10.1007/978-981-15-4095-0.

[59] B. Rolf, I. Jackson, M. Müller, S. Lang, T. Reggelin, and D. Ivanov, "A review on reinforcement learning algorithms and applications in supply chain management," *International Journal of Production Research*, vol. 61, no. 20. pp. 7151–7179, 2023. doi: 10.1080/00207543.2022.2140221.

[60] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016, doi: 10.1038/nature16961.

[61] K. F. Muteba, K. Djouani, and T. O. Olwal, “Deep reinforcement learning based resource allocation for narrowband cognitive radio-Iot systems,” in *Procedia Computer Science*, 2020, vol. 175, pp. 315–324. doi: 10.1016/j.procs.2020.07.046.

[62] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” 2000.

[63] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” 2017, doi: 10.48550/arXiv.1707.06347.

[64] V. R. Konda and J. N. Tsitsiklis, “On actor-critic algorithms,” *SIAM J. Control Optim.*, vol. 42, no. 4, 2003, doi: 10.1137/S0363012901385691.

[65] V. Mnih *et al.*, “Asynchronous methods for deep reinforcement learning,” in *33rd International Conference on Machine Learning, ICML 2016*, 2016, vol. 4.

[66] J. J. Lee, Y. Hur, S. H. Kim, and J. R. Wilson, “Monitoring nonlinear profiles using a wavelet-based distribution-free CUSUM chart,” *Int. J. Prod. Res.*, vol. 50, no. 22, pp. 6574–6594, 2012, doi: 10.1080/00207543.2012.655865.

[67] M. A. Graham, A. Mukherjee, and S. Chakraborti, “Design and implementation issues for a class of distribution-free Phase II EWMA exceedance control charts,” *Int. J. Prod. Res.*, vol. 55, no. 8, pp. 2397–2430, 2017, doi: 10.1080/00207543.2016.1249428.

[68] S. Hou and K. Yu, “A non-parametric CUSUM control chart for process distribution change detection and change type diagnosis,” *Int. J. Prod. Res.*, vol. 59, no. 4, pp. 1166–1186, 2021, doi: 10.1080/00207543.2020.1721588.

[69] C. Li, A. Mukherjee, Q. Su, and M. Xie, “Optimal design of a distribution-free quality control scheme for cost-efficient monitoring of unknown location,” *Int. J. Prod. Res.*, vol. 54, no. 24, pp. 7259–7273, 2016, doi: 10.1080/00207543.2016.1173254.

[70] D. Liu, H. Kim, S. H. Kim, T. Kim, D. Lee, and Y. Xie, “Distribution-free multivariate time-series monitoring with analytically determined control limits,” *Int. J. Prod. Res.*, vol. 61, no. 20, pp. 6960–6977, 2023, doi: 10.1080/00207543.2022.2140364.

[71] L. Liu, F. Tsung, and J. Zhang, “Adaptive nonparametric CUSUM scheme for detecting unknown shifts in location,” *Int. J. Prod. Res.*, vol. 52, no. 6, pp. 1592–1606, 2014, doi: 10.1080/00207543.2013.812260.

[72] G. Celano and S. Chakraborti, “A distribution-free Shewhart-type Mann–Whitney control chart for monitoring finite horizon productions,” *Int. J. Prod. Res.*, vol. 59, no. 20, pp. 6069–6086, 2021, doi: 10.1080/00207543.2020.1802079.

[73] Q. T. Nguyen, K. P. Tran, H. L. Heuchenne, T. H. Nguyen, and H. Du Nguyen, “Variable sampling interval Shewhart control charts for monitoring the multivariate coefficient of variation,” *Appl. Stoch. Model. Bus. Ind.*, vol. 35, no. 5, pp. 1253–1268, 2019, doi: 10.1002/asmb.2472.

[74] J. Yue and L. Liu, “Multivariate nonparametric control chart with variable sampling interval,” *Appl. Math. Model.*, vol. 52, pp. 603–612, 2017, doi: 10.1016/j.apm.2017.08.005.

[75] M. R. Reynolds and K. Kim, “Multivariate control charts for monitoring the process mean and variability using sequential sampling,” *Seq. Anal.*, vol. 26, no. 3, pp. 283–315, 2007, doi: 10.1080/07474940701404898.

[76] H. Ye, Z. Zheng, J. R. C. Cheng, B. Hable, and K. Liu, “Online monitoring of high-dimensional asynchronous and heterogeneous data streams for shifts in location and scale,” *Int. J. Prod. Res.*, vol. 62, no. 3, pp. 720–736, 2024, doi: 10.1080/00207543.2023.2172474.

[77] X.-Y. Liu *et al.*, “FinRL: A Deep Reinforcement Learning Library for Automated Stock Trading in Quantitative Finance,” *SSRN Electron. J.*, 2021, doi: 10.2139/ssrn.3737859.

[78] M. Neves and P. Neto, “Deep reinforcement learning applied to an assembly sequence planning problem with user preferences,” *Int. J. Adv. Manuf. Technol.*, vol. 122, no. 11–12, pp. 4235–4245, 2022, doi: 10.1007/s00170-022-09877-8.

[79] J. Schrittwieser *et al.*, “Mastering Atari, Go, chess and shogi by planning with a learned model,” *Nature*, vol. 588, no. 7839, pp. 604–609, 2020, doi: 10.1038/s41586-020-03051-4.

[80] Y. Shao, R. Li, B. Hu, Y. Wu, Z. Zhao, and H. Zhang, “Graph Attention Network-Based Multi-Agent Reinforcement Learning for Slicing Resource Management in Dense Cellular Network,” *IEEE Trans. Veh. Technol.*, vol. 70, no. 10, 2021, doi: 10.1109/TVT.2021.3103416.