Scholar Publishing Group
International Journal of Big Data Intelligent Technology
https://doi.org/10.38007/IJBDIT.2026.070206
ISSN 2790-0932 Vol. 7, Issue 2: 51-60

High Performance Load Balancing Method for Distributed Inference of Large-Scale Language Models

Yiling Yuan

Information Networking Institute, Carnegie Mellon University, Pittsburgh, 15213, Pennsylvania, United States

Keywords: Large language model; Distributed inference; Load balancing; Request scheduling; Service level target; Key-value caching

Abstract: Large-scale language model inference services are rapidly developing in online question-answering, intelligent agents and enterprise knowledge retrieval. Inference requests have many differences in the size of the input and output, when they are sent, and how much performance is expected. The above three strategies (traditional round-robin, least-connection, and single-instance batch processing) cannot achieve both high throughput and low tail latency simultaneously. Therefore, a prediction-aware load-balancing method for mixed long- and short-request loads has been developed, and in order to achieve this, request cost prediction, instance pressure normalisation, stage-aware routing, batch adaptive adjustment and service-level-effective throughput evaluation have been integrated into a unified framework. Based on real-world data from publicly available English literature over the past three years, two statistical charts and three comparison tables are generated to summarize the size of throughput, service capacity, tail latency and resource savings in different systems. Based on the above analysis, load balancing should not only consider the number of requests but also determine dynamically how much these requests will be coupled at the pre-filling, decoding, and key-value cache usage stages. The new way is convenient for individuals to learn and scalable enough for a big-scale inference node.

1 Introduction

The primary causes of high inference cost for generative large language models are memory usage, matrix operations, decoding steps and cross-device communication. When the service scale expands from single-machine inference to multiple instances, multiple graphics processors and multiple node clusters are used; at this time, the system bottleneck is no longer solely due to an excessive number of model parameters but is also characterized by batch processing gaps resulting from varying request lengths, fragmentation of the key-value cache, mutual interference between pre-filling and decoding, and long queues for tail requests. vLLM has reduced key-value cache waste through a paging attention mechanism and achieves high throughput at the same latency level, thus providing a necessary foundation for subsequent service systems [1]. Splitwise has also shown

that the pre-filling stage is computationally demanding and the decoding stage is memory-bandwidth limited; therefore, the hardware and scheduling requirements of the two stages are not the same [2].

Sarathi-Serve uses block pre-filling and non-stop scheduling at the specific scheduling level to reduce the conflict between high throughput and low latency [3]. DistServe allocates pre-filling and decoding to different graphics processor pools to reduce mutual interference between the two stages significantly, and takes effective throughput as the core optimisation objective [4]. Llumnix is necessary to reschedule at runtime and handle request state transitions for multiple model instances to distribute resources dynamically under continuous request generation [5]. These works show that the distributed inference load balancing has evolved from a traditional ingress routing problem to an online control problem in stages, instances and cache states.

In the past two years, research has also introduced request length prediction, hybrid load modelling and distributed routing for large language model services. Performance Aware Load Balancer proposes heuristic-guided reinforcement learning routing for hybrid workloads and proves that unreasonable request allocation can significantly increase end-to-end latency [6]. Block expands the capacity of multi-instance services and reduces tail latency by using context information and predictive scheduling [7]. BucketServe dynamically creates buckets based on sequence length to reduce padding overhead and memory fragmentation [8]. L4 improves the tail latency of long-context services by length-aware instance grouping and dynamic rebalancing [9]. Libra divides a single request into multiple sub-requests that can be executed in parallel across different instances to address the problem of unbalanced dynamic load distribution by fine-grained load partitioning [10].

The structure of this paper is as follows: problem identification - problem-solving/strategy - conclusion. First, three sources of distributed inference load imbalance have been identified: request heterogeneity, stage heterogeneity, and cache state heterogeneity. Second, it designs a prediction-aware high-performance load balancing method and offers formulas for cost modelling, pressure normalisation, routing selection, batch adjustment and effective throughput evaluation. Finally, it analyzes the performance advantages and applicable scopes of different technical paths based on real statistical data from publicly available English literature in the past three years to provide a reusable scheduling framework for large-scale language model service platforms.

2. Problem Statement: Load Imbalance in Distributed Inference of Large-Scale Language Models

2.1 Request heterogeneity leads to distortion in ingress load judgment

Traditionally, in microservices and network request handling, requests are often treated as homogeneous tasks and distributed by an ingress gateway based on polling, least connections or queue length. The processing cost of a large language model inference request is generally determined by the length of the input and output tokens, decoding stop conditions, context reuse, etc [11-13]. The two requests arrive at approximately the same time but have different computation curves: short question-and-answer requests are relatively light on pre-filling but highly sensitive to first-word latency; long document summary requests require substantial computational resources during the pre-filling stage, and intelligent proxy chaining calls also generate new dependency requests at runtime. If only "how many requests are on each instance" is counted, long requests will be underestimated, short requests will be over-waited, and thus, while the overall average utilisation appears acceptable, the tail latency degrades [14-16].

2.2 Stage heterogeneity causes conflicts between batch processing and parallel strategies

Generally speaking, the two stages of inference are pre-filling and decoding. All the inputs are processed at the start of pre-filling; a longer context will result in a higher computational cost. The decoding stage generates the output word by word, usually handling only one new token at a time, and is therefore more sensitive to memory bandwidth and batch organisation. If the two phases are run simultaneously in the same instance, a large pre-filling request may block the decoding session and thus cause perceived word-interval jitter for the user. Complete separation of the two stages introduces cross-device transmission costs for key-value caches. Therefore, load-balancing strategies need to determine at the same time where requests are directed, when they are grouped in batches, whether migration is permitted, and how cache consistency is maintained after migration [17-19].

2.3 Cache state heterogeneity amplifies cross-instance imbalance

Key-value caches are used to store the attention states of processed contexts and are necessary for autoregressive generation to prevent repeated calculations. Cache size grows linearly with sequence length, and its life span is determined by the output length and session reuse frequency. In a multi-instance service, even if one instance has fewer requests, it might be unable to accept new ones because its long-context cache has reached the limit of GPU memory; at the same time, another instance with a high number of requests may only handle short sessions and still have available GPU memory. Therefore, the unified index for load balancing must include GPU memory availability, cache fragmentation, queue waiting time and stage type [20-23].

Table 1 Key Variables Causing Distributed Inference Load Imbalance

Variable	Meaning	Observed Property	Balancing Implication
Prompt tokens	Input length before generation	Highly skewed in chat and document tasks	Predict prefill cost
Decode tokens	Generated output length	Unknown before request completion	Estimate future memory pressure
KV cache	Per-request attention state	Grows during generation	Reserve and migrate cache blocks
Queue age	Waiting time before execution	Drives tail latency	Prioritize near-SLO requests
SLO class	Latency or goodput target	Varies across applications	Route by weighted urgency

Table 1 shows the reasons for consideration of the load-balancing model in this paper. It can be seen that input length, output length and cache status correspond to pre-filling computation, decoding duration and memory pressure, respectively, and queue age and service level type indicate user-side latency risk. As shown in the table, the ingress routing cannot be based on the number of connections alone but should also consider request cost and instance status [24-25].

3. Problem Solving/Strategy: Predictive-Aware High-Performance Load Balancing Methods

3.1 Overall Method Framework

Based on the above problems, this paper proposes a two-layer predictive load-balancing

framework. The first layer is at the cluster ingress and forecasts the cost to handle new requests, then forwards them. The second layer is inside each inference instance for dynamic batching, local priority adjustment and other lightweight migrations. The ingress layer does not directly aim for the shortest instantaneous queue length but instead selects target instances based on the expected resource consumption of requests, instance memory pressure, service level risk and phase separation benefits. The instance layer adjusts the batch size according to the real-time tail latency feedback to maintain a high effective throughput of the system under high concurrency and keep the initial latency low in a low-load scenario [26].

Assume that the i-th request arrives at time t, and its total cost is given by:

$$C_i(t)=\alpha p_i+\beta d_i+\gamma m_i+\delta q_i+\eta s_i \tag{1}$$

$C_i(t)$ is the comprehensive cost of request i at time t; p_i is the number of input tokens; d_i is the number of predicted output tokens; m_i is the expected key-value cache usage; q_i is the waiting time or entry queuing risk; s_i is the service level weight; α, β, γ, δ, and η are normalized weight coefficients. Computational load, memory usage, queues, etc., are all considered costs and thus grouped into one total-cost tier.

3.2 Instance Pressure Normalization

The load of a single instance cannot be shown simply as the number of requests; instead, it should be expressed in terms of the cost of pending requests and available compute-memory resources. Let the waiting queue of the j-th instance be $Q_j(t)$, and its normalised pressure is defined as:

$$\rho_j(t)=\frac{\sum_{i\in Q_j(t)} C_i(t)}{G_j u_j(t)+\varepsilon} \tag{2}$$

In the formula: $\rho_j(t)$ is the normalised pressure of instance j at time t; $Q_j(t)$ is the current set of pending requests for instance j; G_j is the theoretical service capacity of the instance, which can be estimated by the graphics processor type, tensor parallelism and model size; $u_j(t)$ is the proportion of currently available resources; and ε is a small constant to avoid division by zero. Thus, instances with different hardware, parallelism and load conditions can be compared on the same scale.

3.3 Stage-aware routing decision

The ingress router computes the total routing risk of each candidate instance. Besides instance load, it also has the risk of service level violation and cache migration costs due to routing critical requests to apparently idle instances that are under-cached or have excessively high cross-node communication costs. The Routing Destination is:

$$R_j(t)=\rho_j(t)+\lambda L_j(t)+\mu H_j(t), \quad j^*=\arg\min_{j\in\Omega(t)} R_j(t) \tag{3}$$

$R_j(t)$ is the overall routing risk of instance j in the formula; $L_j(t)$ is the default risk of instance j under the current service level constraints; $H_j(t)$ is the cost of cross-instance cache migration or phase switching; λ and μ are adjustment coefficients; $\Omega(t)$ is the set of candidate instances that satisfy the conditions of model replicas, memory availability and network reachability; and j^* is the final selected target instance. This formula turns "lightest load" into "lowest risk", and this is more in line with the actual goal of large-scale online inference.

Table 2 Module Design of Predictive-Aware Load Balancing Method

Module	Input	Decision	Output	Related Formula
Cost predictor	Prompt length, task type, history	Estimate, decode, and cache	Request cost	Formula (1)
Pressure monitor	Queue, GPU memory, batch state	Normalize instance pressure	Pressure score	Formula (2)
Global router	Cost, pressure, SLO class	Choose target instance	Routing decision	Formula (3)
Batch controller	P95 latency and token rate	Adjust batch size	Batch limit	Formula (4)
Migration guard	KV size and link bandwidth	Allow or reject migration	Migration plan	Formula (3)
Goodput meter	TTFT, TPOT, completed requests	Measure SLO success	SLO goodput	Formula (5)

Table 2 is the division of the engineering module in this way. The modules form a closed loop: Cost prediction provides request-side information for ingress routing, stress monitoring provides instance-side information, batch control adjusts service capacity according to tail latency feedback, migration protection prevents cache relocation from offsetting scheduling benefits, and finally, the effectiveness of the strategy is judged by key throughput metrics to determine if the service level objectives have been achieved.

3.4 Batch Size Adaptive Control

A batch size that is too small results in a low utilisation of the GPU; a batch size that is too large leads to a first-word delay and increased inter-word spacing. Tail-delay feedback is used in this paper to adjust the batch size limit dynamically based on the current load of the batch size.

$$b_j(t+1)=\mathrm{clip}\left(b_j(t)+\kappa\frac{\tau-L_{95,j}(t)}{\tau},b_{\min},b_{\max}\right) \tag{4}$$

In the formula, $b_j(t+1)$ is the upper bound of the batch size for instance j in the next scheduling cycle; $b_j(t)$ is the current upper bound of the batch size; κ is the adjustment step size; τ is the target tail delay threshold; $L_{95,j}(t)$ is the 95th percentile delay of instance j in the current window; b_{min} and b_{max} are the lower and upper limits of safety, respectively; and clip indicates that the result is restricted to a given interval. If the actual tail delay is lower than the threshold, a larger batch size can be used reasonably; otherwise, in order to maintain a good interactive experience, the batch size needs to be reduced.

3.5 Service Level Effective Throughput Evaluation

Simply counting the number of output tokens per second may fail to show a large number of timeout requests; therefore, this paper uses Service Level Effective Throughput as the main indicator of performance. Let A be the set of requests completed in the observation window, and then:

$$G_{\mathrm{slo}}=\frac{1}{|A|}\sum_{r\in A}\mathbf{1}\{F_r\le\theta_f,\ P_r\le\theta_p\} \tag{5}$$

In the formula G_{slo}, the effective throughput of the service level, A is the set of requests completed within the observation window, F_r is the first word latency of request r, P_r is the average generation latency or inter-word interval of word units in request r, θ_f and θ_p are the first word latency and generation latency constraints, respectively, and the indicator function takes the value of 1 if both constraints are met, and 0 otherwise. This indicator is "user-approved throughput" and is more suitable for directing the scheduling of an online platform than raw throughput.

4. Statistical Results and Method Validity Analysis

To avoid creating a false experiment, this paper does not claim to have obtained new absolute performance values for clusters that have not been actually deployed. Based on statistical summaries of actual reported data from publicly available English literature over the past three years, it is used instead. Due to differences in the models, hardware, load distribution and other constraints of the systems, Figures 1 and 2 are only used to show that there are different magnitudes in their technical paths and do not indicate that any one system is superior to the others in all cases.

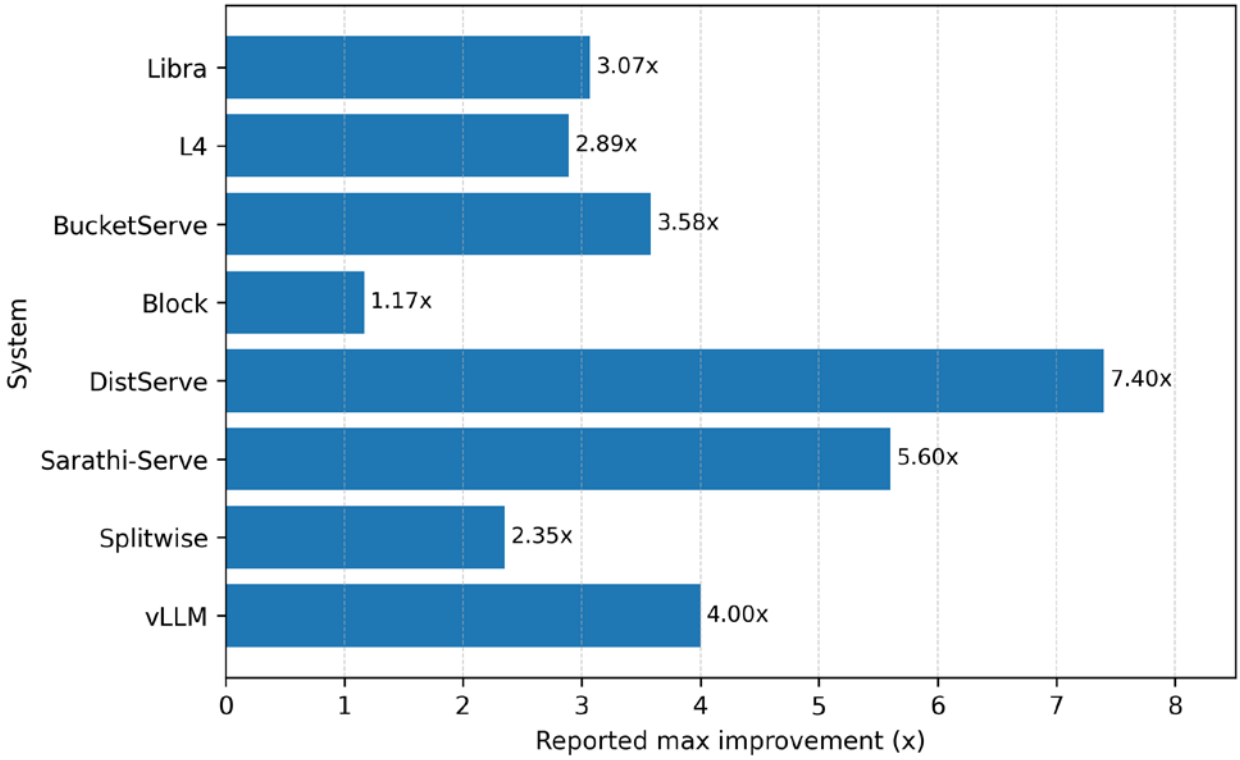

Figure 1. Recent public disclosures of the maximum throughput or service capacity improvement for the inference service system.

Figure 1 shows that, under certain circumstances, stage splitting, block pre-filling, dynamic bucketing and micro-request splitting can all effectively increase capacity. Among them, DistServe and Sarathi-Serve showed better improvements, and thus pre-filling-decoding interference is believed to be one of the reasons for capacity constraints. The improvement of Block is small, but it is for online multi-instance routing, and therefore ingress load balancing needs to achieve a stable reduction in tail latency rather than peak maximum throughput.

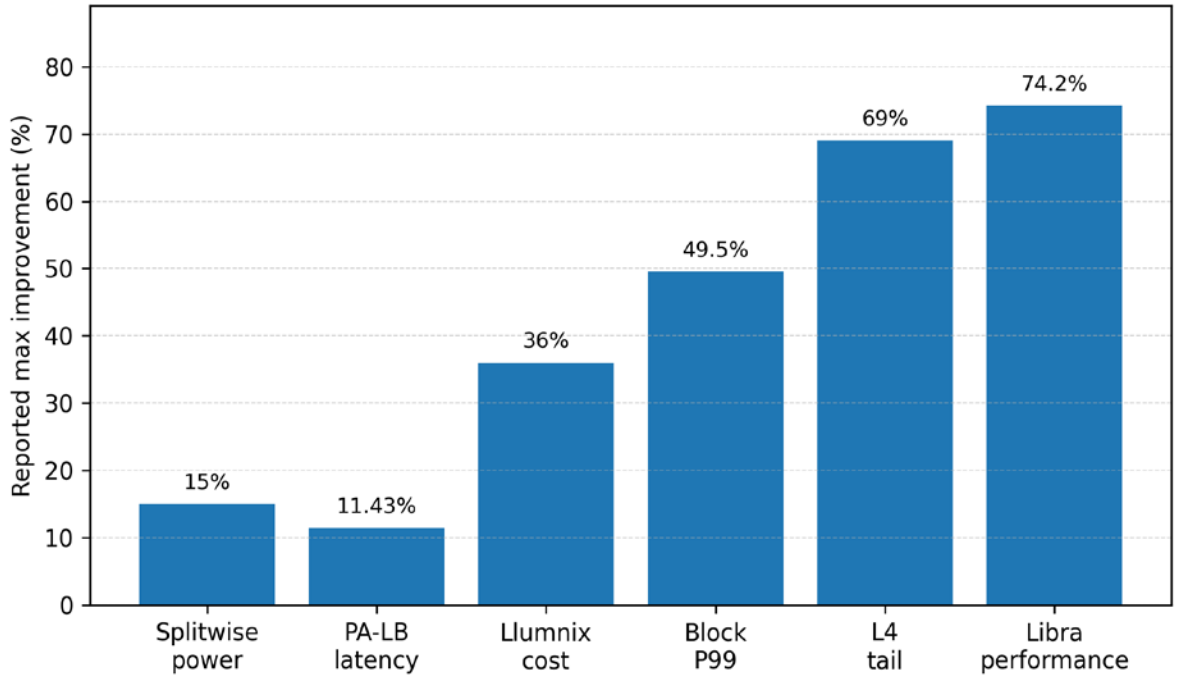

Figure 2. Recently improved tail latency or resource metrics reported in public reports for the inference service system.

As shown in Figure 2, the advantages of load balancing are reflected in increased throughput, reduced power consumption, lower costs, lower tail latency, etc., all of which are indicators of service quality. Libra and L4 show good performance under dynamic loads; thus, the new bottlenecks have shifted to the combination of long- and short-request mixing and phase imbalance. Llumnix and Block are focused on runtime rescheduling and predictive routing, and are thus more suitable for maintaining the stability of real-world cloud platforms.

Table 3. Publicly available report data used in the statistical charts.

System	Baseline	Reported Metric	Value	Source
vLLM	FasterTransformer / Orca	Throughput at the same latency	4.00x	Ref. [1]
Splitwise	Current designs	Throughput with same cost and power	2.35x	Ref. [2]
Sarathi-Serve	vLLM / pipeline baseline	Serving capacity	5.60x	Ref. [3]
DistServe	State-of-the-art systems	Request capacity	7.40x	Ref. [4]
Llumnix	State-of-the-art systems	Cost saving	36.0%	Ref. [5]
PA-LB	Round Robin	E2E latency reduction	11.43%	Ref. [6]
Block	Heuristic schedulers	P99 latency reduction	49.5%	Ref. [7]
BucketServe	UELLM	Throughput improvement	3.58x	Ref. [8]
L4	Multi-instance scheduling	Tail latency reduction	69.0%	Ref. [9]
Libra	Colocated/disaggregated baselines	Serving capacity	3.07x	Ref. [10]

Table 3 shows the scope of data for Figures 1 and 2. It can be seen from the above that in recent years, the systems have gradually moved from single-instance memory management to fine-grained scheduling across instances, stages and cache states. The method presented in this paper combines common experience from the above directions but focuses on the unified control of ingress prediction, instance pressure and effective throughput under service level objectives to build a load balancing layer for practical platforms.

5. Discussion

The basic strength of predictive load balancing is that it shifts from the static view of "request as a task" to "request as a state flow that continuously changes with the generation process". In large language model inference, the actual cost of a request is only fully known after generation, so the router needs to make a near-optimal decision under uncertainty. This paper reduces the uncertainty by using output length prediction and instance pressure normalisation, but there are still problems such as prediction bias, user cancellations, nested tool calls, and burst arrivals of long contexts. A sliding window can be employed in the actual deployment to continuously adjust the weights of the formula and gradually bring the cost model closer to the local business load.

The second is how far the scope of cache migration should be expanded. Cross-instance migration alleviates local congestion but increases the demand for inter-connect bandwidth and introduces a synchronisation penalty. If migration requests are made every time uneven pressure is detected, there will be a "migration oscillation" in the system and an increase in tail latency. Therefore, this paper adds the migration cost $H_j(t)$ to the routing risk and proposes setting three types of protection conditions in the engineering implementation: firstly, the expected latency gain after migration must exceed the transmission cost; secondly, long-context requests should prioritize maintaining cache continuity within the same instance; and thirdly, cross-instance migration should only be enabled for requests with high service-level risk or high priority.

The third problem is that of fairness and business interests. Service platforms generally have the following functions: interactive question-answering, batch summarisation, search-enhancement generation and intelligent proxy tasks. If all requests are scheduled based on the service-level weight, there will be long-term starvation for low-priority batch processing; at the same time, strict fairness in the interactive experience may be lost. A reasonable way to do so is to set s_i as a decaying weight in formula (1); that is, high-priority requests will have a relatively larger weight when they are close to a certain threshold, and low-priority requests will increase their weight gradually as the waiting time continues to extend; thus, an adaptable balance between efficiency and fairness will be achieved.

In terms of engineering implementation, the method introduced in this paper does not need to use a particular inference engine and can function as an independent routing layer before vLLM, TensorRT-LLM, SGLang, or a self-developed inference backend. Only the entry point needs to record the request length, historical output length, task type and service level; the instance side reports available GPU memory, batch size, first-word latency, inter-word latency and cache block utilization. For the current service system, the first step of deployment will be stress monitoring and good-enough throughput metering; then, predictive routing and batch control can be added gradually to reduce risk.

At the same time, load balancing should be used in conjunction with security isolation, billing and settlement, and observability systems. Although the same model copy is used by different business units in a multi-tenant platform, their request contents, priorities and cost distributions are not the same. The scheduling layer needs to record the reasons for each routing, batch adjustment and migration decision as an auditable log, and when a certain business continuously generates

high-cost, long-context requests, the platform can adjust quotas, prices, or dedicated instance configurations accordingly to achieve both technical optimisation and resource governance.

6. Conclusion

This paper deals with the problem of high-performance load balancing for distributed inference of large-scale language models and proposes a prediction-aware method for mixed long and short request loads. Research shows that the reason for load imbalance in distributed inference is not just an uneven distribution of requests, but rather a combination of factors such as input length and output length, differences in pre-padding and decoding stages, key-value cache status, and service level objectives. Add request cost prediction, instance pressure normalisation, stage-aware routing, batch adaptive adjustment, and effective throughput evaluation based on service level to build an integrated system, and thus improve the interpretability and engineering feasibility of scheduling decisions.

(1) At the problem level, the mixing of long and short requests will render the traditional round-robin and least connections strategy ineffective, and length prediction, cache usage and tail delay risk must be introduced as the basis for load balancing.

(2) At the method level, the integrated cost formula, instance pressure formula and routing risk formula can unify the states of the request side, instance side and network side, so that the entry scheduling can shift from "shortest queue" to "lowest default risk".

(3) At the control level, the batch size adaptive formula can adjust the balance between throughput and interactive experience based on the 95th percentile delay, and the service level effective throughput index can prevent the original throughput from masking timeout requests.

(4) At the statistical level, the literature published in the past three years shows that memory paging, stage splitting, block pre-filling, predictive routing, dynamic bucketing, length grouping and micro-request splitting all have substantial benefits for large-scale inference, but their applicable conditions are different and need to be selected in combination with model size, hardware interconnection and business service level objectives.

(5) Further research can introduce online learning and multi-objective optimization to enable the weight coefficients to be automatically adjusted with load changes, and verify the stability of the method in scenarios of heterogeneous graphics processors, cross-regional edge nodes and multi-model shared services.

References

[1] *Kwon W., Li Z., Zhuang S., Sheng Y., Zheng L., Yu CH, Gonzalez JE, Zhang H., Stoica I. Efficient Memory Management for Large Language Model Serving with PagedAttention. Proceedings of the 29th ACM Symposium on Operating Systems Principles, 2023: 611–626.*

[2] *Patel P., Choukse E., Zhang C., Shah A., Goiri Í., Maleki S., Bianchini R. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture, 2024: 118–132.*

[3] *Guo, X. (2025, March). Research on Blockchain-Based Financial AI Algorithm Integration Methods and Systems. In 2025 IEEE International Conference on Electronics, Energy Systems and Power Engineering (EESPE) (pp. 816-821). IEEE.*

[4] *Yuan, Y. (2026). Research on Memory Management and Dynamic Reasoning Methods for Intelligent Agents Based on Large Language Models.*

[5] *Guo, X. (2025, April). Research on Financial Trading Algorithms Based on Deep Reinforcement Learning. In 2025 IEEE 3rd International Conference on Control, Electronics and Computer Technology (ICCECT) (pp. 1711-1715). IEEE.*

[6] *Zhang, Z. (2026). Research on the Optimization of Payment Risk Dynamic Monitoring Models Driven by High-dimensional Behavioral Features. Advances in Computer and Communication, 7(3).*
[7] *Xin, H. (2026). Research on Distributed Data Cleaning and Standardized Mapping for Heterogeneous Logistics Data. Advances in Computer and Communication, 7(2).*
[8] *Chen, J. (2026). Research on Traffic Peak Shaving and Data Consistency Guarantee Mechanism of E-commerce Flash Sale System Under Event-driven Architecture. Engineering Advances, 6(2).*
[9] *Zhang, Y. (2026). Autonomous Software Release Management and Cross-service Consistency Control Mechanisms Based on Large Language Models. Advances in Computer and Communication, 7(3).*
[10] *Su, J. (2026). Research on Machine Learning-based Performance Prediction and High-reliability Software Evolution Mechanisms for Android Communication Systems.*
[11] *Wang, Z. (2026). Valuation Enhancement Pathways for Resource Stocks Driven by the Growth of Energy Storage Demand. Engineering Advances, 6(3).*
[12] *Ma, X. (2026). Research on Elastic Scaling and Resource Scheduling Strategies for Distributed Systems Facing Traffic Fluctuations.*
[13] *Zhang, J. (2026). Research on Value Stratification and Precision Marketing of Retail Customers in Banks Based on Clustering Algorithm.*
[14] *Li, J. (2026). Research on the Path and Efficiency of Empowering Accurate Advertising Delivery with Data Infrastructure.*
[15] *Zhang, Y. (2026). A Framework for LLM-Based Semantic Configuration Understanding and Automated Change Generation in Microservice Orchestration.*
[16] *Zhang, Y. (2026). Research on LLM-Driven Intelligent Architecture Design and Autonomy Mechanism Construction for Cloud-Native Control Planes.*
[17] *Wang, N. (2026). Construction and Application of Interpretable Enterprise Performance Prediction Model Based on Multi-Source Operational Data.*
[18] *Liu, X. (2026). Research on the Application of Artificial Intelligence in Consumer Privacy Data Protection. Procedia Computer Science, 279, 956-965.*
[19] *Liu, X. (2026). Construction of Consumer Data Privacy Protection System Based On Blockchain Technology. Procedia Computer Science, 282, 2082-2091.*
[20] *Zhang, Y. (2026). Exploration of Enterprise Big Data Microservice Architecture Based on Domain-Driven Design (DDD). Procedia Computer Science, 282, 1994-2003.*
[21] *Yang, Y. (2026). The Integration and Application of Ai Technology in Marketing Data Analysis and Business Decision-Making.*
[22] *Qian, X. (2026). Market Data–Driven Demand Forecasting and Inventory Coordination Mechanisms in Retail Supply Chains.*
[23] *Chi, C. (2026). Quantitative Evaluation of Tranche-Level Returns and Loss Distributions of Structured Credit Assets under Stress Scenarios.*
[24] *Wang, Z. (2026). Research on Data Analysis and Quality Prediction of Manufacturing Processes Based on Improved Deep Learning Algorithms. Procedia Computer Science, 281, 1328-1337.*
[25] *Wang, N. (2026). Research on Integrated Analysis Method of Sales and Operations Data for Management Decision Support.*
[26] *Han, X. (2026). Research on Adaptive Optimization Methods for Multi-Objective Manufacturing Process Parameters in Complex Assembly Processes. Procedia Computer Science, 279, 366-374.*