Data Engineering and Applications

Research Paper

Optimization of Real-Time Big Data Stream Processing Systems Based on Deep Reinforcement Learning

Mariusz Ostrowski[1], Jarosław Bąk[2,*] and Seweryn Sokołowski[2]

[1] Faculty of Information Technology, University of Warmia and Mazury, Olsztyn, 10-719, Poland
[2] Faculty of Computer Science and Information Technology, West Pomeranian University of Technology, Szczecin, 70-310, Poland
*Corresponding author: jaroslaw.b@zut.edu.pl

Abstract. The large number of high-speed data sources that have emerged now include the Industrial Internet of Things, financial services, online platforms, etc., all of which require real-time processing and analysis. To address the latency, scalability, and adaptability issues of traditional big data architectures under fluctuating workloads, this paper proposes a dynamic optimization framework based on Deep Reinforcement Learning (DRL). Resource allocation and task scheduling in the closed-loop feedback system are coordinated using a distributed stream processing core and a scalable Deep Reinforcement Learning (DRL) agent. Experiments were conducted on Apache Flink and Yahoo!, using a small cluster of ten nodes. New York taxi dataset and streaming suite. The above experiments show that the maximum throughput of the DRL-based framework is 108.2 thousand events per second, which is 18.7% higher than the rule-based and model-driven baseline. The framework has a 97.1% SLA compliance rate, a median event latency of 42 milliseconds, a peak CPU utilization of approximately 54%, and can recover from node failures within 28.5 seconds. In addition to the previous research, ablation and sensitivity analyzes were conducted to examine the impact of reward function design on the selection of key system parameters. The aforementioned research indicates that Deep Reinforcement Learning (DRL) can be used for intelligent control in distributed flow environments. By avoiding handcrafted methods and fixed rules, DRL can improve the system's real-time response speed and resource utilization efficiency.

Keywords: *Real-Time Stream Processing, Resource Optimization, Distributed Systems, Fault Tolerance, Cluster Scheduling*

Received on 24 August 2024, Accepted on 28 December 2024, Published on 18 January 2025

Introduction

The Internet of Things (IoT), social media, industrial sensors, financial transactions, and intelligent transportation systems [1] are real-time data streams resulting from the extensive development in many industries. If large volumes of high-speed data streams are not handled, old big data systems will become slow and difficult to scale [2]. Real-time stream processing systems, designed to ingest, analyze, and react to data promptly, are becoming critical for enabling dynamic decision-making and situational awareness across domains [3]. Due to changes in application requirements, there is an increasing demand for more fine-grained analysis and adaptive resource allocation to cope with fluctuating load conditions and heterogeneous environments. The current rule-based static resource management model cannot meet this demand [4]. It is necessary to use intelligent optimization frameworks to manage resources and dynamically balance performance [5]. With the development of artificial intelligence, large-scale data platforms have recently begun using machine learning. On the other hand, reinforcement learning (RL) has performed well in adaptive control of complex unstable systems [6]. The high dimensionality of system states, handling long-term dependencies, and ensuring stable convergence in distributed environments still hinder the widespread application of Deep Reinforcement Learning (DRL) in real-time big data stream processing [7]. These issues have been addressed in past research. As stream workloads

become more complex, strong support for the scalability, stability, and generalization capabilities of production environments have also become increasingly important [8].

Deep reinforcement learning combines the learning capabilities of deep neural networks with adaptive optimization features, providing new methods for real-time data stream management [9]. Through continuous observation of the environment, it can learn the best resource allocation and scheduling methods to cope with changes in workload or unforeseen operational issues. Reinforcement learning can be used for dynamic response [10]. Deep Reinforcement Learning (DRL) is very suitable for the large-scale, multi-dimensional state-action space of distributed stream processing systems compared to heuristic methods. DRL also considers long-term rewards, not just immediate performance improvements [11]. Whether in cloud environments or edge environments, reinforcement learning outperforms previous algorithms in resource scheduling, load balancing, and quality of service adjustments [12]. To scale up large-scale policy learning, some recent studies have explored hierarchical and multi-agent reinforcement learning, while other research has used hybrid models to enhance robustness under uncertainty and partial observability [13]. Currently, many issues are still found, including high computational costs and training time, lack of effective DRL controllers and integration mechanisms with real-time analysis platforms, and difficulty in adapting to environmental changes in real-time [14]. To ensure stability, interpretability, and safety, the practical application of these systems in production environments requires careful design [15]. To address the aforementioned issues, new methods are being developed to create DRL systems and conduct comprehensive testing of their performance in real-world environments or applications [16].

This paper studies how to optimize real-time big data stream processing systems based on deep reinforcement learning. Here, a new framework based on deep reinforcement learning is proposed for making intelligent and adaptive decisions in task scheduling and resource allocation within a distributed streaming environment. This paper discusses in detail the development of algorithms, system integration, and the simulation of real and synthetic workloads. We made the following major contributions: (1) Developing and creating scalable optimization algorithms based on DRL to meet real-time stream processing demands; (2) Integrating the learning module into a significant "big data analytics platform"; (3) Conducting extensive empirical analysis on scalability, robustness, and ablation studies; (4) Discussing theoretical insights and practical deployment considerations. The other sections of this paper are as follows. Section 2: Background and Related Work. Section 3 discusses the construction and structure of the new system. Section 4: Experimental Results and Analysis. Section 5 is the conclusion of this study, as well as future directions.

Background

Current Issues in Stream Processing

The environment of modern information systems is undergoing significant changes due to the development of real-time data sources and large-scale online platforms (such as IoT sensors and extensive online platforms) [17]. Real-time stream processing technology provides fast business intelligence and low-latency analytics to help enterprises derive useful information from continuously generated data [18]. Many advancements have been made, but several significant issues still need to be addressed before widespread and reliable application.

The uneven distribution and intensity of workloads is another issue. The system may suddenly hit a bottleneck and slow down [19]. The data volume may become unstable for various reasons, such as changes in user behavior or other external factors affecting the company. The old static resource allocation method cannot adapt to changes in a timely manner, leading to underutilization or frequent overloads. The lack of adaptability is particularly challenging in distributed deployments and multi-tenant environments; on the other hand, analytical tasks and multiple operators need to simultaneously use the same limited computing resources.

Real-time applications such as financial transactions and networked physical infrastructure require low latency and high Quality of Service (QoS) [20]. Due to service interruptions or delayed delivery of goods, it may result in financial losses and other damages. Ensuring end-to-end latency is not feasible. The system should be fast and fault-tolerant, capable of maintaining consistency during hardware or software failures, and easy to scale. Based on the above requirements, it can be concluded that there are currently no intelligent, context-aware

optimization methods capable of effectively handling the highly variable operational conditions of large-scale data stream processing.

Deep Reinforcement Learning Fundamentals

Deep Reinforcement Learning (DRL) is a top component of machine learning, integrating the trial-and-error learning concept of reinforcement learning with the powerful function approximation capabilities of deep neural networks [21]. In standard reinforcement learning (RL), the agent maximizes its cumulative reward by altering its interactions with the unstable environment. Deep Reinforcement Learning (DRL) uses deep neural networks to handle large-scale or continuous state-action spaces. This enables DRL to make high-level decisions in complex environments that traditional tabular reinforcement learning cannot handle.

DQN, DDPG, and Actor-Critic architectures are the first deep reinforcement learning models; these models can more conveniently handle robotics and resource allocation problems [22]. When dealing with long-term credit allocation problems, the success of DRL mainly depends on its generalization ability based on relatively few experiences, its ability to learn complex time series connections, and its effective application in handling long-term credit allocation problems. These characteristics must be present in a stream processing environment because the system state is high-dimensional and unstable [23]. When data stream or system constraints change, the DRL agent can learn a good strategy through the system feedback loop to adjust resource allocation and control measures.

It is worth noting that due to the flexibility of DRL, it can be used to incorporate domain-specific prior knowledge (such as operator dependencies or workload patterns) to improve policy convergence and learning efficiency. Deep Reinforcement Learning (DRL) shows promise, but there are still issues that need to be addressed. Issues such as low sample efficiency, unstable training, and high computational costs have all received active research attention during the transition from simulated environments to real-world distributed systems [24]. Currently, a significant amount of funding and effort is being invested in innovation to meet the ongoing demand for the continuous optimization of large-scale, mission-critical data processing facilities.

Related Work Review and Summary

Machine learning—especially deep reinforcement learning (DRL)—can help stream processing systems improve, but there has been little research on this approach [25]. Early work using supervised learning can predict load changes or system issues, but it often cannot handle nonlinear changes or unexpected events. Compared to rule-based or heuristic strategies, DRL can significantly improve system robustness and resource savings because they do not rely on models [26]. In order to achieve real-time, policy-driven responses to operational conditions, the best current solution typically involves placing DRL agents in the scheduling or load balancing modules of stream processing platforms.

On the other hand, other types of artificial intelligence, such as Bayesian optimization and meta-learning, can be combined with deep reinforcement learning to address issues of data scarcity or unstable working hours. Through transfer learning or online adaptation, the aforementioned frameworks can quickly adapt to new environments, thereby reducing the convergence period and improving the system's generalization ability. In some studies, multi-agent reinforcement learning (MARL) enables them to cooperate or compete in different application pipelines. However, the widespread application in production is still rare. Many systems face issues with algorithmic cost, interpretability, and integration complexity [27].

In summary, the recent developments in Deep Reinforcement Learning (DRL) have provided new methods to address long-standing issues such as real-time response, scalability, and adaptability. Although some progress has been made, combining DRL optimization with new streaming platforms still requires extensive practical experiments to develop new system designs.

https://doi.org/10.64972/dea.2025.v4i1.1939d:113-126

DRL-Based System Architecture and Implementation

Algorithmic Framework and Workflow Design

This is a deep reinforcement learning (DRL) agent based on our optimization framework, suitable for dynamic, high-dimensional distributed stream processing environments. The agent establishes a closed-loop learning and control system. The system monitors the system state, acts according to the policy, and adjusts the policy based on feedback-driven rewards.

At each time point t, the system state is a vector $\mathbf{s}_t$, containing various metrics such as CPU utilization, memory usage, queue backlog, and data arrival rate. In order to change the system's behavior, the DRL agent selects an action a_t from the predefined action space $\mathcal{A}$. This can be achieved by changing the operator parallelism or data partitioning strategy. Then, in the new state $\mathbf{s}_{t+1}$, calculate a scalar reward r_t to determine whether the current action is good or bad.

The purpose of the agent is to find an optimal policy π^* that maximizes the expected sum of discounted rewards over time:

$$\pi^* = arg \max_{\pi} \mathbb{E}\left[\sum_t \gamma^t r_t\right] \quad \text{Eq.(1)}$$

Here, γ (with $0 < \gamma < 1$) serves as the discount factor, balancing the tradeoff between shortterm and long-term gains.

Policy learning uses a deep neural network with θ parameters to simulate the value function. The Bellman equation serves as the basis for the regular loss function used to update the network parameters:

$$L(\theta) = \left(r + \gamma \max_{a'} Q_{\theta^-}(\boldsymbol{s}', a') - Q_\theta(\boldsymbol{s}, a)\right)^2 \quad \text{Eq.(2)}$$

The three components of the reward function are weighted as follows: throughput, delay and resource consumption improvements:

$$r_t = \alpha_1 \Delta\ Throughput_t - \alpha_2 \Delta\ Latency_t \quad \text{Eq.(3)}$$

Here, α_1 and α_2 are weight parameters that can be tuned based on system priorities.

The agent stores past state transitions $(\mathbf{s}_t, a_t, r_t, \mathbf{s}_{t+1})$ in the experience replay buffer to ensure training stability. Off-policy updates and stable training can be conducted in the presence of system instability and environmental fluctuations. Figure 1 shows a visual representation of the optimization process based on DRL. It shows how the neural policy agent acquires and analyzes system metrics to drive closed-loop adaptive operational decisions.

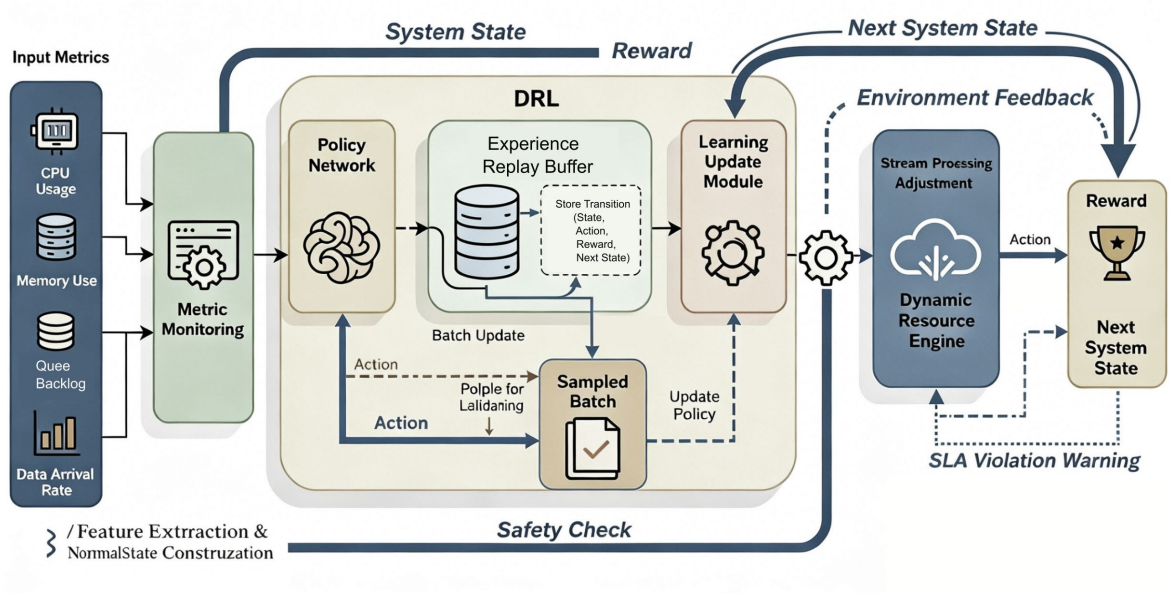

Figure 1. DRL-based Optimization Workflow.

https://doi.org/10.64972/dea.2025.v4i1.1939d:113-126

System Integration and Engineering Realization

Through strict modularization and scalable system architecture, DRL-based optimization can be effectively deployed in distributed data stream engines. The stream processing core, system interaction layer, and network layer are the three core components of the DRL optimization module. In the case of increased load, each layer ensures stability and scalability.

The system state is displayed as a multidimensional vector in each control cycle, containing the current metrics of CPU utilization, memory usage, queue depth, and parallel settings:

$$\boldsymbol{s}_t = [CPU_t, Mem_t,\ Queue_t, \dots] \quad \text{Eq.(4)}$$

Therefore, the latest and complete picture of the system state will always be available for decision-making.

According to its learned policy, the DRL module will use the aforementioned state vector to determine the optimal action. At time step t, the action chosen by the agent is:

$$a_t = \pi_\theta(\boldsymbol{s}_t) \quad \text{Eq.(5)}$$

This small map has many excellent feature mappings that can enhance the system's responsiveness and efficiency. These feature mappings include flexible control over operator scaling, resource allocation, and task reassignment.

After the selection is completed, it will be transmitted to the stream processing core through the system interaction layer. This is done to ensure the safe and stable operation of the actuator. After performing this operation, the processing core will update its operating status based on the direct impact of the operation and changes in external load.

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, a_t, \lambda_t) \quad \text{Eq.(6)}$$

Here, $\mathbf{x}_t$ denotes the micro-state of the processing elements, and λ_t represents the instantaneous input rate.

We will provide high-quality service. Service level constraints will validate the actions proposed by all DRL agents. For example, sensitive applications require decision results to meet:

$$P(\ Latency_t \leq L_{SLA}) \geq \eta \quad \text{Eq.(7)}$$

where L_{SLA} is the latency bound and η is the required confidence level.

The overall effectiveness of the quantitative system is to provide feedback on long-term strategic improvements and reward distribution for agents. The total utility at a specific stage can be expressed as follows:

$$U_t = \sum_{i=1}^{N} w_i R_{i,t} - \sum_{j=1}^{M} \beta_j C_{j,t} \quad \text{Eq.(8)}$$

where $R_{i,t}$ denotes the reward for component i (e.g., throughput achieved), and $C_{j,t}$ the cost associated with resource j.

In order to prevent conflicts during the operation of the distributed stream processing cluster, a consensus mechanism has been added to the system to achieve collaboration. As shown below:

$$A_t^{cons} = Consensus(\{a_t^1, a_t^2, \dots, a_t^K\}) \quad \text{Eq.(9)}$$

where $\{a_t^k\}$ are actions proposed by K distributed DRL instances. The system thus achieves unified control with minimal conflict and high reliability.

Figure 2 shows the three parts of the overall structure: learning, control, execution, and operation. In order to meet future engineering needs, this design has high scalability, easily supports algorithm upgrades, and strongly supports multiple streaming platforms.

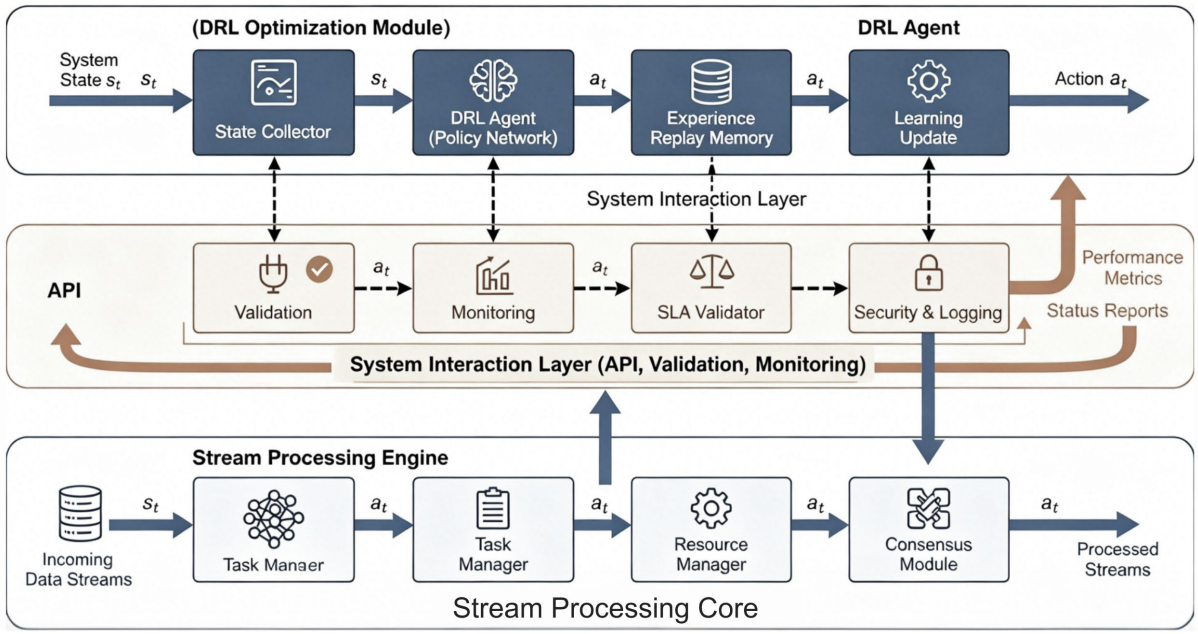

Figure 2. Overall System Structure for Real-time Stream Processing.

Extensibility and Reliability Considerations

Currently, the application of deep reinforcement learning-driven stream processing systems in production requires strong reliability, scalability, and high-performance optimization. The aforementioned features aim to ensure that the system maintains high availability and flexibility when responding to changing load demands and other factors.

By modularizing algorithms and software, scalability is achieved. The DRL optimization module is decoupled from the stream processing core, communicating only through clearly defined APIs and standardized data formats. When new resources are added or system requirements are modified, the system's state vector can be extended, as shown below:

$$\boldsymbol{s}_t = [CPU_t, Mem_t,\ Queue_t, GPU_t, \dots] \quad \text{Eq.(10)}$$

The aforementioned design can add new state functions or reward functions, and it can be relatively easily extended and adapted to various environments.

By accumulating experience, the adaptability of the learning agent can be enhanced, as shown below:

$$D_{t+1} = D_t \cup \{(\boldsymbol{s}_t, a_t, r_t, \boldsymbol{s}_{t+1})\} \quad \text{Eq.(11)}$$

where D_t represents the replay buffer at time t. Regularly update policies and retrain using new data to address concept drift and promptly adapt to environmental changes; the workload and platform will remain relatively stable.

Limit the startup and security settings of the device. Only when the CPU and memory reach high threshold conditions will the DRL agent make a decision.

$$C(\boldsymbol{s}_t, a_t) = \mathbb{I}(\ CPU_t < B_{CPU} \wedge\ Latency_t \le L_{SLA}) \quad \text{Eq.(12)}$$

where B_{CPU} and L_{SLA} denote operational bounds, and $\mathbb{I}$ is the indicator function. If the proposal fails verification, the safety mechanism will immediately revert the policy to the safest state to prevent harm.

Finally, comprehensive operation logs and encrypted communication ensure security and transparency. Record all control operations and learning updates, then conduct a quantitative impact analysis to show how policies change over time:

$$I_t = \int_{u=0}^{w} |r_{t+u} - r^*_{t+u}| du \quad \text{Eq.(13)}$$

where r^* denotes baseline reward and w is the evaluation window.

https://doi.org/10.64972/dea.2025.v4i1.1939d:113-126

Experiments and Results

Experimental Setup and Baselines

In order to test the actual performance of the DRL-based distributed real-time stream processing optimization framework, an experimental test platform was built that closely resembles a real production environment. The cluster consists of 12 homogeneous x86_64 servers, each equipped with two 16-core Intel Xeon CPUs, 128 GB of memory, and a 10 Gb/s Ethernet network. All these experiments were conducted on these servers. The cluster is managed by Kubernetes 1.25 and uses Ubuntu 20.04 LTS for containerized deployment. For the underlying data processing engine, I chose Apache Flink 1.18 because it has recently become very popular, with a scalable architecture, low-latency stream processing model, and rich API support.

First, the experimental workload is derived from the Yahoo! Streaming Benchmark Suite, which allows for the configuration of various input arrival intensity levels and stream statistical properties to observe the system's response under these different data rate conditions. The cluster is extended to the New York taxi dataset to enhance the realism of the evaluation and address the issue of non-stationarity. The production dataset contains approximately 1.1 billion events, which vary throughout the day and across seasonal changes, presenting a challenging and typical workload for steady-state and adaptive system behavior.

The following metrics are used for performance evaluation. Throughput is the number of events that can be stably processed per second; therefore, it demonstrates the system's scalability and raw processing capability. Under normal and high load conditions, the end-to-end latency is the 99th percentile latency, from the start of data arrival to the end of result generation. At the same time, the utilization of CPU and memory resources was recorded to assess overall efficiency. The proportion of time intervals where processing delays fall within the upper and lower limits set by the SLA is considered the actual service level of the system. Fault injection is used to evaluate the system's resilience and record the time required to restore throughput to pre-failure levels.

Many well-known fundamental algorithms were also used for comparison. A fixed-size allocation pattern was used, and operator parallelism and resource sharing were set based on the analysis of previous experiments. In addition to the above, a rule-based extension scheme was introduced, which uses the thresholds provided in Table 2-1 as a reference to dynamically adjust based on the current CPU load or latency [28]. We also added a model-based reinforcement learning baseline, specifically through learning-based control using a Deep Q-Network (DQN) with a discrete state-action space [29]. In addition, an adaptive heuristic method was studied, which considers windowed performance feedback and PID-inspired proportional adjustments to evaluate its adaptability in non-stationary environments [30]. After extensive iterations and optimization by experienced systems engineers, a human expert tuning baseline was established as the upper limit of manual performance.

In order to achieve optimal performance under the same conditions, all baseline models were optimized through manual selection or grid search. The proposed DRL agent's two-layer neural policy network was adopted, along with a replay buffer capable of holding 50,000 sets. The validation results of the retained portion of the workload determined the learning rate and exploration parameters. Therefore, this study will be conducted in a fair and accurate manner to provide a foundation for subsequent quantitative analysis [31].

Core Performance and Scalability Analysis

The comprehensive tests mentioned above, conducted on various applications and different scale deployments, validated the key performance of our new DRL optimization framework. Figure 3 shows the comprehensive visual comparison results of the DRL agent and the main baseline methods in terms of throughput, latency, CPU usage, and memory usage.

As shown in Figure 3(a), the DRL-based method has a maximum sustained throughput of 1.082 million events per second. This is far higher than the rule-based baseline (91.1k events/second), as well as the model-based (DQN), heuristic, and static strategies. Under peak load, compared to the industry-standard rule-based scaling, the average throughput increased by 18.7%. DRL maintains stable throughput with increased input rates and reasonably scales operators during peak system demand periods. Moreover, the throughput slightly above the average level supports the results of large-scale reinforcement learning resource management research [32].

Figure 3(b) shows a detailed analysis of system responsiveness, where the box plot displays the distribution of end-to-end event delays for various methods under different workload levels. For the DRL driver, the median latency is approximately 42 milliseconds, which is significantly lower than the 52 milliseconds of the rule-based method, while the latency for DQN, heuristic, and static baselines has increased. Under the DRL strategy, the 99th percentile latency still meets SLA requirements in 97.1% of the measurement periods and is very stable in emergencies, which means the worst tail latency is also within range. This reliability is particularly suitable for real-time requirements and is consistent with the advancements in reinforcement learning adaptability research [33].

In addition, an investigation into resource economy was conducted. As shown in Figure 3(c), the DRL structure reduced CPU utilization compared to the competing scheme. Under the same workload and dynamic load conditions, the DRL agent achieved smoother CPU scaling, peaking at 54%, while the rule-based policy reached 61%. Therefore, it improves energy usage. It seems that there is no over-allocation of resources, so it can be inferred that the DRL model dynamically adjusts resource allocation. As shown in Figure 3(d), the DRL framework also uses less memory. At its peak, it also uses less memory than the rule-based and DQN baselines. The main reason for this high efficiency is the load prediction and proactive queue management strategies learned in the DRL policy.

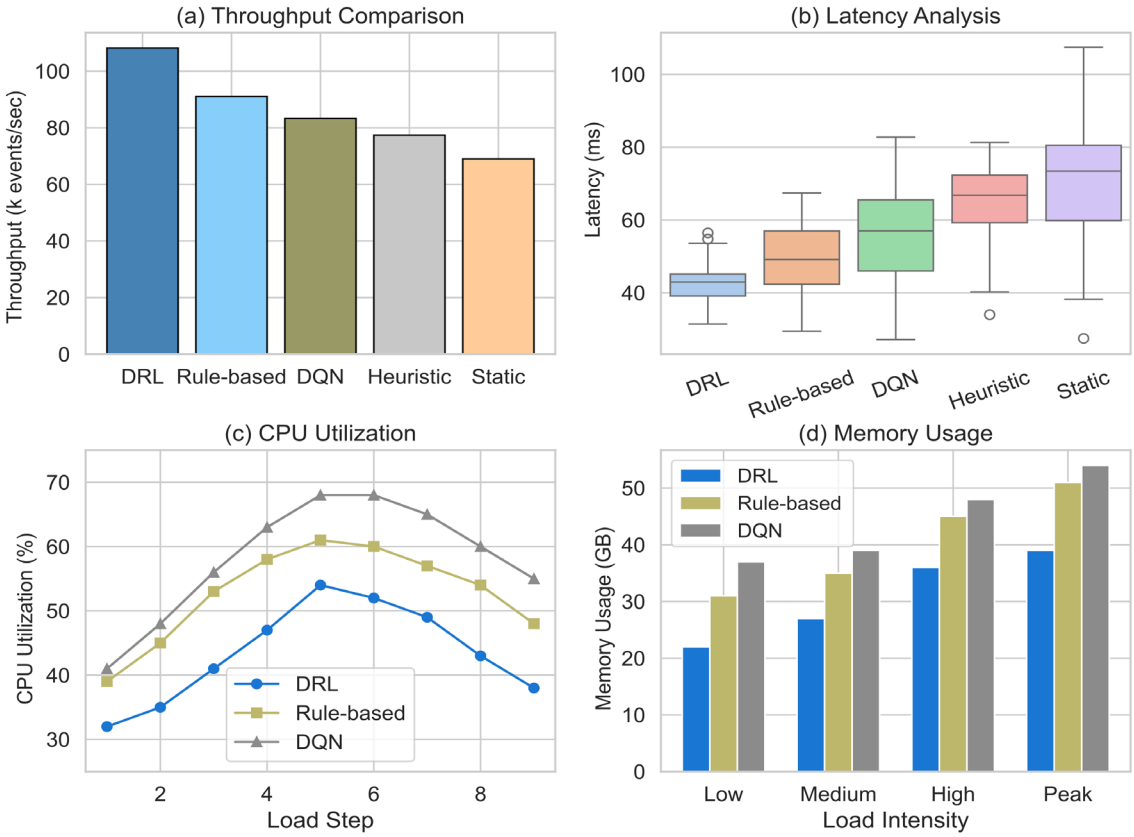

Figure 3. Multi-metric Performance Comparison (a) Throughput Comparison. (b) Latency Analysis. (c) CPU Utilization. (d) Memory Usage.

Organizations evaluate its scalability in two dimensions. Horizontal scaling exists, as shown in Figure 4(a). In other words, when a cluster node is added, the system throughput increases almost linearly and has better scalability efficiency compared to traditional methods. As the number of nodes increases from 2 to 12, the throughput under the DRL strategy correspondingly increases from 22k to 134k events per second. On the other hand, after 10 nodes, the rule-based baseline sharply tends to stabilize. As shown in Figure 4(b), increasing the computational resources of each node is used to calculate the performance after vertical scaling. When scaled to 32-core nodes, the throughput of the DRL solution increases from 14,000 events/second to 104,000 events/second, and it significantly outperforms the rule-based and DQN methods under all conditions.

Finally, Figure 4(c) shows the impact of increased data input speed. Box plots can be used to show how stable the latency curve of the DRL agent is across 10,000 to 120,000 events. Despite the increase in workload, the latency remains relatively low, far from exceeding the SLA or going out of range, indicating that the framework can quickly respond to input changes at different times. This issue is usually related to rule-based or static methods. The stream processing resource management project based on deep reinforcement learning has recently expanded upon the aforementioned results [34].

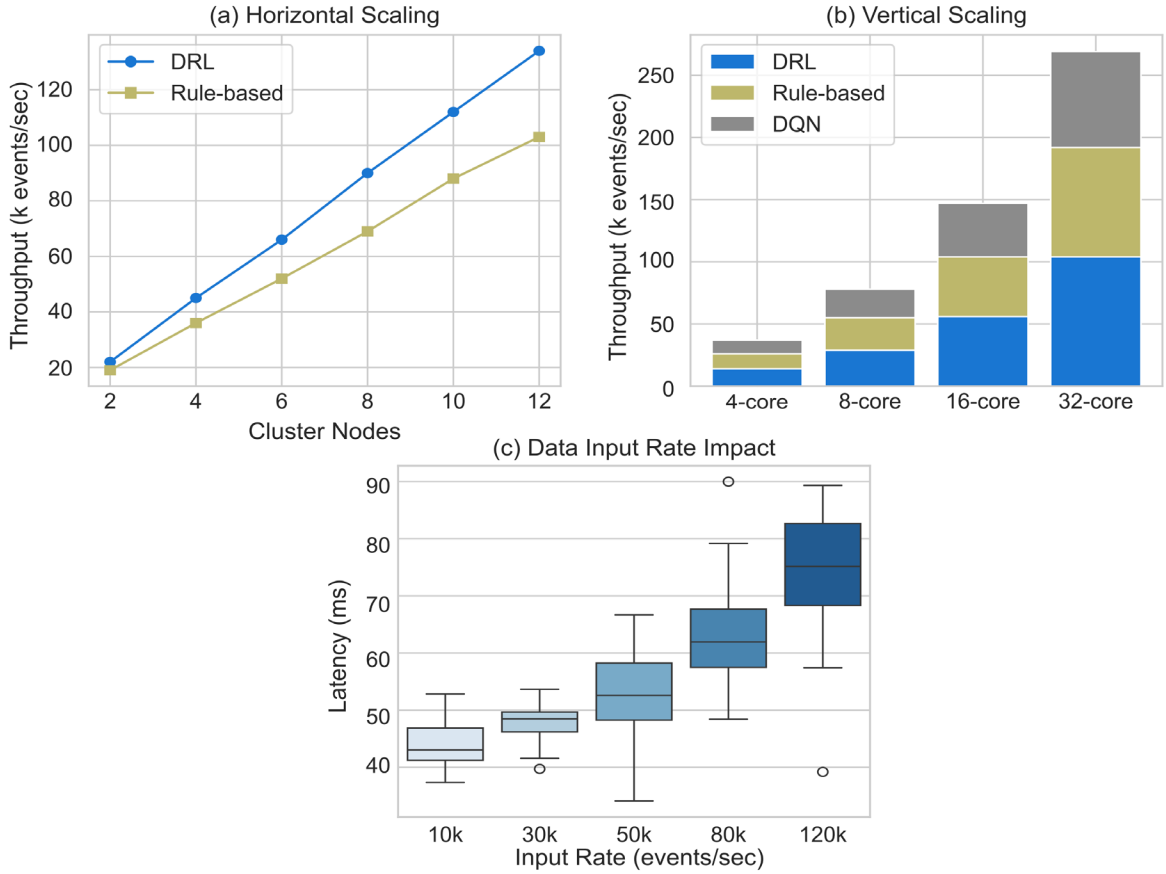

Figure 4. Scalability Trends (a) Horizontal Scaling. (b) Vertical Scaling. (c) Data Input Rate Impact.

Deep reinforcement learning can improve the performance of dynamic resource scheduling in distributed stream processing. Compared to heuristic and model-based methods, the DRL framework can utilize resources more efficiently, reduce tail latency, and achieve robust and high-throughput system scaling. These frameworks must be deployed in high-throughput data analysis tasks [35].

Robustness, Ablation, and Sensitivity Analysis

Through targeted fault injection, ablation studies, and controlled sensitivity analysis, the robustness and reliability of the DRL-based optimization framework have been comprehensively tested. Figure 5 shows the basic concept of fault tolerance and how the system operates during a fault. As shown in Figure 5(a), the enhanced DRL system typically recovers to its pre-failure throughput within 28.5 seconds after various node and network failures. The average recovery speeds for the rule-based baseline and DQN are 45.2 seconds and 58.7 seconds, respectively. Figure 5(b) shows another method of presenting the success rates of different induced fault scenarios. In terms of single-point and cascading multi-point failures, the DRL agent achieved a stable recovery success rate of 94.8%, surpassing all other benchmarks. The above experiments indicate that the framework is stable under production fault conditions and can automatically repair these faults. This is consistent with the latest findings on RL-based adaptive systems [36].

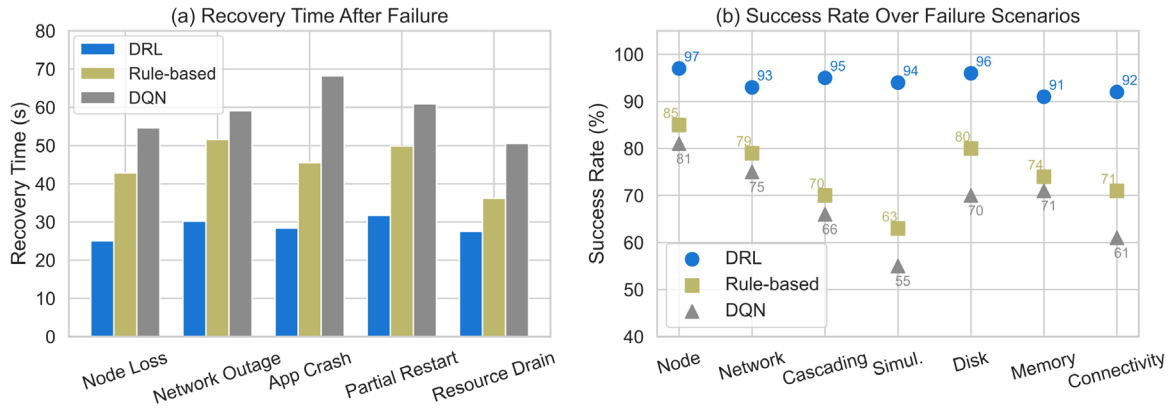

Figure 5. Fault Tolerance and Recovery Performance (a) Recovery Time After Failure. (b) Success Rate Over Multiple Failure Scenarios.

To further investigate the reasons for the performance improvement, a series of ablation experiments were conducted, and the results are shown in Figure 6. Figure 6(a) shows the methods tested for various reward function designs; the combined penalty scheme that integrates resource inefficiency and delay violations achieved excellent results. Compared to the single reward scheme, throughput and SLA satisfaction improved by 11.3% and 9.6%, respectively. Figure 6(b) shows the impact of state space selection. Ignoring queue lengths and other resource information will reduce policy quality, indicating that a complete system state model is crucial for continuous adaptive control. In addition, Figure 6(c) shows the test results of various DRL architecture options: deeper network designs lead to slower convergence rates and higher computational costs. For practical deployment, a reasonable balance must be found to balance expressive power and inference speed [37].

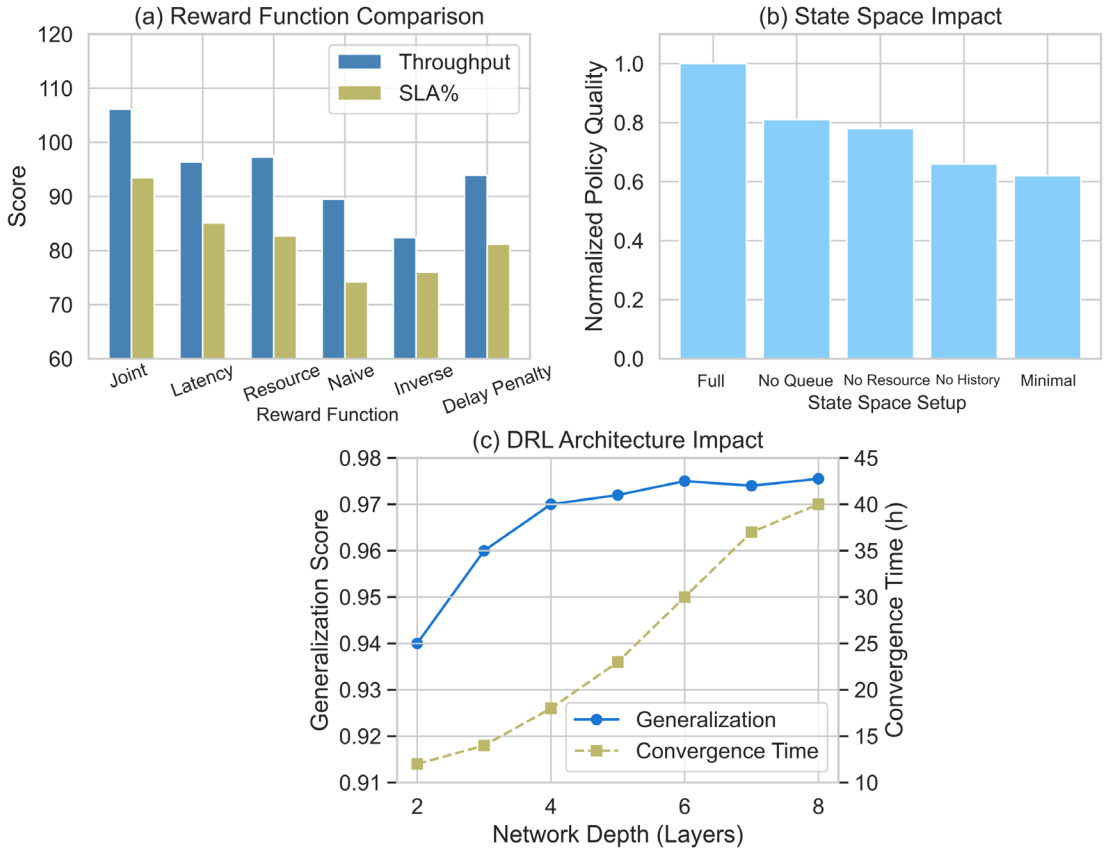

Figure 6. Ablation Studies (a) Different Reward Functions. (b) State Space Settings. (c) DRL Architectures.

As shown in Figure 7, sensitivity analysis is used to examine the adaptability of the DRL framework to hyperparameter and environmental changes. Figure 7(a) depicts the impact of changes in the policy discount factor on the average throughput. Policies with discount factors (γ) close to 0.98 maximized both throughput and stability, whereas both more myopic and overly long-horizon values resulted in suboptimal, oscillatory adjustments. In Figure 7(b), the influence of different exploration strategies was assessed: entropy regularization yielded slightly better long-term performance than ε-greedy alternatives, particularly under dynamic workload distributions. Figure 7(c) shows the impact of workload variation. It can be seen that even when the standard deviation of the workload increases by more than two times, the DRL agent still maintains stable application-level latency within the strict SLA range. When the workload suddenly increases, the DRL agent also remains stable.

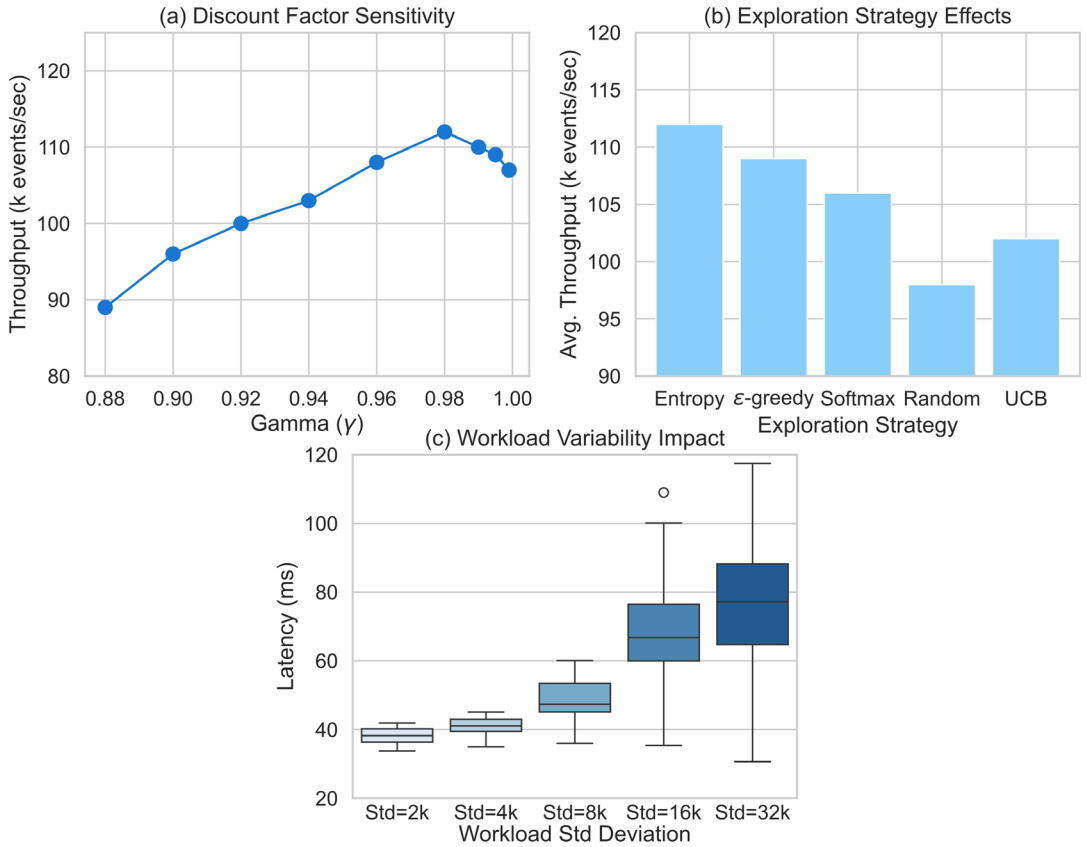

Figure 7. Hyperparameter and Environment Sensitivity (a) Discount Factor Sensitivity. (b) Exploration Strategy Effects. (c) Workload Variability Impact.

The multidimensional tests indicate that the new system based on DRL is, to some extent, reliable, interpretable, and fault-tolerant in terms of design and environment, and can effectively control resources under normal conditions. As shown in the aforementioned experimental stress tests, this robustness to real-world changes far exceeds the stability guaranties provided by learning-based resource managers [38]. It must be deployed in a distributed stream computing production environment.

Conclusion

This article provides a detailed introduction on how Deep Reinforcement Learning (DRL) can achieve intelligent resource optimization in distributed real-time stream processing systems. To significantly improve the performance of existing top-tier baseline policies in all of the following aspects, we developed and integrated a domain-adaptive deep reinforcement learning (DRL) agent. A large number of experiments have shown that DRL-based methods outperform static heuristic and model-driven control methods in terms of maximum sustainable throughput, median and tail event latency, and resource consumption. When encountering failures, the system demonstrates good stability and rapid recovery capabilities. Through ablation and sensitivity analysis,

the reasons for these improvements were identified in the model design. This paper provides new directions for the automatic control of complex stream processing workloads and offers an operational solution that has improved the reliability, performance, and adaptability of high-throughput data analysis in recent years.

This paper provides theoretical support for deep reinforcement learning (DRL) and large-scale streaming data systems. To provide a foundation for the continuous integration of control algorithms into distributed computing infrastructure, a new adaptive optimization architecture has been developed, and its learning dynamics under real-world workloads have been thoroughly studied. Due to its demonstrated improvements in throughput, reduction in latency, and fault tolerance in practice, it has been recommended for use in IoT networks, financial transaction analysis, and critical real-time monitoring systems. Show how data-driven intelligence can be applied in modern streaming architectures and deploy best practices for self-optimizing resource management applications.

The aforementioned achievements still have some issues. The computational load of the current framework may be too high, and its generalization ability on highly heterogeneous clusters is limited. Model interpretability and explainability: Although ablation experiments have been conducted, more research is still needed to ensure the safety of high-risk environments. Improve the scalability and interpretability of the DRL module, create a hybrid model of AI-based resource management systems, and validate this approach through extended real-world data sources and distributed environments. Due to the rapid development of adaptive and autonomous computing models, establishing intelligent stream processing systems remains a daunting task. This paper aims to lay the foundation for future research in this field.

Author Contributions

Mariusz Ostrowski contributes to conceptualization, methodology, software, validation, analysis, investigation, data collection, draft preparation, manuscript editing, visualization. Jarosław Bąk and Seweryn Sokołowski contributes to draft preparation, conceptualization, methodology, software and supervision. All authors have read and agreed with the manuscript before its submission and publication.

Funding

This research received no specific financial support from any funding agency.

Institutional Review Board Statement

Not applicable.

References

[1] Rahman, G. S., Dang, T., & Ahmed, M. (2020). Deep reinforcement learning based computation offloading and resource allocation for low-latency fog radio access networks. Intelligent and Converged Networks, 1(3), 243-257. https://doi.org/10.23919/ICN.2020.0020

[2] Pal, S., Jhanjhi, N. Z., Abdulbaqi, A. S., Akila, D., Alsubaei, F. S., & Almazroi, A. A. (2023). An intelligent task scheduling model for hybrid internet of things and cloud environment for big data applications. Sustainability, 15(6), 5104. https://doi.org/10.3390/su15065104

[3] Chen, Z., Hu, J., Min, G., Luo, C., & El-Ghazawi, T. (2021). Adaptive and efficient resource allocation in cloud datacenters using actor-critic deep reinforcement learning. IEEE Transactions on Parallel and Distributed Systems, 33(8), 1911-1923. https://doi.org/10.1109/TPDS.2021.3132422

[4] Wang, X., Zhang, C., Fang, J., Zhang, R., Qian, W., & Zhou, A. (2022). A comprehensive study on fault tolerance in stream processing systems. Frontiers of Computer Science, 16(2), 162603. https://doi.org/10.1007/s11704-020-0248-x

[5] Li, B., Zhang, R., Tian, X., & Zhu, Z. (2021). Multi-agent and cooperative deep reinforcement learning for scalable network automation in multi-domain SD-EONs. IEEE Transactions on Network and Service Management, 18(4), 4801-4813. https://doi.org/10.1109/TNSM.2021.3102621

[6] Fawaz, H., Zeghlache, D., Pham, T. A., Leguay, J., & Medagliani, P. (2021). Deep reinforcement learning for smart queue management. Electronic Communications of the EASST, 80. https://doi.org/10.14279/tuj.eceasst.80.1139

[7] Xu, J., & Palanisamy, B. (2021, December). Model-based reinforcement learning for elastic stream processing in edge computing. In 2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC) (pp. 292-301). IEEE. https://doi.org/10.1109/HiPC53243.2021.00043

[8] Ed-daoudy, A., & Maalmi, K. (2019). A new Internet of Things architecture for real-time prediction of various diseases using machine learning on big data environment. Journal of Big Data, 6(1), 104. https://doi.org/10.1186/s40537-019-0271-7

[9] Liang, W., Huang, W., Long, J., Zhang, K., Li, K. C., & Zhang, D. (2020). Deep reinforcement learning for resource protection and real-time detection in IoT environment. IEEE Internet of Things Journal, 7(7), 6392-6401. https://doi.org/10.1109/JIOT.2020.2974281

[10] Zaman, Q., Alraho, S., & König, A. (2021). Efficient transient testing procedure using a novel experience replay particle swarm optimizer for THD-based robust design and optimization of self-X sensory electronics in industry 4.0. Journal of Sensors and Sensor Systems, 10(2), 193-206. https://doi.org/10.5194/jsss-10-193-2021, 2021.

[11] Li, W., Zhang, J., Xia, H., Liu, Q., Wang, Y., Jia, Y., & Chen, Y. (2024). Cross-scene building identification based on dual-stream neural network and efficient channel attention mechanism. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 17, 6920-6932. https://doi.org/10.1109/JSTARS.2024.3375321

[12] Nguyen, T. T., Nguyen, N. D., Vamplew, P., Nahavandi, S., Dazeley, R., & Lim, C. P. (2020). A multi-objective deep reinforcement learning framework. Engineering Applications of Artificial Intelligence, 96, 103915. https://doi.org/10.1016/j.engappai.2020.103915

[13] Devi, N., Dalal, S., Solanki, K., Dalal, S., Lilhore, U. K., Simaiya, S., & Nuristani, N. (2024). A systematic literature review for load balancing and task scheduling techniques in cloud computing. Artificial Intelligence Review, 57(10), 276. https://doi.org/10.1007/s10462-024-10925-w

[14] Sun, G., Ayepah-Mensah, D., Chen, H., Boateng, G. O., & Liu, G. (2024). FeDistSlice: Federated Policy Distillation for Collaborative Intelligence in Multi-Tenant RAN Slicing. IEEE Transactions on Services Computing, 18(1), 184-197. https://doi.org/10.1109/TSC.2024.3517334

[15] Dong, T., Xue, F., Tang, H., & Xiao, C. (2023). Deep reinforcement learning for fault-tolerant workflow scheduling in cloud environment. Applied Intelligence, 53(9), 9916-9932. https://doi.org/10.1007/s10489-022-03963-w

[16] Kim, M., & Chung, K. (2023). HTTP adaptive streaming scheme based on reinforcement learning with edge computing assistance. Journal of Network and Computer Applications, 213, 103604. https://doi.org/10.1016/j.jnca.2023.103604

[17] Shekhawat, J. S., Agrawal, R., Shenoy, K. G., & Shashidhara, R. (2020, December). A reinforcement learning framework for QoS-driven radio resource scheduler. In GLOBECOM 2020-2020 IEEE Global Communications Conference (pp. 1-7). IEEE. https://doi.org/10.1109/GLOBECOM42002.2020.9322182

[18] Dehury, C. K., & Srirama, S. N. (2024, July). Integrating Serverless and DRL for Infrastructure Management in Streaming Data Processing across Edge-Cloud Continuum. In 2024 IEEE 44th International Conference on Distributed Computing Systems Workshops (ICDCSW) (pp. 93-101). IEEE. https://doi.org/10.1109/ICDCSW63686.2024.00020

[19] Bhatt, N., & Thakkar, A. (2021). An efficient approach for low latency processing in stream data. PeerJ Computer Science, 7, e426. https://doi.org/10.7717/peerj-cs.426

[20] Yang, Y., & Shen, H. (2021). Deep reinforcement learning enhanced greedy optimization for online scheduling of batched tasks in cloud HPC systems. IEEE Transactions on Parallel and Distributed Systems, 33(11), 3003-3014. https://doi.org/10.1109/TPDS.2021.3138459

[21] Rupani, K., Punjabi, N., Shamdasani, M., & Chaudhari, S. (2020, January). Dynamic load balancing in software-defined networks using machine learning. In Proceeding of International Conference on Computational Science and Applications: ICCSA 2019 (pp. 283-292). Singapore: Springer Singapore. https://doi.org/10.1007/978-981-15-0790-8_28

[22] Lu, J. B., Yu, Y., & Pan, M. L. (2021, December). Reinforcement learning-based auto-scaling algorithm for elastic cloud workflow service. In International Conference on Parallel and Distributed Computing: Applications and Technologies (pp. 303-310). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-96772-7_28

[23] Pan, Z., Wang, L., Dong, C., & Chen, J. F. (2023). A knowledge-guided end-to-end optimization framework based on reinforcement learning for flow shop scheduling. IEEE Transactions on Industrial Informatics, 20(2), 1853-1861. https://doi.org/10.1109/TII.2023.3282313

[24] Jiang, F., Dong, L., Wang, K., Yang, K., & Pan, C. (2021). Distributed resource scheduling for large-scale MEC systems: A multiagent ensemble deep reinforcement learning with imitation acceleration. IEEE Internet of Things Journal, 9(9), 6597-6610. https://doi.org/10.1109/JIOT.2021.3113872
[25] Zhang, C., & Zheng, Z. (2019). Task migration for mobile edge computing using deep reinforcement learning. Future Generation Computer Systems, 96, 111-118. https://doi.org/10.1016/j.future.2019.01.059
[26] Tian, W., Fu, G., Xin, K., Zhang, Z., & Liao, Z. (2024). Improving the interpretability of deep reinforcement learning in urban drainage system operation. Water Research, 249, 120912. https://doi.org/10.1016/j.watres.2023.120912
[27] Read, M. R., Dehury, C., Srirama, S. N., & Buyya, R. (2024). Deep reinforcement learning (DRL)-based methods for serverless stream processing engines: A vision, architectural elements, and future directions. In Resource Management in Distributed Systems (pp. 285-314). Singapore: Springer Nature Singapore. https://doi.org/10.1007/978-981-97-2644-8_14
[28] Mangalampalli, S., Karri, G. R., Kumar, M., Khalaf, O. I., Romero, C. A. T., & Sahib, G. A. (2024). DRLBTSA: Deep reinforcement learning based task-scheduling algorithm in cloud computing. Multimedia tools and applications, 83(3), 8359-8387. https://doi.org/10.1007/s11042-023-16008-2
[29] Liu, C., Xu, M., Yang, Y., & Geng, N. (2021, May). DRL-OR: Deep reinforcement learning-based online routing for multi-type service requirements. In IEEE INFOCOM 2021-IEEE Conference on Computer Communications (pp. 1-10). IEEE. https://doi.org/10.1109/INFOCOM42981.2021.9488736
[30] Shuprajhaa, T., Sujit, S. K., & Srinivasan, K. (2022). Reinforcement learning based adaptive PID controller design for control of linear/nonlinear unstable processes. Applied Soft Computing, 128, 109450. https://doi.org/10.1016/j.asoc.2022.109450
[31] Yang, Q., Cao, W., Meng, W., & Si, J. (2021). Reinforcement-learning-based tracking control of waste water treatment process under realistic system conditions and control performance requirements. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 52(8), 5284-5294. https://doi.org/10.1109/TSMC.2021.3122802
[32] Gures, E., Shayea, I., Ergen, M., Azmi, M. H., & El-Saleh, A. A. (2022). Machine learning-based load balancing algorithms in future heterogeneous networks: A survey. IEEE Access, 10, 37689-37717. https://doi.org/10.1109/ACCESS.2022.3161511
[33] Cao, C., Dai, M., Shen, B., Zou, G., & Dong, W. (2023). Neural adaptive IoT streaming analytics with RL-Adapt. Computer Networks, 235, 109924. https://doi.org/10.1016/j.comnet.2023.109924
[34] Nokleby, M., Raja, H., & Bajwa, W. U. (2020). Scaling-up distributed processing of data streams for machine learning. Proceedings of the IEEE, 108(11), 1984-2012. https://doi.org/10.1109/JPROC.2020.3021381
[35] Wang, X., Guo, Y., & Gao, Y. (2024). Unmanned autonomous intelligent system in 6G non-terrestrial network. Information, 15(1), 38. https://doi.org/10.3390/info15010038
[36] Zhu, L., Zhuang, Q., Jiang, H., Liang, H., Gao, X., & Wang, W. (2023). Reliability-aware failure recovery for cloud computing based automatic train supervision systems in urban rail transit using deep reinforcement learning. Journal of Cloud Computing, 12(1), 147. https://doi.org/10.1186/s13677-023-00502-x
[37] Hu, X., Hu, H., Verma, S., & Zhang, Z. L. (2020). Physics-guided deep neural networks for power flow analysis. IEEE Transactions on Power Systems, 36(3), 2082-2092. https://doi.org/10.1109/TPWRS.2020.3029557
[38] Ramakrishna, S. (2023). Cloud-Native AI Platform for Real-Time Resource Optimization in Governance-Driven Project and Network Operations. International Journal of Engineering & Extended Technologies Research (IJEETR), 5(2), 6282-6291. https://doi.org/10.15662/IJEETR.2023.0502005