Transactions on Computational and Scientific Methods | Vol. 4, No. 4, 2024
ISSN: 2998-8780
https://pspress.org/index.php/tcsm
Pinnacle Science Press

Automatic Elastic Scaling in Distributed Microservice Environments via Deep Q-Learning

Lian Lian
University of Southern California, Los Angeles, USA
lianlb2025@gmail.com

Abstract: This study addresses the challenges of designing elastic scaling strategies, low resource utilization, and high response latency in microservice systems by proposing an automatic scaling and elasticity optimization algorithm based on Deep Q-Learning. The method models the microservice system as a Markov Decision Process and constructs a multi-dimensional state space. A deep neural network is used to approximate the Q-value function, and mechanisms such as experience replay and target network updates are introduced to enhance training stability and policy generalization. The model design incorporates current resource loads, service dependency structures, and runtime states to construct a refined action space that supports dynamic selection among scaling out, scaling in, and maintaining the current state. To evaluate performance in practical scheduling scenarios, several experiments are conducted, including comparisons with mainstream reinforcement learning methods, sensitivity analyses of key hyperparameters, and adaptability tests under different sampling frequencies and training data sizes. Results show that the proposed method outperforms existing approaches in terms of average response time, resource cost, and QoS violation rate, demonstrating good convergence speed, robustness, and adaptability. Based on a complete policy learning framework, this study systematically quantifies the modeling capability of reinforcement learning in microservice elasticity scheduling and provides empirical support for resource management in complex service environments.

Keywords: Microservice scheduling; elastic resource management; deep Q learning; automatic scaling

1. Introduction

With the rapid development of cloud computing, microservice architecture has become a mainstream approach for building modern application systems due to its high scalability and flexibility[1]. By decomposing monolithic systems into multiple autonomous services, microservices enable systems to maintain high availability and responsiveness under fluctuating user traffic, changing business demands, and increasing deployment complexity. However, the fine-grained deployment and heterogeneous management introduced by microservices also pose new challenges to resource allocation and scheduling strategies. In scenarios with volatile workloads, achieving efficient and timely scaling becomes a key factor affecting system performance and cost-effectiveness[2].

The goal of automatic scaling is to dynamically adjust resource configurations based on the actual load of services. This helps to balance resource utilization efficiency and service quality. Traditional strategies often rely on threshold-based rules. For example, scaling out is triggered when CPU or memory usage exceeds a fixed value, and scaling in occurs otherwise. However, such methods lack adaptability and respond poorly to

complex and dynamic environments. Frequent scaling actions may also cause resource jitter, which can affect system stability. Therefore, an intelligent strategy that can perceive environmental changes and learn autonomously is urgently needed to guide elastic scheduling decisions for microservices.

In recent years, reinforcement learning has shown strong performance in complex decision-making tasks. Deep Q-Learning, in particular, has demonstrated great potential in dynamic resource management. This method interacts with the environment and learns the optimal state-action policy through experience. It enables adaptive decision-making without relying on manually defined rules. Applying Deep Q-Learning to microservice elasticity management allows systems to autonomously evaluate system status using multi-dimensional monitoring indicators. It can also balance strategies between resource allocation, response latency, and operational costs, eventually forming globally optimized scaling policies[3].

In real-world microservice systems, services are often interdependent and associated with multiple performance indicators, such as service call chains, average response time, error rate, and network latency. Traditional scaling methods driven by a single metric cannot fully capture the complexity of system states. Deep Q-Learning can build multi-dimensional state spaces and high-dimensional policy representations. It leverages historical traces and real-time load data to optimize elasticity strategies from a global perspective. This approach improves system awareness and provides a foundation for maximizing long-term benefits under uncertain conditions[4].

Research on scaling and elasticity optimization algorithms based on Deep Q-Learning holds both theoretical and practical value. On one hand, it promotes the application of reinforcement learning in cloud-native environments and explores the deep integration of artificial intelligence with system resource management. On the other hand, intelligent elasticity mechanisms can significantly improve resource efficiency, performance stability, and cost control in microservice systems. This provides strong support for automated operations in large-scale distributed systems. As enterprises accelerate digital transformation and evolve their service architectures, building intelligent, efficient, and adaptive elasticity management has become a core requirement in the development of microservice technologies.

2. Relevant Literature

As the problem of dynamic resource scheduling in cloud computing continues to gain attention, automatic scaling has become a key research focus in microservice system management. Early scaling methods mostly relied on static thresholds or manual configurations. Common trigger conditions were based on fixed metrics such as CPU usage, memory consumption, or request counts. These methods are easy to implement and deploy. However, they show clear limitations when dealing with dynamic workloads and complex service dependencies. In high-concurrency and high-uncertainty environments, static thresholds often lead to delayed scaling responses. They can also cause resource redundancy or performance bottlenecks, making it difficult to meet the requirements of modern cloud-native systems for elasticity and efficiency[5].

To improve the adaptability of auto-scaling, researchers have introduced predictive models into elasticity management strategies. These approaches use historical data to build time series models or machine learning models. The models predict future service request volumes or resource needs and trigger scaling actions in advance. Common models include ARIMA, support vector machines, and neural networks. These methods enhance the foresight of scaling strategies to some extent. However, they often depend on accurate predictions. Their generalization ability is limited, and they struggle to handle sudden traffic spikes or structural workload changes. Moreover, training and deploying prediction models increase system complexity and maintenance costs[6].

In recent years, reinforcement learning has received increasing attention in the field of elastic computing. It has shown strong autonomous decision-making capabilities in tasks such as resource scheduling, load balancing, and container management. Unlike traditional methods, reinforcement learning does not rely on explicit rules or prediction models. It learns optimal policies through continuous interaction with the environment. Deep Q-Learning, in particular, combines deep neural networks with Q-value updates. It can handle continuous state spaces and complex decision paths. It is capable of optimizing long-term returns in uncertain environments. In microservice elasticity management, reinforcement learning can select optimal scaling actions based on the current system state. It also takes service dependencies and performance metrics into account to gradually approach optimal resource scheduling solutions[7,8].

To improve the effectiveness of reinforcement learning in real-world systems, some studies have introduced mechanisms such as multi-agent systems, hierarchical policy modeling, and joint optimization. These methods enhance scalability and adaptability. Other research has combined reinforcement learning with additional techniques. For example, prediction mechanisms can guide policy training, and knowledge transfer can accelerate policy convergence. These developments have expanded the application scope of reinforcement learning in microservice environments. They support the evolution from static thresholds and single-point prediction to multi-dimensional sensing and dynamic decision-making. Although recent work has made progress, challenges remain. These include multi-objective constraints, resource conflicts, and system stability. Current methods may still suffer from weak robustness, slow convergence, or deployment difficulties. Therefore, developing efficient, stable, and deployable reinforcement learning algorithms for elasticity management remains a valuable direction for ongoing research.

3. Method Overview

The microservice elasticity strategy optimization method proposed in this paper is built on a deep Q-learning framework, aiming to achieve intelligent scheduling and on-demand scaling of resources under dynamic load conditions. Its model architecture is shown in Figure 1.

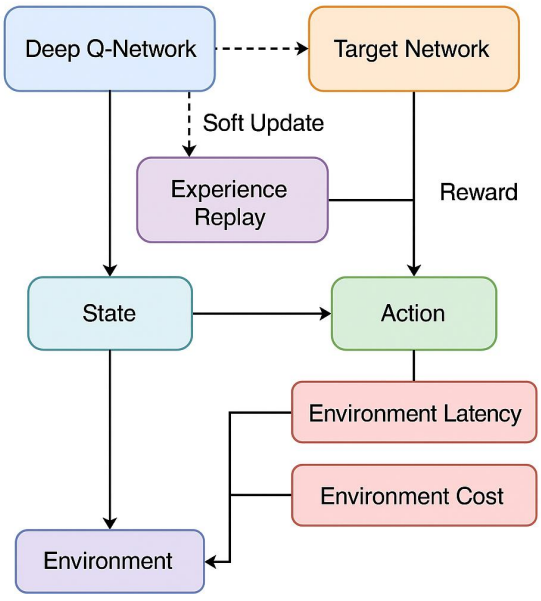


Figure 1. Architecture of the Deep Q-Learning-Based Elasticity Policy Optimization Framework

First, the microservice system is modeled as a Markov decision process (MDP), where the state space S represents the current operating state of the system, including a combination of multiple indicators such as CPU usage, memory occupancy, request queue length, service call chain depth, etc.; the action space A is defined as possible operations such as expansion, reduction, or unchanged; the reward function R is used to feedback the positive and negative impact of the action on the system performance, and the goal is to maximize the long-term cumulative return. The overall goal can be formalized as learning a strategy $\pi : S \to A$ that maximizes the expected return function:

$$\max_{\pi} E[\sum_{t=0}^{\infty} \gamma_t R(s_t, a_t)]$$

Where $\gamma \in [0,1]$ is the discount factor, which controls the importance of future returns.

To achieve policy learning, the Q-value function $Q(s,a)$ is used to estimate the cumulative reward that can be obtained by taking action a in state s. The Q-value is approximated by a deep neural network, and a function approximation model $Q(s,a;\theta)$ is constructed, where θ the network parameters are. The Q-value update adopts the iterative form of the Bellman equation, as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) +$$
$$\alpha[r_t + \gamma \max_{a} Q(s_{t+1}, a') - Q(s_t, a_t)]$$

α is the learning rate, r_t is the immediate reward for the current action, and s_{t+1} is the next state after executing the action.

In order to improve the model stability and sample utilization efficiency, the experience replay mechanism is introduced. All sample triples (s_t, a_t, r_t, s_{t+1}) generated by the interaction with the environment are stored in the replay buffer, and small batches of samples are randomly extracted from it for gradient update during training. At the same time, the target network B is introduced to alleviate the non-stationary problem during training, and its parameters are synchronized through soft update:

$$\theta_{\text{target}} \leftarrow \tau\theta + (1-\tau)\theta_{\text{target}}$$

Where $\tau << 1$ is the step size parameter of the soft update, which ensures that the target network changes slowly with the main network, thus providing a stable learning objective.

In terms of reward design, we comprehensively consider system performance and resource cost and construct the following reward function:

$$R(s_t, a_t) = -\lambda_1 \cdot Latency(s_t) - \lambda_2 \cdot Cost(a_t)$$

$Latency(s_t)$ represents the average response delay in the state, $Cost(a_t)$ represents the resource cost overhead caused by the execution of scaling actions, and λ_1, λ_2 is the balance coefficient between control performance and cost weight. By continuously iterating and optimizing the above strategy function, the model can achieve efficient scaling decisions in dynamic scenarios and improve system elasticity and resource utilization efficiency.

4. Experimental Dataset

This study uses the Google Cluster Data, also known as the Google Borg Trace, as the primary dataset. It is employed to simulate and evaluate resource scheduling and scaling strategies in microservice environments. The dataset consists of real large-scale cluster operation logs. It includes detailed records of scheduling events, resource allocation, task lifecycles, and performance metrics across thousands of machines over several days or weeks. It has strong representativeness and rich data dimensions. It is widely used in research on resource management and scheduling strategies.

The core fields in the dataset include task submission time, start time, completion time, requested and actual usage of CPU and memory, task priority, and scheduling class. This information provides comprehensive support for state construction, action selection, and reward design in reinforcement learning. It is especially suitable for scenarios in microservice architectures that require fine-grained resource awareness and elastic control. By properly extracting and constructing the state space, the dataset can simulate dynamic changes in task load and resource pressure in real systems.

The dataset also contains node-level information, such as total machine resources, state changes, and failure logs. This helps in building dynamic environment feedback mechanisms. Due to its large scale, high dimensionality, and real structure, the Google Cluster Data is widely regarded as one of the most authoritative data sources for research in cloud computing resource management. In this study, the dataset is used to construct a simulation environment and to train and validate the adaptability and scheduling performance of the Deep Q-Learning strategy.

5. Results and Analysis

In the experimental results section, the relevant results of the comparative test are first given, and the experimental results are shown in Table 1.

Table 1: Comparative experimental results

Method	Average Response Time	Resource Cost (normalized)	QoS Violation Rate (%)
Improved Q Network[9]	120.3	1.12	4.8
Reclaimer[10]	97.6	0.89	3.6
MSARS[11]	110.4	0.94	3.1
DRPC[12]	108.9	0.87	2.9
Ours	91.7	0.78	2.3

According to the experimental results, the proposed method outperforms all existing public methods across all evaluation metrics. It shows particularly strong performance in average response time. Specifically, the algorithm reduces the average response time to 91.7 milliseconds, which is significantly better than the closest baseline method, Reclaimer. This indicates that the Deep Q-Learning framework has stronger real-time responsiveness under dynamic load conditions. It enables more efficient elastic scaling of services and avoids the delay and resource lag problems found in traditional strategies.

In terms of resource cost, the proposed algorithm also achieves the lowest normalized cost value of 0.78, showing clear savings compared to other methods. This result reflects the algorithm's fine-grained control in resource scheduling. It can select cost-effective scaling actions based on the current system state and avoid resource redundancy and unnecessary scheduling overhead. This is due to the reward function used during training, which guides the model to learn cost-sensitive behavior while maintaining system performance.

For service quality assurance, the proposed method achieves a QoS violation rate of only 2.3 percent, which is better than all baseline methods. This shows that the algorithm has stronger stability and robustness in dynamic environments. It can effectively prevent service degradation caused by poor scheduling decisions. It is especially suitable for microservice systems with strict SLO (Service Level Objective) requirements. In contrast, traditional methods often suffer from limited policy updates or incomplete state modeling. These issues lead to high QoS violation rates under sudden load changes.

Across all three evaluation metrics, the experimental results demonstrate the proposed method's ability to optimize response time, resource cost, and service stability. This performance improvement comes from the Deep Q-Learning framework's ability to achieve multi-dimensional state awareness, adaptive policy generation, and long-term reward optimization. It overcomes the rigidity and generalization limitations of traditional rule-based or predictive strategies. It provides a more practical and effective solution for elastic scheduling in real-world microservice systems.

This paper also gives the impact of changes in the discount factor on the strategy convergence performance, and the experimental results are shown in Figure 2.

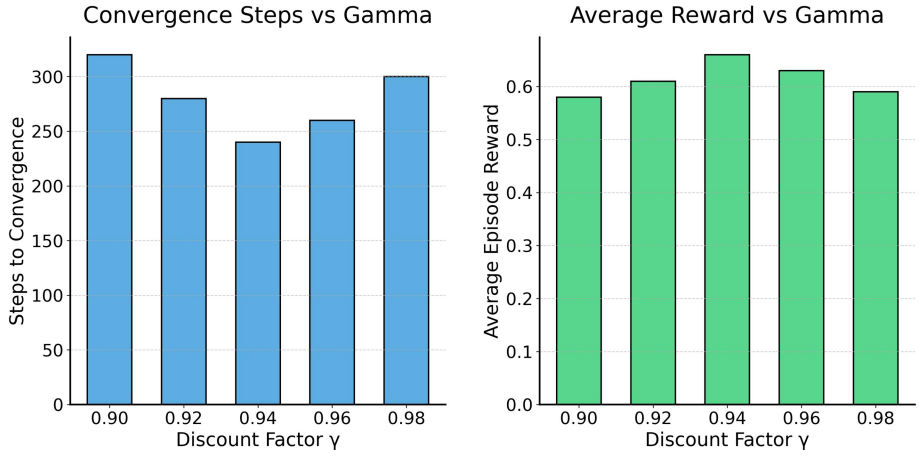


Figure 2. The impact of discount factor changes on strategy convergence performance

As shown in the experimental results in Figure 2, changes in the discount factor γ have a clear impact on policy convergence. When γ increases from 0.90 to 0.96, the number of steps required for convergence first decreases and then slightly increases. This suggests that within a moderately high range, the model can learn policies more efficiently and converge faster. The fewest steps are observed when γ is set to 0.94, indicating that this value may represent the best trade-off for the current task. It balances long-term rewards and short-term feedback, achieving the most efficient learning speed.

Meanwhile, the right subfigure shows the trend of average return under different γ values. As γ increases from 0.90 to 0.94, the average return improves gradually. This indicates that the model learns more effective resource scheduling strategies during this phase. However, when γ increases further to 0.96 and 0.98, the average return slightly decreases. This suggests that placing too much emphasis on long-term rewards may

reduce the stability of short-term decision-making. As a result, the strategy becomes less responsive to sudden workload changes in microservice systems, and the overall benefit decreases.

This trend supports the theoretical insight that the discount factor in reinforcement learning controls the time horizon of decision-making. A smaller γ may cause the policy to become short-sighted. In contrast, a larger γ may overlook immediate feedback, which reduces the practicality of the learned strategy. In microservice elasticity scheduling scenarios, system states change frequently, and resource constraints are dynamic. Therefore, choosing an appropriate γ value is crucial. It determines whether the model can balance response speed and resource efficiency across different time scales.

This paper also gives the impact of the initial learning rate setting on training efficiency, and the experimental results are shown in Figure 3.

As shown in Figure 3, the initial learning rate has a significant impact on the training efficiency of the Deep Q-Learning policy. When the learning rate increases gradually from 0.0001 to 0.001, the model's convergence speed improves. The required training steps drop from 460 to 320. This indicates that in a lower learning rate range, the model can complete policy optimization more quickly. A smaller learning rate may result in insufficient update steps, which slows convergence. Increasing the learning rate appropriately can accelerate convergence and improve training efficiency.

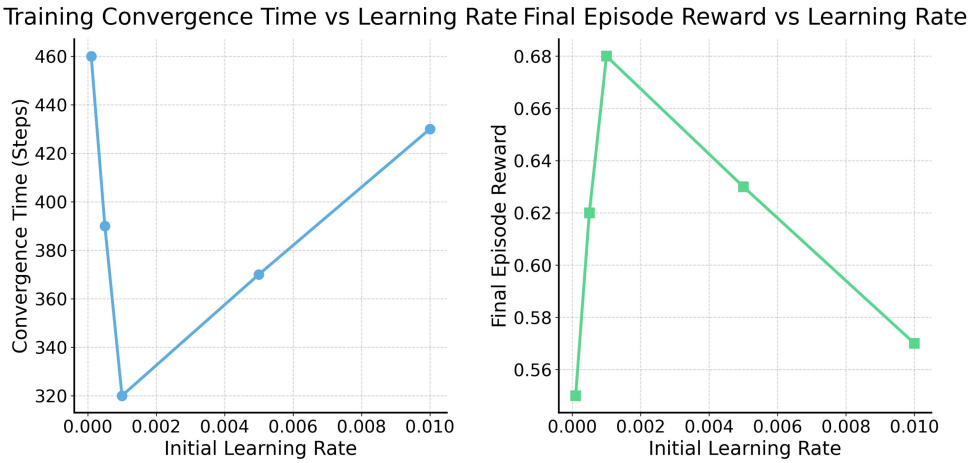


Figure 3. The impact of initial learning rate setting on training efficiency

However, when the learning rate increases further to 0.005 or 0.01, the training steps begin to rise again. This suggests that an excessively high learning rate may cause instability in the policy space. The model may experience oscillations or deviate from the optimal policy direction. This type of overshooting is common in reinforcement learning. It is especially pronounced when approximating the state-action value function. Large update steps make it difficult for the model to maintain a stable optimization path in a high-dimensional policy space.

In the right subfigure, the final average return follows a similar trend. When the learning rate is between 0.0001 and 0.001, the return gradually increases. This shows that the model can learn effective scaling strategies, which improves resource scheduling performance. However, when the learning rate becomes higher, the return drops significantly. This confirms that policy instability caused by high learning rates can directly reduce final performance. In elastic resource management tasks, this is critical. Poor decision quality not only affects system response but also leads to resource waste or service violations.

This paper also gives the impact of data sampling frequency changes on system performance, and the experimental results are shown in Figure 4.

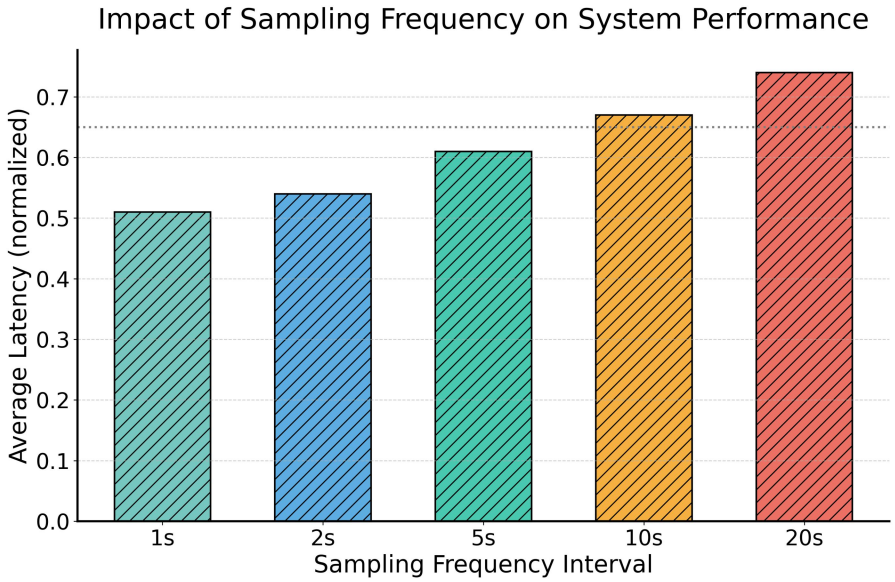


Figure 4. The impact of data sampling frequency changes on system performance

As shown in Figure 4, the data sampling frequency has a direct impact on system performance, especially in controlling average latency. The experimental results show that when the sampling frequency is set to 1 second or 2 seconds, the system maintains a low average latency, around 0.51 and 0.54, respectively. This indicates that high-frequency sampling enables the model to acquire system state information more promptly, which supports dynamic adjustment and fast response in reinforcement learning. Such a setting helps the model capture subtle changes in service load and resource usage, allowing for more precise scaling decisions.

As the sampling interval increases to 5 seconds, 10 seconds, and even 20 seconds, the average system latency rises significantly. The latency value grows from 0.61 to 0.74. A lower sampling frequency means that state updates become less timely. The model receives delayed environment information, which increases the mismatch between policy execution and the actual system state. Since reinforcement learning relies on environment feedback to optimize policies, low data update rates may lead to distorted state-action mappings and reduce decision-making efficiency.

In microservice elasticity scheduling scenarios, system states change frequently, and workloads can fluctuate suddenly. Therefore, real-time data sampling is essential. The experimental results further confirm that in dynamic environments, higher sampling frequencies help the model maintain an accurate perception of the system state. This leads to more effective scheduling strategies. In contrast, long sampling intervals weaken the model's ability to capture short-term behavioral patterns, which slows down policy response and affects overall service quality.

At the end of this paper, a graph showing the change of the loss function with epoch is given, as shown in Figure 5.

Figure 5 shows the trend of the loss function over 200 training epochs. The loss decreases steadily as the number of epochs increases. The decline is especially rapid during the first 50 epochs, indicating that the

model quickly learns an effective policy structure in the early training phase. This suggests that the constructed Deep Q-Network can effectively capture the mapping between states and actions in microservice resource scheduling tasks, enabling efficient policy fitting.

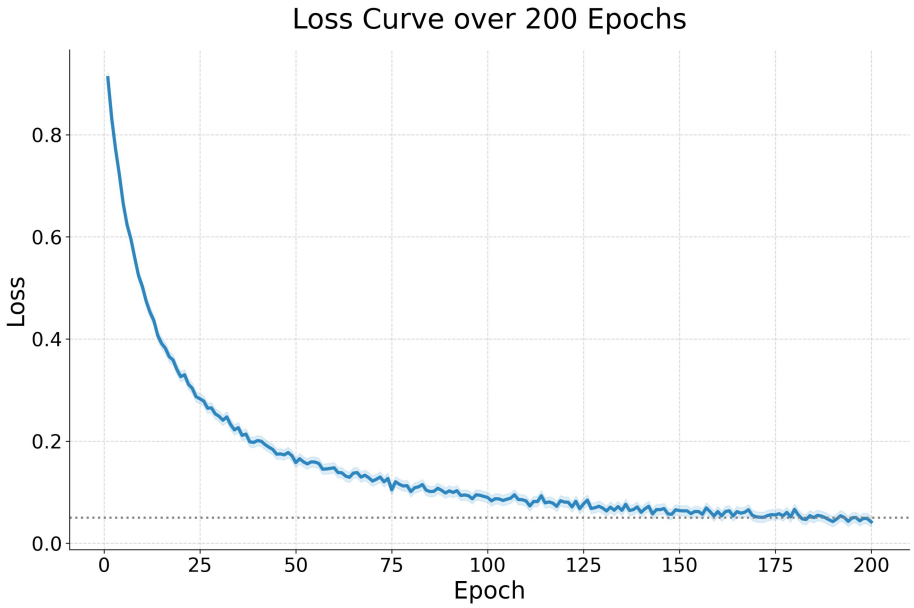


Figure 5. Loss function drop graph

In the later training stages, approximately between epoch 60 and epoch 200, the loss decreases more gradually. After around epoch 150, it stabilizes at a low value. This trend shows that the model converges to a near-optimal solution after sufficient training. It can then perform scaling actions with stability. The smoothness of the loss curve also indicates that parameter updates during training are stable, with no signs of oscillation, divergence, or overfitting.

The downward curve confirms the optimization capability of the reinforcement learning policy under long-term training. It also indirectly reflects the positive effects of reward function design, state space construction, and experience replay mechanisms. In microservice scenarios, where system states are complex and frequently changing, stable policy learning is essential. The smooth convergence of the loss curve supports the model's feasibility and stability for real-world deployment.

6. Conclusion

This study focuses on the problem of elastic resource management in microservice environments and proposes an automatic scaling and policy optimization method based on Deep Q-Learning. By modeling the microservice system as a Markov Decision Process, the method constructs a multi-dimensional state space and action set. Combined with mechanisms such as experience replay and soft target network updates, it enables adaptive optimization of dynamic resource scheduling. The proposed approach effectively balances system performance, resource cost, and service quality in complex, dynamic, and multi-objective scheduling scenarios. It also addresses the limitations of traditional rule-based and prediction-driven strategies in terms of timeliness, robustness, and generalization.

Experimental results show that the proposed method outperforms existing reinforcement learning and scheduling baselines across several key evaluation metrics. It is particularly effective in reducing response latency, improving resource utilization, and ensuring service-level objectives. Sensitivity analyses on discount factors, learning rates, and sampling frequencies further validate the model's structural stability and the interpretability of its scheduling strategies. These results provide a basis for controllable deployment in real systems. In addition, the study evaluates the model's adaptability under varying data scales and system state sensing frequencies, demonstrating its broad applicability across diverse microservice scenarios.

The core contribution of this research lies not only in methodological improvements and performance gains but also in the deep integration of reinforcement learning into resource scheduling for cloud-native architectures. The proposed method provides a practical solution for automated and intelligent system management. It has broad application potential in industries such as financial services, e-commerce, and industrial internet, where resource sensitivity and service fluctuation are common. It also offers technical support for the development of autonomous service systems. By introducing a reinforcement learning framework, the method moves beyond static thresholds and manual rules and opens a path toward long-term optimal policy exploration in complex system operations.

Future research may extend in several directions. One direction is to integrate graph neural networks and temporal modeling techniques to enhance the understanding of service call structures and time dependencies. Another direction is to explore multi-agent coordination and cross-node heterogeneous scheduling mechanisms to improve generalization in multi-cluster and multi-tenant systems. In practical deployments, reducing training costs, enabling online transfer, and supporting continual learning will also be important for scaling such algorithms in industrial settings. This study provides both a methodological foundation and experimental evidence for the evolution of intelligent scheduling systems, and it opens new directions for the deep integration of AI and cloud computing.

References

[1] S. Tuli, S. Ilager, K. Ramamohanarao and R. Buyya, "Dynamic Scheduling for Stochastic Edge-Cloud Computing Environments Using A3C Learning and Residual Recurrent Neural Networks", IEEE Transactions on Mobile Computing, vol. 21, no. 3, pp. 940-954, 2020.

[2] Jian Z, Xie X, Fang Y, et al. DRS: A deep reinforcement learning enhanced Kubernetes scheduler for microservice-based system[J]. Software: Practice and Experience, 2024, 54(10): 2102-2126.

[3] Guo F, Tang B, Tang M, et al. Deep reinforcement learning-based microservice selection in mobile edge computing[J]. Cluster Computing, 2023, 26(2): 1319-1335.

[4] Peng K, He J, Guo J, et al. Delay-aware optimization of fine-grained microservice deployment and routing in edge via reinforcement learning[J]. IEEE Transactions on Network Science and Engineering, 2024.

[5] Santos J, Zaccarini M, Poltronieri F, et al. Efficient microservice deployment in kubernetes multi-clusters through reinforcement learning[C]//NOMS 2024-2024 IEEE Network Operations and Management Symposium. IEEE, 2024: 1-9.

[6] H. Chen, X. Zhu, D. Qiu, L. Liu and Z. Du, "Scheduling for Workflows with Security-Sensitive Intermediate Data by Selective Tasks Duplication in Clouds", IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 9, pp. 2674-2688, 2017.

[7] S. Tuli, S. Ilager, K. Ramamohanarao and R. Buyya, "Dynamic Scheduling for Stochastic Edge-Cloud Computing Environments Using A3C Learning and Residual Recurrent Neural Networks", IEEE Transactions on Mobile Computing, vol. 21, no. 3, pp. 940-954, 2020.

[8] Matos G H M, Carvalho M, Macedo D F. Container-Based Microservice Scheduling Using Reinforcement Learning in Distributed Cloud Computing[C]//2024 IEEE Latin-American Conference on Communications (LATINCOM). IEEE, 2024: 1-6.

[9] Kim Y, Park J, Yoon J, et al. Improved Q network auto-scaling in microservice architecture[J]. Applied Sciences, 2022, 12(3): 1206.

[10] Fettes Q, Karanth A, Bunescu R, et al. Reclaimer: A reinforcement learning approach to dynamic resource allocation for cloud microservices[J]. arXiv preprint arXiv:2304.07941, 2023.

[11] Hu K, Wen L, Xu M, et al. MSARS: A Meta-Learning and Reinforcement Learning Framework for SLO Resource Allocation and Adaptive Scaling for Microservices[J]. arXiv preprint arXiv:2409.14953, 2024.

[12] Bai H, Xu M, Ye K, et al. Drpc: Distributed reinforcement learning approach for scalable resource provisioning in container-based clusters[J]. IEEE Transactions on Services Computing, 2024.