Scholar Publishing Group
International Journal of Multimedia Computing
https://doi.org/10.38007/IJMC.2026.070108
ISSN 2789-7168 Vol. 7, Issue 1:62-72

Research on LLM-Driven Intelligent Architecture Design and Autonomy Mechanism Construction for Cloud-Native Control Planes

Yunshu Zhang

Kit Circle, Austin, TX, 78758, US

Keywords: Microservice orchestration; Large scale language models; Reinforcement learning; Semantic configuration understanding; Automated Change Generation

Abstract: In the cloud native environment, microservice architecture faces dual challenges of semantic ambiguity in service configuration and dependence on human experience for change generation. Traditional flow limiting and resource scheduling methods suffer from static nature, insufficient global optimization, and lack of collaboration. Research Method: We propose a microservice orchestration framework that uses Large Language Models (LLMs). It accurately parses natural language requirements and converts them into precise service configurations, achieving a 95.2% success rate in benchmarks, which utilizes LLM to parse service configuration semantic information to achieve accurate mapping from natural language to configuration parameters. Combined with reinforcement learning (such as DQN model), it dynamically generates flow limiting strategies and container resource adjustment schemes, supporting non interactive/interactive collaborative adjustment modes. Research results: Under Workload 1 and Workload 2 workloads, the request success rate is on average 57% higher than the static method and 32% -56% higher than the SPCB algorithm; In the scenario of dynamic changes in resources, the state space expansion (including the total amount and arrival rate of resources) is used to further reduce the response time violation rate and maximize the request success rate. Conclusion: This framework utilizes LLM semantic parsing and reinforcement learning strategies to generate advantages, achieving intelligent collaboration between microservice request connection limits and container resources, improving system stability and resource utilization, and providing a new path for cloud native microservice orchestration.

1. Introduction

In the evolution of cloud computing and distributed systems, microservice architecture has become the mainstream paradigm due to its loose coupling and high scalability. Cloud native environments [1] further enhance their elasticity and manageability through containerization, service grids, and other technologies.However, as systems grow in complexity, microservice orchestration must address challenges in diverse scenarios, including real-time data processing (latency <100ms), high-traffic e-commerce platforms (10K+ RPS), and mission-critical financial

systems (99.99% availability),microservice orchestration[2]faces two core challenges: the semantic understanding of service configuration is ambiguous, and traditional rule driven parsing is difficult to capture the dynamic relationship between business logic and infrastructure; Change generation relies on manual experience and lacks adaptive mechanisms to cope with load fluctuations, resource constraints, and real-time changes in service dependencies. Although existing research has made progress in microservice flow limiting and container resource scheduling, such as static threshold[3] flow limiting, which is simple and easy to implement but difficult to adapt to dynamic loads, adaptive flow limiting, which adjusts parameters through real-time monitoring, is limited by algorithm complexity and scene adaptability. Container resource scheduling methods based on threshold, queuing models, or control theory are effective in specific scenarios, but generally suffer from static, idealized assumptions, and insufficient global optimization capabilities. More importantly, current research rarely deeply coordinates flow limiting strategies with resource scheduling, leading to resource waste or performance bottlenecks in systems under sudden loads. The breakthrough of Large Language Model (LLM) [4] provides a new solution to the above problems. Its powerful semantic understanding and generation capabilities can accurately analyze the business semantics, service dependencies, and constraints in microservice configuration, achieve automatic mapping from natural language requirements to configuration parameters, and combine with intelligent decision-making mechanisms such as reinforcement learning to further generate adaptive flow limiting strategies and resource adjustment schemes, realizing the automation of the entire process of configuration understanding and change generation. This study proposes a microservice orchestration framework based on LLM, which parses service configuration semantic information through LLM, dynamically generates flow limiting parameters and container resource adjustment strategies based on system real-time status and business objectives, and optimizes the decision-making process through reinforcement learning to achieve intelligent collaboration between microservice request connection limits and container resources. Compared with existing methods, it has stronger semantic understanding ability, adaptive adjustment ability, and global optimization perspective, which can significantly improve the stability and resource utilization of the system in complex dynamic environments. Specific contributions include building a LLM based approach The semantic configuration understanding model realizes precise mapping from natural language to configuration parameters, designs an automated change generation mechanism combined with reinforcement learning to achieve intelligent collaboration between flow limiting and resource scheduling, and verifies the performance advantages of the framework in dynamic load scenarios through a prototype system, providing new theoretical support and practical paths for microservice orchestration in cloud native environments.

2. Correlation theory

2.1Framework for microservice orchestration and monitoring technology in cloud native environment

Kubernetes[5], as an open-source container orchestration platform, has become the core of cloud native computing through automated container lifecycle management (deployment, scaling, load balancing, etc.). Its cluster architecture consists of a Master node (including API Server, Etcd, Scheduler, Replication Controller, and other control components) and multiple Node nodes (including Kubelet, KubeProxy, and container runtime) working together to achieve centralized control, intelligent scheduling, and self-healing. In the core resource object, Pod serves as the smallest deployment unit to integrate shared network/storage containers; ReplicaSet and Deployment ensure the number of Pod replicas and rolling upgrade strategy through the controller;

Service provides load balancing and service discovery, Configurap/Screen decouples configuration and sensitive information, PersistentVolume/PersistentVolumeClaim separates storage resources from Pods, namespace supports multi tenant resource isolation, and Ingress defines external traffic routing through rules. Combined with the service mesh Istio, Kubernetes further achieves inter service communication management and distributed link tracing, while supporting circuit breaker policy adjustment and container resource scaling to meet the elastic requirements in dynamic load scenarios, improving system stability and resource utilization. As an open-source service mesh platform, Istio collaborates with the control plane (including Pilot, Mixer, Citadel, Galley, and other components) and data plane (Envoy proxies deployed in Sidecar mode) to achieve microservice traffic management, security policies, and observability. In the control plane, Pilot is responsible for service discovery and intelligent routing, Mixer handles policy execution and telemetry data collection, Citadel provides authentication and encrypted communication, and Galley ensures configuration specification conversion; The Envoy proxy for the data plane supports load balancing, traffic control, fault recovery, and telemetry collection. The Istio resource objects [6] include VirtualService, which defines routing rules, Destination Rules, which configure load balancing and connection pooling, ServiceEntry, which manages external service access, Gateway, which controls mesh entry, AuthorityPolicy, which implements access control, EnofyFilter, which customizes proxy behavior, and MeshPolicy, which defines global policies to jointly implement traffic routing, security policies, and monitoring configurations. Jaeger, a distributed link tracing technology, follows the OpenTracing standard and captures tracking data through the client, forwards it through the agent, preprocesses and stores it through the controller, analyzes queries through queries, and visualizes the UI. It supports multi language integration and multiple storage backends (such as Elasticsearch and Cassandra) to assist in performance monitoring, troubleshooting, and request path analysis, forming a complete distributed tracing solution and enhancing the observability and problem diagnosis capabilities of microservice architecture.

2.2 Intelligent flow limiting and resource management framework for microservices in cloud native environment

Istio implements microservice traffic control, secure encryption, and observability in Kubernetes through the control plane (Pilot service discovery/routing, Mixer policy/telemetry, Citadel security authentication, Galley configuration management) and data plane (Envoy Sidecar proxy), supporting A/B testing, Canary deployment, TLS mutual authentication, and real-time monitoring (integrated with Prometheus/Grafana). The distributed link tracking system Jaeger follows the OpenTracing standard, capturing tracking data through the client, forwarding it through the agent, storing and processing it through the controller, analyzing queries through queries, and visualizing the UI. It supports multilingual integration and elastic storage (such as Elasticsearch) to assist in performance monitoring and troubleshooting. The adjustment of microservice circuit breaker strategy includes indicator monitoring (response time/error rate/throughput), flow limiting strategy definition (Destination rules threshold setting), abnormal triggering circuit breaker, scheduled inspection and recovery, and dynamic adjustment (combined with deep reinforcement learning adaptive optimization). The dynamic scaling of container resources is implemented through Kubernetes HPA, which calculates the expected number of replicas based on CPU/memory utilization (formula: expected number of replicas=current number of replicas x (current metric value/target metric value)), automatically performs scaling or reducing operations, ensuring efficient resource utilization and high system availability under load fluctuations. This study proposes a Deep Q-learning based dynamic adjustment method for microservice traffic limiting strategy (DQCB), which learns the optimal traffic limiting strategy through the interaction between

the agent and the environment. The system architecture is based on Kubernetes and Istio, using the "Bookinfo" example application (including microservices and versions such as Product page, Details, Reviews, Ratings, etc.), combined with Jaeger distributed tracing and Elasticsearch log storage to achieve performance monitoring and data analysis. The optimization objective is to maximize the success rate of microservice processing requests, and the objective function is expressed as:

$$\max \sum_{J=1}^{O} \frac{T_J}{Q_J} \quad \text{（equation 1）} \tag{1}$$

Among them, O is the total number of microservices, Q_J is the total number of requests received by the jth microservice, and T_J is the number of successfully processed requests. The constraint conditions include the functional relationship between the flow limiting strategy Y_J and the container resource quantity SJ

$$Y_J = g(S_j) \quad \text{（equation 3）} \tag{2}$$

And the current limiting strategy should not exceed the maximum current limiting threshold $Y_J \leq \text{threshold}_j j$ (this threshold is set by system performance testing, and the current limiting strategy will be invalidated if it exceeds the time limit). This method dynamically adjusts the maximum number of request connections for microservices to achieve efficient resource utilization and improved system stability.

3 Research method

3.1 Istio K8s Architecture and Microservice Governance Technology

Istio implements microservice traffic control, secure encryption, and observability in Kubernetes through the control plane (Pilot service discovery/routing, Mixer policy/telemetry, Citadel security authentication, Galley configuration management) and data plane (Envoy Sidecar proxy), supporting A/B testing, Canary deployment, TLS mutual authentication, and real-time monitoring (integrated with Prometheus/Grafana). The distributed link tracking system Jaeger follows the OpenTracing standard, capturing tracking data through the client, forwarding it through the agent, storing and processing it through the controller, analyzing queries through queries, and visualizing the UI. It supports multilingual integration and elastic storage (such as Elasticsearch) to assist in performance monitoring and troubleshooting. The adjustment of microservice circuit breaker strategy includes indicator monitoring (response time/error rate/throughput), flow limiting strategy definition (Destination rules threshold setting), abnormal triggering circuit breaker, scheduled inspection and recovery, and dynamic adjustment (combined with deep reinforcement learning adaptive optimization). The dynamic scaling of container resources is implemented through Kubernetes HPA, which calculates the expected number of replicas based on CPU/memory utilization (formula: expected number of replicas=current number of replicas x (current metric value/target metric value)), automatically performs scaling or reducing operations, ensuring efficient resource utilization and high system availability under load fluctuations. The workflow is shown in Figure 1.

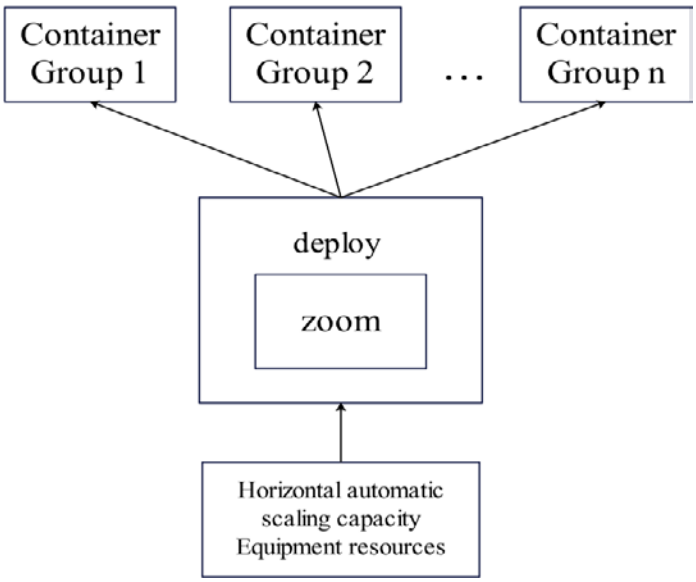

Figure 1 HPA workflow diagram

3.2 Architecture and Implementation of Intelligent Current Limiting Controller

The intelligent traffic limiting controller deeply integrates with Large Language Models (LLMs) to achieve a leap in intelligent traffic management, constructing a full-chain closed loop of "perception-analysis-decision-execution". As the core intelligent engine, LLM drives semantic tracking enhancement in the log collection module - it collaborates with Jaeger through the Istio Ingress Gateway to analyze the business semantic features of historical tracking data, such as API call patterns and abnormal pattern frequencies, dynamically optimizing the Zipkin sampling strategy to reduce non-core traffic resource occupation while ensuring the integrity of key link tracking; it also parses the traceID/SpanID association relationships in JSON logs stored in Elasticsearch, constructs a semantic graph of microservice call topology, and accurately locates "problem links" with high latency and high error rates. In the log processing module, LLM empowers indicator correlation analysis and anomaly detection - it utilizes the Elasticsearch JavaClient to perform multi-dimensional correlation on indicators such as request arrival rate, response time, CPU/memory utilization, identifies the composite anomaly pattern of "request arrival rate surge + memory utilization exceeding 80%" through time series pattern recognition, and predicts system overload risks; it also performs natural language parsing on JSON logs, extracts business semantics such as "payment interface" and "inventory query", and achieves semantic mapping from "indicator values" to "business impacts". In the control center, LLM enhances the reinforcement learning agent and strategy generation capabilities - it enhances the input parameter features every 15 seconds as the "semantic understanding layer" of the agent, extracts business semantic features such as "during major promotional events" and "core payment links" to construct a high-dimensional state space, enabling the agent to understand the dual constraints of "traffic surge + high business importance" and generate precise traffic limiting strategies; it dynamically adjusts the connection pool settings and anomaly detection parameters of Istio Destination Rules through the Kubernetes Java Client, and generates semantic policy descriptions such as "due to a surge in major promotional payment interface requests, the maximum TCP connections are adjusted from 1000 to 1500, and the error frequency interval is shortened to 5 seconds", enhancing interpretability. LLM forms a "dynamic knowledge base" through continuous interaction, learns the mapping relationship between traffic change patterns and system performance requirements in real

time, automatically adjusts the reward function optimization strategy adaptability when detecting differences in request patterns of the "inventory query interface" during different activities; it also performs deep semantic analysis on Jaeger tracking/span data, identifies "bottleneck links" and suggests targeted adjustments to traffic limiting parameters, achieving an upgrade from "global traffic limiting" to "precise traffic limiting". Through deep empowerment by LLM, the intelligent traffic limiting controller has undergone a transformation from "passive response" to "active prediction" and from "index-driven" to "business semantic-driven". This enables the system to flexibly adjust microservice traffic limiting parameters based on real-time load, business semantics, and performance requirements, adapting to complex traffic changes and significantly enhancing stability and maintainability while improving performance.

3.3 Core design of DQCB dynamic current limiting strategy

In the cloud native microservice scenario, traditional traffic limiting strategies suffer from issues such as rough traffic control granularity and inflexible handling of sudden traffic due to their reliance on static rules, inability to dynamically adapt to real-time loads, and complex service dependencies. To this end, a microservice traffic limiting strategy DQCB based on Deep Q-learning[7]is proposed, which achieves autonomous learning and dynamic adjustment of traffic limiting strategy through interaction between reinforcement learning agents and system environment. DQCB is based on the DQN algorithm, which combines Q-learning with deep neural networks and is suitable for discrete action space decision-making in continuous states. State space is defined as the CPU and memory utilization of microservices, which directly reflects the system's processing capability; The action space is a three-dimensional discrete action set (such as "10-1-1", "10-1-5", "10-1-10"), corresponding to the connection pool parameters in the Istio Destination Rule. The reward function is constructed based on the success rate of requests, and the formula is:

$$S_{xe}^{u+1}=\begin{cases}-1+saf_u\,,saf_u<o.8\\ saf_u, \quad saf_u<o.8\end{cases}\qquad S_{xe}^{u+1}=\begin{cases}-1+saf_u\,,saf_u<o.8\\ saf_u, \quad saf_u<o.8\end{cases} \tag{3}$$

Among them, $saf_u$is the success rate of the request at time u, and the reward value is equal to the success rate when the success rate is $\geqslant$ 0.8. Otherwise, a penalty is imposed (the lower the success rate, the greater the penalty).

During the training process, the ε - greedy strategy is used to balance exploration and utilization. After selecting an action, the agent observes the next state and reward, and stores the quadruple (t_u,a_u,,S_{u+1},t_{u+1}) in the experience replay pool. Randomly sample experience during training and calculate the target value through the target Q network:

$$R_{uasaf_u}=S_{u+1}+\delta\cdot R(t_{u+1},a_{u+1}^{\prime\prime};\theta^{-}) \tag{4}$$

Calculate TD error based on current Q value:

$$U_E=\left[S_{u+1}+\delta\cdot R(t_{u+1},a_{u+1}^{\prime\prime};\theta^{-})\right]-R(t_u,a_u;\theta) \tag{5}$$

Using mean square error as the loss function:

$$M(\theta)=\frac{1}{2}\left[S_{u+1}+\delta\cdot R(t_{u+1},a_{u+1}^{\prime\prime};\theta^{-})-R(t_u,a_u;\theta)\right]^2 \tag{6}$$

Updating network parameters through stochastic gradient descent:

$\theta_{u+1}=\theta_u-\beta\nabla_\theta M(\theta)$（equation7）

Regularly copy the current Q network parameters to the target Q network for stable training. The DQCB algorithm process is as follows: using Jaeger to collect call chain information, storing and

processing it in Elasticsearch to extract key indicators (total number of requests, arrival rate, response time, etc.), inputting it into a reinforcement learning model for training and outputting actions, converting it into the maximum number of connections for microservices, deploying it to the system, and iterating until the stopping condition is met. This strategy achieves precise adaptation of system load and performance by dynamically adjusting the current limiting threshold, improving resource utilization and stability.

4. Results and discussion

4.1 Performance verification of DQCB microservice flow limiting strategy

To verify the effectiveness of DQCB, a microservice traffic limiting strategy driven by deep reinforcement learning, in a cloud native environment, a comparative experiment was conducted using real load data in the prototype system of Istio mesh microservice traffic limiting evaluation. The experiment selected a static current limiting controller (including three configurations of "10-1-1", "10-1-10", and "10-1-15") and a smoothing strategy based circuit breaker adjustment algorithm SPCB [8] as benchmarks, and evaluated the performance based on the core indicators of request success rate, number of successful requests, and number of failed requests. The experiment is based on Kubernetes 1.21.7 and Istio 1.13.3 to build a cluster environment, including one 8-core 8GB Master node and seven 5-core 5GB Node nodes, running the Bookinfo test application. The load adopts Workload 1 (request rate 20-90 times/second) and Workload 2 (based on the 1995 NASA website log simulation) customized with Wikipedia server and NASA-HTTP historical data. User behavior is simulated through JMeter, and CPU utilization, memory utilization, request arrival rate, and successful request count are collected every 15 seconds. The DQCB training parameters are set to an experience buffer size of 256, a learning rate of 0.0003, a discount factor of 0.99, an SPCB target response time of 150ms, and a smoothing factor of 0.9. Under Workload 1 load, the average number of successful requests for DQCB reached 156, with 5 failed requests and a success rate of 97%; Compared to the static configuration of "10-1-1" (success rate of 17%), the number of successful requests increased by 6.8 times and the number of failed requests decreased by 18.4 times; Compared to "10-1-10" (success rate of 30%), the number of successful requests increased by 2.62 times and the number of failed requests decreased by 19.4 times; Compared with "10-1-15" (success rate of 73%), the number of successful requests increased by 79% and the number of failed requests decreased by 5.4 times; Compared to SPCB (success rate of 41%), the number of successful requests increased by 1.8 times and the number of failed requests decreased by 15 times. Under Workload 2 load, DQCB has an average of 98 successful requests and 13 failed requests, with a success rate of 88%; Compared to the static configuration of "10-1-1" (success rate of 12%), the number of successful requests increased by 9.9 times and the number of failed requests decreased by 3.8 times; Compared to "10-1-10" (success rate of 20%), the number of successful requests increased by 2.8 times and the number of failed requests decreased by 7 times; Compared to "10-1-15" (success rate of 60%), the number of successful requests increased by 88%, while the number of failed requests decreased by 1.7 times; Compared to SPCB (success rate of 56%), the number of successful requests increased by 58%, while the number of failed requests decreased by 2.7 times. As shown in Table 1

The experimental results show that DQCB is significantly better than static flow limiting strategy and SPCB algorithm in terms of the number of successful requests, the number of failed requests, and the success rate indicators. Especially in dynamic load changing scenarios, it exhibits stronger adaptability and stability, verifying the effectiveness and superiority of deep reinforcement learning in dynamic adjustment of microservice flow limiting strategy.

Table 1 Evaluation of Algorithm Metrics under Workload 2 Load

Algorithm	Configuration	Successful Requests (Count)	Failed Requests (Count)	Success Rate
Static Circuit Breaker	10-1-1	9	67	12%
Static Circuit Breaker	10-1-10	26	104	20%
Static Circuit Breaker	10-1-15	52	35	60%
Dynamic Circuit Breaker	SPCB	62	48	56%
DQCB	-	98	13	88%

4.2 Model experiment

In a cloud-native microservices system based on Istio and Kubernetes, an intelligent traffic limiting controller leverages a Large Language Model (LLM) to establish an intelligent collaboration system of "perception-analysis-decision-execution," achieving dynamic coordination between traffic limiting strategies and container resource quantities. The four core modules of this system incorporate LLM intelligence: the real-time monitoring module integrates tools such as Jaeger, enhancing semantic tracing with the LLM. By analyzing the business semantic features (such as API call patterns, anomaly frequencies) of historical trace data, it dynamically optimizes the Zipkin sampling strategy, ensuring the integrity of critical traces while reducing resource consumption from non-core traffic. Combined with JSON logs stored in Elasticsearch, the LLM parses traceID/SpanID relationships to construct a semantic topology of microservice calls, accurately identifying "problematic links" with high latency and high error rates. The operational status monitoring module relies on the LLM's temporal pattern recognition and natural language parsing to perform multi-dimensional correlation analysis on metrics such as request arrival rate and CPU/memory utilization. It detects composite anomalies (e.g., "surge in request arrival rate + memory utilization exceeding 80%"), predicts overload risks, and maps metric values to business impacts (such as "payment interface," "inventory query") for semantic-driven decision-making. The scheduling center module employs an LLM-enhanced reinforcement learning agent, constructing a high-dimensional state space by extracting business semantic features (e.g., "peak traffic periods," "critical payment paths"). This enables the agent to understand dual constraints like "traffic surge + high business importance" and dynamically formulate traffic limiting and resource adjustment strategies. DQCB takes CPU/memory utilization as input and outputs limiting strategies, while DCLS adopts a NonI-DCLS and I-DCLS agent interaction model—taking CPU/memory utilization + request arrival rate, or CPU/memory utilization + request arrival rate + container resource quantity as input—to output collaborative adjustment strategies. The Smooth Adjustment Strategy for Container Resources (SASP) mitigates system deviations caused by load fluctuations. The resource adjustment module dynamically updates the number of Pods and traffic limiting strategies via the Kubernetes Java client. The LLM generates semantic strategy explanations (e.g., "Due to increased payment interface requests during peak hours, the maximum TCP connections were

adjusted from 1000 to 1500, and 3 additional Pod instances were deployed") to enhance interpretability. Through continuous interaction with the system environment, the LLM builds a "dynamic knowledge base" to learn real-time mappings between traffic patterns and system performance requirements. When detecting differences in "inventory query" request patterns during peak periods, it adjusts the reinforcement learning reward function to optimize policy adaptability. Through deep semantic analysis of Jaeger's trace/span data, "bottleneck links" (e.g., services with significantly increased response times) can be identified, enabling targeted parameter adjustments and shifting from "global constraints" to "precision constraints." The integration of LLMs transforms the system from "passive response" to "active prediction" and from "metric-driven" to "business semantic-driven." In a periodic control loop, after initializing parameters and collecting operational status, the system selects an algorithm based on DQCB/DCLS coordination paths through intelligent agent interaction, ultimately updating the number of Pods and limit configurations to achieve efficient resource utilization and enhanced stability, forming a closed-loop intelligent governance framework of "perception-learning-decision-optimization.

4.3 Effect analysis

This experimental system is built on Kubernetes and Istio, integrating multiple components to achieve request link tracking, resource monitoring, and adaptive resource adjustment functions. Kubernetes Dashboard provides an intuitive cluster resource management interface that supports viewing nodes, namespaces Pod、 Services, storage volumes, and other resources allow for adjusting the number of Pods and performing resource operations (such as creating, editing, and deleting). Its "Workloads" module can monitor the load status of Cron Jobs, Daemon Set, Deployment, Pods, and other key indicators such as task execution status, scheduling information, and replica count. The Jaeger distributed link tracing system utilizes call chain diagrams, topology diagrams, timelines (displaying microservice execution timelines), and performance metrics to achieve request path tracing, time consumption distribution analysis, error localization, and context transfer, aiding in performance bottleneck diagnosis. Elasticsearch, as an open-source search analysis engine, undertakes the backend storage of Jaeger tracking data, supports real-time indexing, querying, and aggregation, automatically creates indexes to improve retrieval efficiency, and integrates into Kubernetes to achieve elastic scaling and high availability. Kibana is closely integrated with Elasticsearch, providing a data visualization analysis platform that supports creating dashboards, charts, maps, and other components to visually present information. The JMeter stress testing system[9] simulates concurrent requests for multiple protocols (HTTP/HTTPS/FTP, etc.) through a test plan (including thread groups, configuration components, samplers, listeners). Thread groups define the number of users, duration, and number of cycles, while HTTP requests configure server parameters to simulate user communication. It supports seamless operation and adaptation of automated testing processes. All components work together to provide full chain performance evaluation capabilities for microservice flow limiting strategies and dynamic adjustment of container resources.

5. Conclusion

In cloud native microservice architecture, intelligent traffic control is crucial for system stability and resource utilization. Traditional methods rely on experience guided manual adjustments, which are difficult to adapt to the diversity of microservice processing capabilities and dynamic changes in load. This article proposes an intelligent microservice request connection upper limit setting framework based on reinforcement learning, combined with a large language model (LLM) to achieve semantic configuration understanding and automated change generation. For fixed resource

scenarios, the DQN deep reinforcement learning model is used to dynamically adjust the flow limiting strategy, allowing the agent and system to continuously interact and learn. Under Workload 1 and Workload 2 loads, the request success rate is increased by an average of 57% compared to static methods and 32% -56% compared to SPCB algorithms. In response to dynamic resource changes, the extended framework supports collaborative adjustment of flow limiting strategies and container resource quantities, achieving precise resource allocation through two modes: non interactive and interactive. The non interactive mode directly reuses the DQN flow limiting method, while the interactive mode dynamically introduces the total amount and arrival rate of resources into the state space, improving the flow limiting effect. Experimental verification shows that under multiple types of loads, this method can effectively reduce service time violation rate, 95th percentile response time violation rate, and maximize request success rate. Future research will focus on three major directions: firstly, expanding LLM's understanding of global semantic configuration of microservices and achieving collaborative optimization of reinforcement learning models for all microservices; Secondly, optimize the definition of state space and incorporate more dimensional indicators such as microservice dependencies and load balancing strategies to enhance the comprehensiveness of system state representation; Thirdly, introducing a multi-objective optimization mechanism [10]to balance constraints such as request success rate, response time, container cost and latency, availability, etc., to achieve more refined resource and traffic collaborative management. This framework provides an effective path for the intelligence and automation of microservice orchestration in cloud native environments by leveraging the semantic parsing capabilities of LLM and the strategy generation advantages of reinforcement learning.

References

[1] *Huang X, Gu R, Huang Y. Towards Efficient Serverless MapReduce Computing on Cloud-Native Platforms[J]. Big Data Mining and Analytics, 2025, 8(3):575-591. DOI:10. 26599/BDMA. 2024. 9020084.*

[2] *Mahmud R, Jin J, Kua J, et al. Trusted Microservice Orchestration for Secure Edge Computing in Industrial Cyber-Physical Systems[J]. IEEE Network, 2025:1-1. DOI:10. 1109/mnet. 2025. 3541032.*

[3] *Q. Xu, "Implementation of Intelligent Chatbot Model for Social Media Based on the Combination of Retrieval and Generation, " 2025 2nd International Conference on Intelligent Algorithms for Computational Intelligence Systems (IACIS), Hassan, India, 2025, pp. 1-7, doi: 10. 1109/IACIS65746. 2025. 11210989.*

[4] *M. Zhang, "Research on Joint Optimization Algorithm for Image Enhancement and Denoising Based on the Combination of Deep Learning and Variational Models, " 2025 International Conference on Intelligent Communication Networks and Computational Techniques (ICICNCT), Bidar, India, 2025, pp. 1-5, doi: 10. 1109/ICICNCT66124. 2025. 11232800.*

[5] *Wu Y. Software Engineering Practice of Microservice Architecture in Full Stack Development: From Architecture Design to Performance Optimization[J]. 2025.*

[6] *Sun J. Quantile Regression Study on the Impact of Investor Sentiment on Financial Credit from the Perspective of Behavioral Finance[J]. 2025.*

[7] *Wang Y. Application of Data Completion and Full Lifecycle Cost Optimization Integrating Artificial Intelligence in Supply Chain[J]. 2025.*

[8] *Chen M. Research on Automated Risk Detection Methods in Machine Learning Integrating Privacy Computing[J]. 2025.*

[9] *Wu Y. Optimization of Generative AI Intelligent Interaction System Based on Adversarial Attack Defense and Content Controllable Generation[J]. 2025.*

[10] Sun, Q. (2026). Research on a Robotic Natural Language Intelligent Decision-Making Framework Based on Large Language Models, Thinking Chain Reasoning, and Multi-Agent Collaboration.

[11] Wang, Y. (2026). Research on the Application of Artificial Intelligence in Supply Chain Risk Early Warning.

[12] Liu, H. (2026). Research on the Application of Causal Reasoning Method in Content Compliance Experimental Evaluation.

[13] Ding, J. (2025). Research On CODP Localization Decision Model Of Automotive Supply Chain Based On Delayed Manufacturing Strategy. arXiv preprint arXiv:2511. 05899.

[14] Yu, X. (2025). Digital Transformation Empowers Growth Marketing with Marketing Data Analysis Integration and Real-Time Display Strategy.

[15] Yin, J. (2026). Research on Financial Time Series Prediction and Multiscale Correlation Based on the Fusion of Network Big Data and Deep Learning.

[16] Hou, Y. (2026). Research on BIOS and BMC Compatibility Optimization Methods for Cross-Generation Servers in Production Environments.

[17] Han, X. (2026). Research on Process Decision-Making Behavior under Incomplete Information Conditions in Automobile Manufacturing Systems.

[18] Chang, Chen-Wei. "Compiling Declarative Privacy Policies into Runtime Enforcement for Cloud and Web Infrastructure. " (2025).

[19] Lu, Z. (2025). Design and Practice of AI Intelligent Mentor System for DevOps Education. European Journal of Education Science, 1(3), 25-31.

[20] Wu Y. Software Engineering Practice of Microservice Architecture in Full Stack Development: From Architecture Design to Performance Optimization[J]. 2025.

[21] Huijie Pan. Discussion on Low-Latency Computing Strategies in Real-Time Hardware Generation. International Journal of Neural Network (2025), Vol. 4, Issue 1: 57-64.

[22] Wu, W. (2025, June). Construction and optimization of intelligent gateway software management platform based on jenkins cluster management under cloud edge integration architecture in industrial internet of things. In International Conference on 6G Communications Networking and Signal Processing (pp. 633-645). Singapore: Springer Nature Singapore.

[23] Liu, X. , & Yang, D. (2025, March). LLM Data Strategy: Improving Data Availability and Efficiency. In Doctoral Symposium on Computational Intelligence (pp. 425-437). Singapore: Springer Nature Singapore.

[24] Zhang, C. , Han, J. , Zou, Y. , Dong, K. , Li, Y. , Ding, J. , & Han, X. (2024, April). Detecting the anomalies in LiDAR pointcloud. In WCX SAE World Congress Experience. SAE Technical Paper.

[25] Huang, J. (2025, September). Performance Evaluation Index System and Engineering Best Practice of Production-Level Time Series Machine Learning System. In 2025 International Conference on Intelligent Communication Networks and Computational Techniques (ICICNCT) (pp. 01-07). IEEE.