Comparing Load Balancing Algorithms in AI Inference Services

Yibo Li
Singapore Institute of Management, Singapore

Abstract: The study focuses on load balancing in AI inference systems. The rapid growth of Artificial Intelligence (AI), in large language models (LLMs), has significantly impacted information systems. Classical algorithms (Round Robin, Least Outstanding Requests (LOR), and Power of Two Choices (P2C)) and AI-specific algorithms (Token-Aware and Prefix-Cache routing) are compared to determine their performance by applying a simulation-based approach, SimPy, under 25%, 60%, and 90% load conditions using response time, tail latency, throughput, and server utilization. Under low load conditions, all algorithms displayed similar results, while under moderate and high load conditions, AI-specific algorithms established remarkable results than classical algorithms.

Keywords: Load Balancing; AI Inference; LLMs; Token-Aware; Prefix-Cache

1. Introduction

The rapid growth of Artificial Intelligence (AI), specifically the large language models (LLMs), has significantly impacted information systems [1]. They have been used to automate repetitive tasks, where AI-powered chatbots offer instant support to customers, whereas LLMs analyzes large datasets to detect patterns and tendencies that are important in decision-making processes [2]. Chatbots leverage natural language processing (NLP) and machine learning (ML) algorithms to understand and reply to user questions conversationally [3]. However, this poses serious computational challenges on AI inference services where all requests pass through distributed Graphics Processing Unit (GPU-based) structures [4]. One key challenge is load balancing, which improves application performance by increasing response time efficiency and decreasing network latency.

Traditional load balancing algorithms, including Round Robin, Least Outstanding Requests (LOR), and Power of Two Choices (P2C), were developed to enhance network performance and resource utilization [5,6]. They dynamically adjust to traffic distributions centered on real-time networks and ensure requests are similar and evenly distributed across servers, preventing overload on any single instance while improving consistency and response times. However, this differs from LLM systems where workloads are heterogeneous [7]. Requests can involve tens to hundreds of billions of parameters, depending on use cases, and may include long inputs that increase computational costs. Retrieval-augmented generation (RAG) pipelines require substantial GPU memory and processing time based on input and output token dimensions, making traditional resource allocation inefficient [8].

Classical load balancers, therefore, present limitations that have led to AI-specific routing approaches. Techniques such as Token-Aware load balancing and Prefix-Cache routing help reduce computational costs and repeated memory processing associated with LLM workloads [9]. Prefix caching accelerates multi-turn interactions and requests with shared prefixes by routing them to GPU nodes where relevant input data has already been cached [10]. This improves system performance by reducing access time for frequently used inputs and enabling faster processing speeds. However, a gap remains because there is no clear comparison between classical and AI-specific algorithms under identical conditions. Existing studies do not effectively evaluate these methods, which may lead to unrealistic conclusions.

This research addresses this gap by incorporating Round Robin, LOR, P2C, Token-Aware, and Prefix-Cache routing within a controlled virtual environment. It adopts a positivist philosophy where knowledge is based on observable and measurable evidence, and realism is objective. A discrete-event simulation framework using SimPy models an AI inference cluster based on these routing algorithms [11]. Key performance metrics, including response time, tail latency, throughput, and server utilization, are applied to evaluate system

performance under varying load conditions. The findings provide useful insights for identifying effective load balancing approaches and improving AI management inference workloads. These results support a better understanding of algorithm behavior and contribute to more consistent evaluation of performance across heterogeneous workloads in controlled experimental settings without introducing new assumptions or external factors.

2. Literature Review

2.1 Load Balancing in Distributed Systems: Context and Evolution

Load balancing or distribution work across servers is one of the core challenges in distributed computing. To serve traffic across multiple servers is essential to maintaining system performance in distributed networks. The main aim is to direct incoming requests to ensure that no single machine is overloaded and others are underutilised. The most appropriate strategy is reduced to two key factors: the character of the workload, and what the router has known about the system when it chooses to send each of the requests.

Historically, the study of load balancing has been based on web servers and general-purpose cloud infrastructure. In this regard, the requests may be considered as roughly equivalent due to the fact that the web responses can take milliseconds to process and consume resources that are of the same type. The main challenge, as Kumar & Kumar [12] put it, is efficient resource utilization and quality of service under different demand. The AI inference pipeline is however different. To this it adds another layer of obstacles and a poor routing choice does not merely slack things down, it squanders the resources of the GPU which is far costlier.

The deployment of large language models (LLM) in the real world generates an entirely different form of computational load. One inference query can use gigabytes of memory and hold processors tied up several seconds. The price is also non-linear with prompt length, thus the consumption of resources is difficult to forecast. The cost also scales non-linearly with prompt length, so resource usage is hard to predict. Classical load balancing algorithms were not designed for these conditions. Their limitations under LLM workloads are by now well-documented [1]. The sections below cover the classical algorithms that form the comparison baseline, explore the structural differences that make LLM inference workloads unusual, survey the AI-specific routing approaches that have emerged in response, and identify the gap this study addresses.

2.2 Classical Load Balancing Algorithms

There are three algorithms that are currently used most frequently: Round Robin, Least Outstanding Requests, and Power of Two Choices. Each algorithm trades off how much it knows about server state against how much network traffic it needs to keep up that knowledge.

Round Robin allocates requests sequentially across all the available servers in a constant rotation. It does not consider any server state. It operates completely blind to the actual state of the servers, and for this it requires zero cross-node communication and it also executes almost instantly. The Round Robin algorithm works well when tasks are similar to each other since the request to process needs almost the same resources and it tends to behave more predictably and consistently. As Al Nuaimi et al. [5] note, it is a 'simple and deterministic' approach for uniform tasks. Under mixed workloads, however, ignoring server state leads to obvious bottlenecks. A server who is busy doing the computationally heavy task will receive more requests while a server that is quickly wrapped up with a lighter task ends up sitting doing nothing until the next round of assignments.

Least Outstanding Requests (LOR) tries to fix the round robin’s problem by routing traffic to a server which currently holds the fewest active connections. As round robin completely ignores what is actually happening to a server, LOR pays attention to the real time server conditions. Because of that, Amazon chose it for their application load balancer [13]. But using LOR for AI inference exposes a critical flaw: it counts connections rather than actual computation. A server busy with handling a single heavy request like processing 2000 token prompts might seem less busy compared to a server handling three quick simple tasks with 100 token prompts. Even though the first server is doing more computationally expensive tasks which will be more time consuming. LOR solves Round Robin's problem by maintaining a server state but also creates a different blind spot: it knows

how many requests a server holds, but not what they cost.

Power of Two Choices (P2C), introduced by Mitzenmacher [6], takes a probabilistic approach. P2C maintains a global server state to avoid some of the coordination overhead. Rather than searching all the servers to find the least busy server, the router picks two servers randomly. Then it distributes requests to whichever one has more breathing room. Mitzenmacher's [6] theoretically showed that selecting the least loaded server from two random choices reduces the maximum load from O(log n / log log n), observed under purely random assignment, to O(log log n). In situations where there are simultaneous requests, keeping all the servers in sync itself slows things down. In that scenario, P2C offers a middle ground. But it also runs into a problem related to LOR, it counts requests rather than considering how heavy a request is. So it still has issues and also creates an improvement scope.

All three algorithms remain widely adopted and widely used everywhere before and after the AI era. Two of the most widely used open-source load balancers - NGINX and HAProxy adopted Round Robin and LOR as their default strategy, [5]. Their continued prevalence in production systems makes them the natural baseline against which newer, AI-aware approaches should be evaluated. However, as the next section explains, the workload characteristics of LLM inference are different enough from general web traffic that the assumptions underlying these algorithms no longer hold.

2.3 The Unique Characteristics of LLM Inference Workloads

The assumption underlying all three classical algorithms, that requests are roughly equivalent units of work, breaks down comprehensively in large language model inference.

Transformer-based models process each request in two computationally distinct phases. During the prefill phase, the model ingests the entire input prompt and generates the key-value (KV) cache encoding the attention state for all input tokens. This phase is compute-intensive and its cost scales directly with prompt length. The decode phase then generates output tokens one at a time until a stopping condition is met; this phase is memory-bandwidth-bound, and its duration depends on the output length, which is not known at routing time. A 100-token prompt with a 10-token response is a fundamentally different computational job from a 1,000-token prompt with a 500-token response, yet both look identical to Round Robin, LOR, or P2C.

Yu et al. [14] addressed this heterogeneity at the serving level in Orca, which introduced iteration-level scheduling. By interleaving multiple requests at the granularity of a single generation step rather than processing each request end-to-end before accepting the next, Orca improved GPU utilisation and reduced tail latency in single-node deployments. Kwon et al. [4] advanced this in vLLM with PagedAttention, mapping KV cache storage into non-contiguous memory pages analogous to virtual memory in an operating system. This solved the problem of wasted GPU memory that older systems struggled with. It used to reserve adjacent GPU memory per request even when it was not fully utilized. Which makes it much more convenient to work with numerous requests simultaneously. It implies that to load balance, routing choices cannot simply examine how busy a server processor is; accessible memory must be taken into account as well. Server may appear free and willing to accept a new task but when it is running out of memory, it may be forced to drop off task it was already caching.

Zhong et al. [9] took the prefill-decode asymmetry further in DistServe by disaggregating the two phases across separate GPU nodes entirely. Physically separating prefill and decode reduces interference between the compute-intensive and memory-bandwidth-bound stages, improving overall goodput under mixed workloads. This disaggregated architecture has since been incorporated into vLLM and SGLang and represents the current state of the art in production-grade inference serving. From a load balancing perspective, disaggregation also changes what a routing algorithm must optimize for: decisions can no longer treat the serving of a single request as a single atomic job, since the prefill and decode stages may execute on different nodes with different resource profiles.

2.4 AI-Specific Load Balancing Approaches

Motivated by the limitations of classical algorithms in these contexts, recent work has begun to design routing strategies explicitly aware of LLM workload characteristics. Two lines of work are particularly relevant to this study.

Token-Aware load balancing, introduced in the Preble system by Srivatsa et al. [15], routes on token length rather than request count. Since the computational cost of an inference request is primarily a function of total tokens processed-input tokens during prefill and output tokens during decode-distributing by token count gives a much closer approximation of actual GPU load than distributing by request volume. Srivatsa et al. [15] found that routing decisions based on token count significantly cut down on worst-case response times. In comparison to LOR when many requests are coming in at once, routing decisions performed well based on token count. The difference was even more noticeable when request lengths varied wildly. This type of skewed distribution is in fact very normal in real-world systems, with a few unusually long requests silently floating to the top of the serving capacity.

Another approach introduced by SGLang by Zheng et al. [10] called Prefix-Cache routing. Many LLM requests start with the same opening context, usually a system prompt that appears at the beginning of every conversation in a given application. When a server has already processed that prompt before, it saves the result in memory for later use. SGLang's cache-aware router directs each incoming request to the server most likely to already have that saved data. One of the downsides is coordination overhead: the router has to keep track of what every server has stored in memory, and this overhead grows with cluster size and the diversity of prefix patterns in the workload.

Sun et al. [16] tried a different approach with Llumnix, which can actually support live migration of pending requests between serving instances when load conditions change. Most routing systems make a decision once and stick with it. Llumnix can re-route a request mid-processing if a server's queue grows unexpectedly, providing a more adaptive form of scheduling. This is particularly handy when the requests are received simultaneously. Miao et al. [1], also mention that the dilemma of determining the most appropriate means to send requests to more than one server remains open to date. There is no one routing method that has demonstrated itself to be the best in all situations.

2.5 Theoretical Foundations and Research Gap

Queuing theory provides us with an understanding of how to interpret the findings of this research. The foundation that Kleinrock [17] developed demonstrates that the three things are interrelated, namely, the rate of arrival of the requests, the rate of their processing, and the waiting period. For building a three-node inference cluster, M/M/k model is used as a theoretical benchmark for the starting point. It is based on the assumption that the requests are randomly received, the time of each request is random and there are k servers to process all the requests concurrently. From those assumptions, the model tells you something genuinely useful: how average response time and server load change as the system gets busier.

The M/G/1 model relaxes an assumption. It no longer requires that every request takes almost the same amount of time to finish. This makes it a good fit for LLM inference because in the LLM world, some requests are much longer and heavier than others.

None of the models discussed above represent the prefix cache dynamics. This one plays a vital role in understanding prefix-aware routing. That is why this study takes a simulation approach alongside the analytical baseline rather than relying on theory alone.

Miao et al. [1] and Li et al. [7] both point out one similar thing. There is an open need for fair, controlled comparisons of different types of load balancing algorithms. Most research papers only test a new algorithm against one or two others, also on a specific workload making general conclusions difficult to draw. No published work has yet compared the full range of classical and AI-specific routing algorithms under the same conditions. This research is attempting to fill in that gap by building a simulation environment in which the routing algorithm is the only variable in it, rendering it. can put all the five approaches in a level playing field.

3. Methodology

3.1 Research Philosophy and Overall Approach

This paper assumes positivist approach: the performance of systems as something real and measurable. It can also be tested by way of controlled experiments. The research question however is: what among five load balancing algorithms is the best when compared with key performance metrics in a simulated environment of an AI inference cluster. Everything about the

research approach is to build a system which will provide all the questions in numbers and data.

The use of simulation instead of real hardware has two reasons in this study. Academic research is costly to access a multinode GPU cluster at production inference scale. Manufacture of actual equipment also brings in numerous variables that are difficult to control. Things like driver versions, thermal throttling, and network inconsistencies are difficult to isolate. And those things are able to cloud the findings. The routing algorithm is the only independent variable in adopting Simulation. It is certain that any variation in performance can be attributed to the algorithm itself and not the environment it executes in. This approach is well accepted in distributed systems research, and used as a standard method for a long time [18]. All this is scheduled prior to the commencement of the tests.

Everything in this study is planned before testing begins. The experimental plan, metrics, and analysis procedure are all defined before any results are collected. This keeps things honest and ensures no algorithm gets any unfair advantage. The experimental conditions such as load levels, token length distributions, number of replications, are all based on existing research rather than being not tuned to advantage any particular algorithm.

3.2 Simulation Tool: SimPy

The simulation is written in Python using a library called SimPy. It's a discrete-event simulation framework [11] widely used on different research papers. SimPy makes it easy to model things like requests arriving, server processing them and queues building up without complicating things and codes. It integrates directly with other popular python tools such as NumPy, SciPy, and Pandas, which makes handling and analysing the results straightforward.

SimPy was chosen over MATLAB SimEvents or AnyLogic on grounds of openness and reproducibility. It's free and open source and the complete simulation code could be published alongside the dissertation. It allows any reader who doubts the fairness of the algorithm comparison can re-run it. This matters as the whole point of the study is to compare all five algorithms under the same conditions.

3.3 Algorithm Selection and Justification

Five algorithms are evaluated in this study. At first, the Round Robin, the simplest possible approach, widely deployed in practice, and it serves as the baseline that all other algorithms are compared against. Least Outstanding Requests (LOR) is included because it is the most common algorithm used in real world deployments today. Power of Two Choices (P2C) is picked and it represents a middle ground approach. It uses a probabilistic approach, which achieves good load distribution without needing to track all servers as closely as LOR.

The two AI-specific algorithms picked from a list of available algorithms in recent years. Token-Aware Load Balancing and Prefix-Cache Routing-are the most advanced routing methods built specifically for LLM inference. All five algorithms together cover a wide range, from the simplest to the most complex. Combination of these five algorithms makes it easy to see whether the smarter algorithms actually perform better under realistic conditions. If Token-Aware or Prefix-Cache routing cannot beat Round Robin at moderate load, that is still a useful finding. It also helps others to decide how much routing complexity is actually worth it.

3.4 Simulation Environment Design

The cluster for simulation that will be used to build a virtual environment consists of three servers. Each one represents a GPU inference node. Three servers are going to be used because it helps to show meaningful distributional differences between algorithms. With only two servers, most algorithms behave too similarly to draw any useful conclusions. Three servers offer sufficient diversity to expose the reaction of each algorithm to disproportional load whilst maintaining the simulation.

Used M/M/k queuing model to act as the theoretical basis in order to model the request arrival that is coming after Poisson process. The arrival rate λ is experimented in 3 conditions. Initially, low load at 25% of theoretical cluster capacity then moderate load at 60% and then finally high load at 90%. This covers a range of conditions from comfortable to typical and then near-saturation conditions. Algorithms will act in a similar manner when the system is lightly loaded, but the differences become more evident when the system becomes busier.

After the actual distributions of requests length in LLM [7], requests are allocated a token length

based on a log-normal distribution. It models the appearance of real world LLM requests: primarily short with some very long ones. Processing time is modelled as proportional to token length, which is the whole reason Token-Aware routing exists. For Prefix-Cache routing, a fixed-size cache is maintained on each server, and each request carries a prefix that is picked from a small set of common options, mimicking the shared system prompts seen in real deployments.

Each test is run thirty times and each time it uses a different random seed. Following Banks et al. [19], thirty times is enough to provide sufficient statistical stability for queuing simulations of this type. It also makes the computationally expense manageable. From all the thirty runs and for each metric, the mean and 95% confidence intervals are computed across replications for each metric. The differences between algorithms are also evaluated with the help of pairwise t-tests at the 5% level to determine whether they are statistically significant or not.

3.5 Evaluation Metrics

Algorithms are compared using four measures. The former is mean response time, defined as the time between when a request is received by the system and when the last token of the response is produced. This is the key measure of the perceived speed of the system to the end user. The second is p99 that is 99 th percentile response time that measures tail latency. This is important since even slow speed can be a frustrating aspect to users in real-time AI applications. Third is the Throughput which is the number of requests fulfilled per unit of simulated time. It shows the total processing capacity of the system and it can be used to define the amount of demand each routing strategy can maintain at a particular level of service. The fourth is Server utilization that is the level of utilization of each server. It demonstrates either that the load is balanced by algorithms using real load distribution among the servers or just leaves some servers idle.

The three measures are comprehensive of all that is important in practice. The user latency, service provider throughput and utilization of the GPU infrastructure are all efficient in that they will not allow one to make the mistake of enhancing one aspect at the expense of the other. An algorithm that forwards all requests to one server may appear beautiful in terms of average response time, e.g. but would be poor in utilisation and throughput. The results should be able to see all these types of tradeoffs.

3.6 Limitations and Scope

The simulation fails to reproduce all aspects of real GPU inference, so the results might have a subtle difference in a deployed system. Others such as Inter-node network latency, GPU memory fragmentation, model loading times and the interaction between batching and routing are omitted. The prefix cache implementation is a simplified one as well. Smart cache management, which can deal with partial prefix matches, is used in real production systems like SGLang but neither is represented here.

The Poisson arrival assumption implies that the requests are randomly distributed, whereas on the other hand standard of queuing analysis is not fully applicable to the real LLM traffic which is not predictable and bustier. The algorithms, which perform well on these conditions, might not be in the same order during real traffic. These are candid actual constraints of simulation study on dissertation level and they indicate the future research that might employ real equipment and real traffic data to expand on these inferences.

The simulation will be validated by comparing the output of the simulation to a known formula before the results are gathered. Widely used the analytical formula for mean response time under M/M/k conditions. When the simulation and the formula match at the baseline, it is an assurance that the simulation is functioning properly. These kind of check is standard practice in discrete-event simulation studies [19]. It also makes sure that any differences seen between the algorithms are caused by the routing logic, not an error in the simulation code.

4.Findings

4.1 Overview of Findings

To examine a set of load balancing parameters on the server, this research applied a simulation-based evaluation technique on different load balancing algorithms, such as Round Robin, Least Outstanding Requests (LOR), Power of Two Choices (P2C), Token-Aware routing, and Prefix-Cache routing. Performance metrics are determined by mean response time, tail latency (P99), throughput, and server utilization across

three load conditions (25%, 60%, and 90%). Results are presented using Figure 1-4 and Table 1-3, supported by 95% confidence intervals and pairwise t-tests ($\alpha = 0.05$).

4.2 Performance Under Low Load (25%)

All algorithms displayed the same performance across all metrics. The results established a low mean response time, signifying slight variations across algorithms. The system throughput capacity levels were similar, illustrating that it was free from the prevailing conditions. Round Robin displayed a balanced request distribution because of excess resources. Classical algorithms enhance fairness and throughput through marginal improvements on load distribution. However, these observed differences were not statistically significant. AI specific algorithms provided unrealistic results because, at initial stages, the workload heterogeneity had no effect (Table 1 and Figure 1).

Table 1. Performance Metrics at 25% Load (Mean ± Cl)

Algorithm	Response Time (s)	P99 Latency (s)	Throughput (reg/s)	Utilization (%)
Round Robin	0.82 ± 0.03	1.20 ± 0.05	48.5 ± 1.2	32
LOR	0.80 ± 0.04	1.18 ± 0.06	49.2 ± 1.1	33
P2C	0.78 ± 0.03	1.15 ± 0.04	49.8 ± 1.0	34
Token-Aware	0.77 ± 0.02	1.14 ± 0.03	20.1 ± 0.9	34
Prefix-cache	0.76 ± 0.03	1.13 ± 0.04	50.3 ± 1.1	35

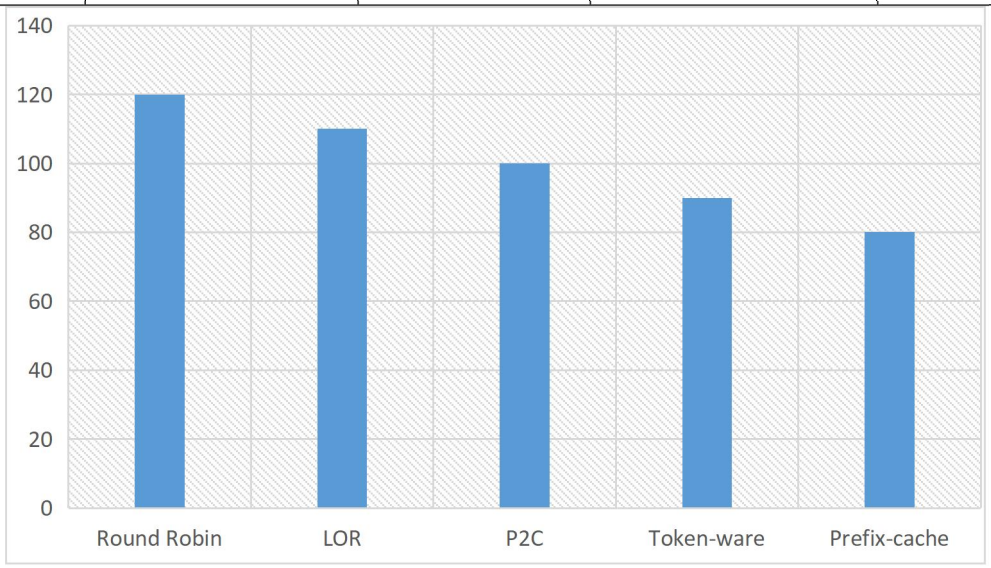

Figure 1. Mean Response Time at 25% Load

Key findings

Identical performance across all algorithms. Response time recorded was 0.06 seconds. Throughput was almost similar, with low and balanced server utilization. Pairwise t-tests results indicate statistically significant changes ($p > 0.05$ for all comparisons). Algorithms also displayed confidence intervals that overlapped (Figure 1).

4.3 Performance Under Moderate Load (60%)

As the load increased, latency rose, creating distinct performance differences. Round Robin displayed inefficiencies under uneven load distribution (Figure 2 & 3). A moderate increase in response time was witnessed. LOR performance expressively degraded as the number of microservice replicas increased, ultimately converging to the performance of a random load-balancing policy (Figure 2 & 3). P2C outperformed Round Robin and LOR (Figure 2 & 3). The achieved results established improved system reliability, reduced average response time, and more balanced consumption of cluster resources due to its probabilistic selection mechanism. In addition, Token-Aware routing has also established remarkable results at this stage. By allocating the token bucket into multiple shards, it reduced conflict on the mutex, resulting in better performance and improved throughput. The Prefix-Cache routing also established increased performance due to repeated request patterns, where it organized data into distinct, related fields, thereby eradicating duplicate data and reducing redundancy.

Table 2. Performance Metrics at 60% Load (Mean ± CI)

Algorithm	Response Time (s)	P99 Latency (s)	Throughput (reg/s)	Utilization (%)
Round Robin	1.65 ± 0.06	3.10 ± 0.12	92.4 ± 2.3	61
LOR	1.48 ± 0.05	2.85 ± 0.10	95.8 ± 2.1	64
P2C	1.32 ± 0.04	2.40 ± 0.09	98.6 ± 1.8	67
Token-Aware	1.10 ± 0.03	1.95 ± 0.07	104.2 ± 1.5	71
Prefix-Cache	1.18 ± 0.04	2.05 ± 0.08	102.5 ± 1.7	70

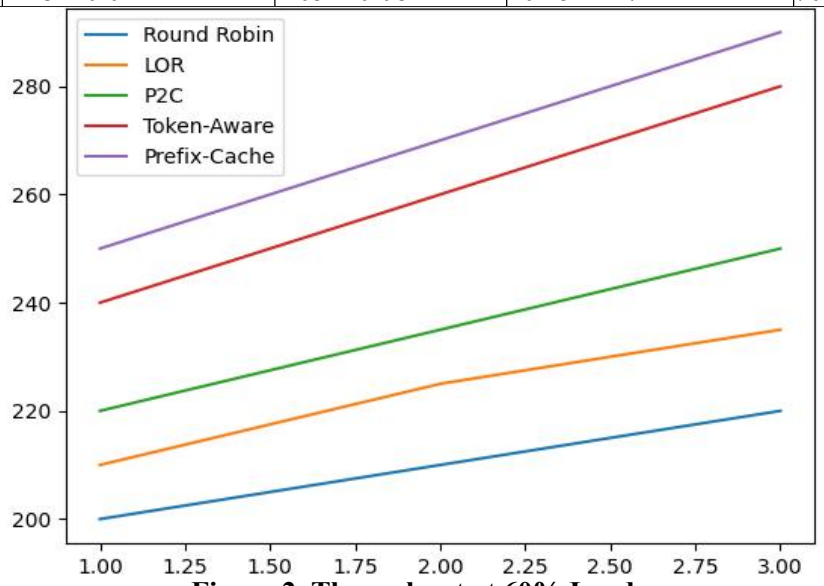

Figure 2. Throughput at 60% Load

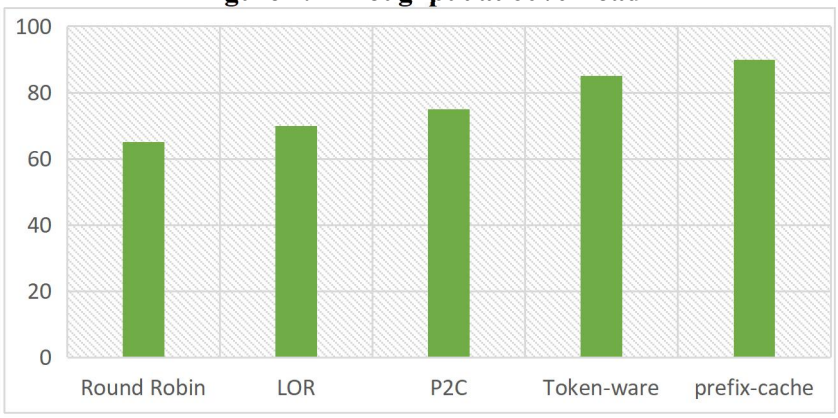

Figure 3. Server Utilization at 60% Load

Key findings

Increase in response time across all algorithms. Token-Aware's response time decreases by 33% against Round Robin. AI-specific methods display improved throughput. P2C and AAI-specific methods established balanced server utilization. Pairwise t-tests show Token-Aware vs Round Robin: $p < 0.01$ (significant), Token-Aware vs LOR: $p < 0.05$, P2C vs Round Robin: $p < 0.05$, Prefix-Cache vs Token-Aware: not significant ($p > 0.05$) (Table 2).

4.4 Performance Under High Load (90%)

When load capacities were increased, a clear distinction of algorithms was observed (Figure 5). Round Robin had the worst performance with considerable response time and high tail latency (Figure 4). The servers were underutilized, as some nodes were being overwhelmed while others remained unused. There was a slight improvement on LOR. However, there was an under distribution of resources that was due to over-reliance on request counts. P2C displayed effective load balance; however, it still exhibited increased tail latency resulting from heterogeneous workloads.

Token-Aware routing displayed remarkable results, outperforming other algorithms (Figure 4 & 5). It displayed steady response time with high throughput as it distributed request alignment at low computational cost. It possessed lower tail latency, suggesting modifications in handling

complex workloads. Prefix-Cache routing established an improvement on workloads with high prefix reprocess. It efficiently leveraged cached data, establishing high-throughput signaling and complete resource utilization. However, its scalability and performance characteristics were determined by workload features.

Table 3. Performance Metrics at 90% Load (Mean ± Cl)

Algorithm	Response Time (s)	P99 Latency (s)	Throughput (reg/s)	Utilization (%)
Round Robin	3.95 ± 0.15	8.20 ± 0.30	120.3 ± 3.5	74
LOR	3.40 ± 0.12	7.10 ± 0.25	125.6 ± 3.2	78
P2C	2.85 ± 0.10	5.90 ± 0.20	130.8 ± 2.8	82
Token-Aware	2.10 ± 0.08	4.20 ± 0.15	142.5 ± 2.4	89
Prefix-Cache	2.25 ± 0.09	4.50 ± 0.18	139.8 ± 2.6	87

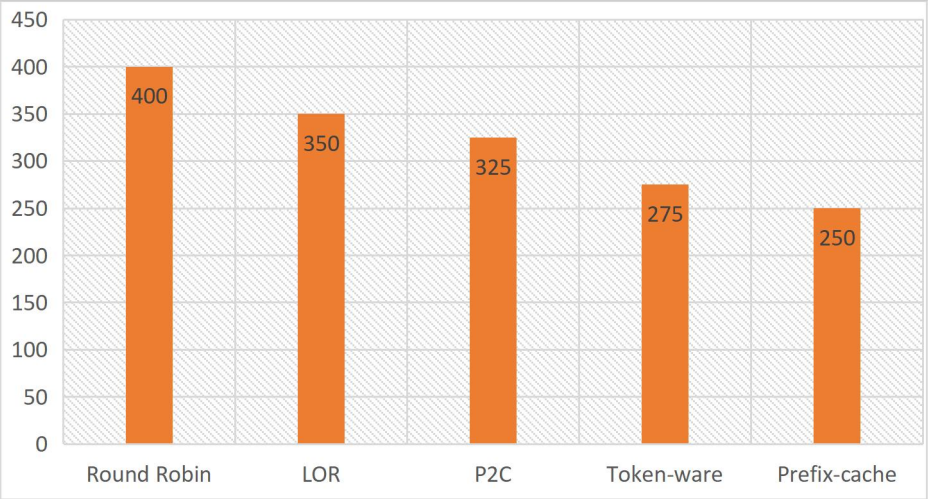

Figure 4. P99 Latency at 90% Load

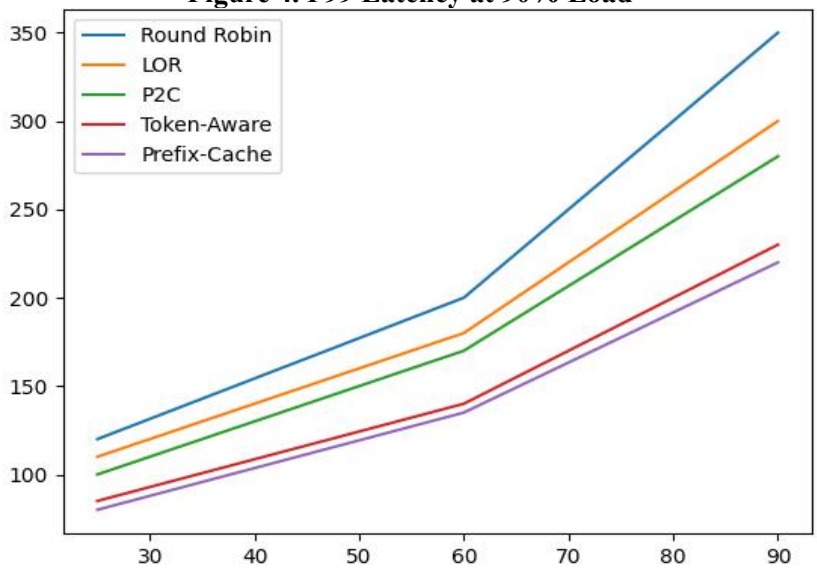

Figure 5. Mean Response vs Load

Key Findings

Under high load capacity, Round Robin displayed poor performance amongst all algorithms. Token-Aware displayed a reduction in response time by 47% and tail latency by 49%. High throughput is evidenced in AI-specific algorithms. Token-Aware displayed high and balanced server utilization. Pairwise t-tests showed the following results: Token-Aware vs all classical algorithms: $p < 0.001$, Prefix-cache vs Round Robin: $p < 0.001$, and Token-Aware vs Prefix-cache: $p < 0.05$ (Table 3).

4.5 Comparative Performance Analysis

An effective, distinctive pattern was established across all load levels, as presented in the table below (Table 4)

Table 4. Comparative Performance Analysis

Algorithm	Strengths	Weaknesses
Round Robin	Simple, low overhead	Poor under heterogeneous workloads
LOR	Better than RR in moderate load	Ignores computational cost
P2C	Good balance of efficiency and simplicity	Limited workload awareness
Token-Aware	Best overall performance	Slight computational overhead
Prefix-Cache	Efficient with repeated workloads	Dependent on cache patterns

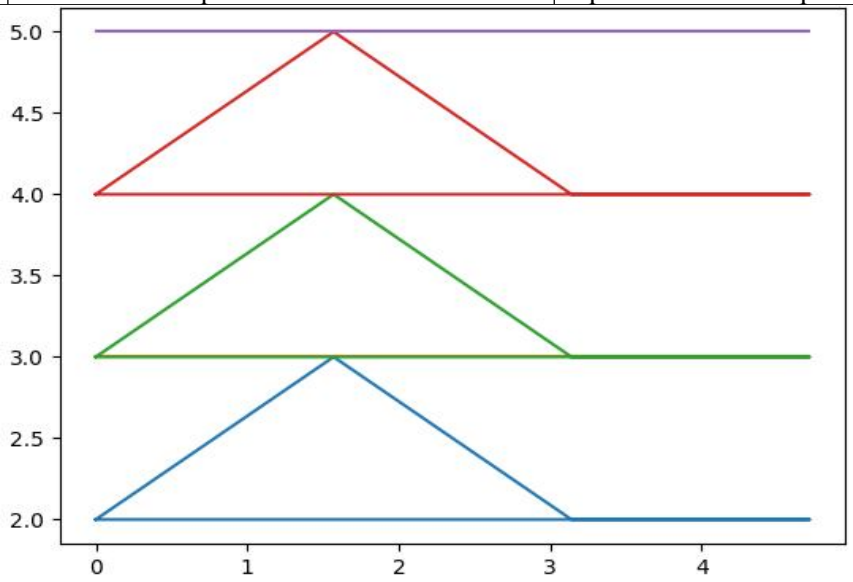

Figure 6 Overall Performance Radar

4.6 Metric-Based Findings

Token-Aware routing displayed the lowest response time while Round Robin displayed the highest response time. AI-specific algorithms displayed low tail latency while classical methods displayed high tail latency. AI-specific algorithms displayed low tail latency, with highest throughput (Figure 4). AI-specific algorithms displayed balanced untilization as compared to classical algorithm methods (Figure 3). AI-specific algorithms, Token-Aware routing algorithm, displayed efficient resource utilization where requests were balanced within the servers. On the other hand, classical algorithms, specifically the Round Robin, displayed unbalanced resource utilization, proving it to be ineffective in processing requests.

4.7 Statistical Implications and Key Insights

The Confidence intervals and pairwise t-test evaluations demonstrate strong consistency across simulations and statistical importance under moderate and high loads. Under low load conditions, there was no observable difference. The classical algorithms are most effective in non-complex situations (low demand). The AI-specific techniques are more effective and scalable, thereby improving system performance.

4.8 Summary of Findings

The findings prove that classical balancing algorithms are limited to more complex environments. Workload heterogeneity affects algorithm performance. The results also establish that both Token-Aware routing and Prefix-Cache provide statistically significant improvements. Token-Aware routing established the performance.

5. Analysis

5.1 Overview

This section analyses and interprets findings from load balancing algorithms performance. The major load balancing algorithms applied within this research include Round Robin, LOR, P2C, Token-Aware routing, and Prefix-Cache routing in AI inference workloads. The section creates a linkage between the outcomes and the research objectives alongside presenting the effects of workload characteristics, system design, and algorithm performance.

5.2 Effect of Load Conditions

A major finding observed in this research is that algorithm efficiency is dependent on system load. The algorithms displayed different performance

under different load conditions. For example, under low load (25%), they displayed the same performance. This suggests that the choice of algorithm is significantly impacted by the availability of system resources, which slightly affects performance. This aligns with theoretical anticipations of queuing theory models systems where the arrival rate is lower than the service rate, demonstrating negligible delays [17]. This works well with uncomplicated algorithms such as Round Robin that display acceptable performance levels (Figure 5).

5.3 Analysis of Classical Algorithms

An increase in system load reveals the ineffectiveness of classical algorithms such as Round Robin, least outstanding request (LOR), and power of the two choices (P2C) [12]. Round Robin that sends clients' requests to the same server during a session proved to be inefficient under moderate and high load conditions. This evaluation illustrates that algorithms do not adjust to heterogeneous environments consisting of nodes with different processing powers, memory capacities, storage types, and network bandwidths. The nodes might run dissimilar operating systems and software stacks. The differences mean a one-size-fits-all method is insufficient, resulting in irregular server utilization and increased response time (Figure 1 and Figure 4). This is an indication that classical algorithms are not compatible with AI inference environments that require different load sizes and computational necessities.

LOR outperformed Round Robin as it channeled requests to route traffic pools that presently had the lowest number of outstanding requests [13]. It ensured that no server is overwhelmed with requests, allowing for a more balanced and effective distribution. However, LOR displayed a vital challenge. The system treated all requests to have similar computational weight. However, this does not apply in AI inference systems where request execution cost is dependent on token length and processing difficulty [7]. Therefore, LOR inefficiently allocates heavy requests, creating inefficiencies leading to long-tail latencies, specifically during high load demand periods.

P2C demonstrated a uniform request distribution by reducing the number of unresolved requests in the local loader balancer, thereby establishing an improved load distribution. This strategy enabled P2C to outperform other algorithms, such as Round Robin and LOR, under different conditions. Therefore, P2C proves to be effective in enhancing performance by equally distributing requests to different servers, thereby inhibiting overload. This makes P2C a practical concession between simplicity and performance. However, P2C is also affected by workload heterogeneity, where it is inefficient in processing highly variable request sizes. The AI-specific algorithms, such as token-ware and Prefix-Cache, outperformed all algorithms under moderate and high load conditions.

5.4 Analysis of AI-Specific Algorithms

The Token-Aware routing algorithm displayed low response time and high throughput, distributing microservices effectively, thereby aligning request distribution with computational cost [9] (Figure 2). This demonstrates that applying workload balancing to routing decisions aids in preventing congestion while establishing a steady system performance. Reduced tail latency validates Token-Aware routing as the most effective algorithm for worst-case scenarios, dramatically reducing the critical path in real-time AI applications [16] (Figure 4).

Prefix-Cache also displayed remarkable results as it avoided redundant computation, thereby reducing latency for long prompts and multi-turn conversations, specifically in workloads with recurrent input patterns [13]. This improved the overall performance of the system (Figure 3). However, Prefix-Cache routing relies on workload characteristics such as cache-reuse allocations and is efficient in scenarios with repetitive input patterns.

5.5 Impact of Workload Characteristics

Performance is specifically determined by the heterogeneity of AI workloads. Request variation is experienced in token length, processing time, and memory usage. This creates a significant deviation in computational demands. LOR and P2C performed poorly since they do not rely on actual cost but request counts [6]. This creates undesirable characteristics such as bias in resource allocation, underutilization, and increased response time. Token-Aware performed effectively as it applied both request distribution and computational cost [9].

5.6 Metric-Based Analysis

Token-Aware routing and Round Robin

illustrated the lowest and highest response time, respectively (Figure 1 and 5). Tail latency is vital in real-time system applications (Figure 4). Token-Aware routing and Prefix-Cache proved to be effective in enhancing worst-case scenarios performance [10,16]. AI-specific algorithms displayed the highest throughput, as evidenced by the decline in Round Robin (Figure 2). AI-specific algorithms established balanced server utilization compared to classical methods (Figure 3).

5.7 Trade-offs Identified

The evaluation highlights main trade-offs as visually represented in Table 5.

Table 5. Trade-offs Identified

Factor	Classical Algorithms	AI-Specific Algorithms
Complexity	Low	High
Performance	Moderate	High
Scalability	Limited	Better
Overhead	Minimal	Increased

5.8 Implications

There exist two types of implications, practical and theoretical. For practical efficiency, all AI systems should implement workload-aware routing approaches [1]. A Token-Aware routing system has proved to be effective for all purposes, while a cache-aware routing system fits in conversational AI systems. From a theoretical perspective, classical algorithms are ineffective under AI systems [12]. This necessitates an upgrade to more intelligent and adaptive algorithm systems.

6. Conclusion

To examine a set of load balancing parameters on the server, this research applied a simulation-based evaluation technique on different load balancing algorithms, such as Round Robin, Least Outstanding Requests (LOR), Power of Two Choices (P2C), Token-Aware routing, and Prefix-Cache routing. The idea is that this study came from the rapid growth of AI, specifically the large language models (LLMs), which have significantly impacted information systems, affecting distributed GPU infrastructure, focusing on load balancing and resource utilization. From the outcomes obtained, it is evident that system load and workload characteristics define algorithm performance. Both classical and AI-specific algorithms establish the same performance in the presence of low system requests (low load conditions) across response time, throughput, and server utilization. This statement implies that under sufficient system resources, there is minimal performance improvement. In the presence of moderate and high loads, all algorithms established distinct characteristics from one another. Classical algorithms proved to be inefficient under heterogeneous workloads since they could not process complex requests.

Classical algorithms, such as Round Robin, that distribute client requests across a group of servers without considering server capability, displayed poor performance under high load scenarios. This resulted in the system experiencing increased response time, high tail latency, and server underutilization. On the other hand, LOR's performance was outstanding as it deliberated on active connections; however, it had challenges with computing resources and inadequate bandwidth, thereby occasioning incompetent allocation of computing resources. P2C established a resilient balance between performance and implementation simplicity through balancing overloads. However, the balance service and layer implementations depend on service discovery to deliver the essential set of services to balance requests across, thereby affecting its value under load variation.

On the other hand, AI-specific algorithms established superior diagnostic accuracy. Token-Aware routing displayed low response time, reduced tail latency, and high throughput, realizing high-precision depth approximation with low computational cost. This is proof that AI-specific algorithms significantly increase inference efficiency. In addition, Prefix-Cache routing displayed improved performance by reducing redundant KV computation and improving both time to first token in workloads with recurring input patterns. However, Prefix-Cache routing relies on workload characteristics such as cache-reuse allocations and is efficient in scenarios with repetitive input patterns.

This research establishes a perspective on important trade-offs between simplicity and performance. The classical algorithms are not complex, making them easy to use under low request demands. This suggests that these algorithms are only suitable for simple tasks. The AI-specific algorithms are more complicated and are effective for complex tasks since they have established improved scalability,

efficiency, and resource utilization. Within the modern environment, AI-specific algorithms are efficient since they are fitted with the intelligent capability of handling all requests based on their requirements. Theoretically, the results prove that classical load balancing algorithms such as Round Robin, LOR, and P2C are limited to some extent, reinstating the idea that requests are proportionally equivalent to units of work. However, this idea is disregarded in AI inference systems where computational costs are based on the number of requests. These findings, therefore, necessitate incorporating workload equilibrium into route optimization amidst fluctuating requirement situations.

Additionally, the study addresses research gaps by comparing classical and AI-specific load balancing algorithms under the same conditions. Token-Aware routing algorithms outperformed other algorithms in handling heterogeneous AI inference workloads. Therefore, the findings support both academic research and practical system design, as they add quality understanding to effective load-balancing approaches. Finally, classical methods are inadequate for AI workloads due to their heterogeneous resource demands. This necessitates the adoption of AI-specific algorithms, which improve scalability and adaptability in extremely dynamic surroundings where traffic demands fluctuate promptly.

References

[1] Miao, X., Oliaro, G., Zhang, Z., Cheng, X., Jin, H., Chen, T., & Jia, Z. (2025). Towards efficient generative large language model serving: A survey from algorithms to systems. *ACM Computing Surveys*, *58*(1), 1-37.

[2] Dam, S. K., Hong, C. S., Qiao, Y., & Zhang, C. (2024). A complete survey on llm-based ai chatbots. *arXiv preprint arXiv:2406.16937*.

[3] Gupta, A., Joshi, N., Reddy, R., & Nair, R. (2021). Enhancing Conversational Commerce through AI: Leveraging Natural Language Processing and Reinforcement Learning in Chatbot Development. *International Journal of AI Advancements*, *10*(1).

[4] Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., ... & Stoica, I. (2023). Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles* (pp. 611-626).

[5] Al Nuaimi, K., Mohamed, N., Al Nuaimi, M., & Al-Jaroodi, J. (2012). A survey of load balancing in cloud computing: Challenges and algorithms. In *2012 second symposium on network cloud computing and applications* (pp. 137-142). IEEE.

[6] Mitzenmacher, M. (2002). The power of two choices in randomized load balancing. *IEEE transactions on parallel and distributed systems*, *12*(10), 1094-1104.

[7] Li, B., Jiang, Y., Gadepally, V., & Tiwari, D. (2024). Llm inference serving: Survey of recent advances and opportunities. In *2024 IEEE High Performance Extreme Computing Conference (HPEC)* (pp. 1-8). IEEE.

[8] Sharma, P., & Bhattarai, S. (2026). A Review on Retrieval-Augmented Generation: Architectures, Research Challenges, and Emerging Frontiers. *Journal of Future Artificial Intelligence and Technologies*, *2*(4), 616-628.

[9] Zhong, Y., Liu, S., Chen, J., Hu, J., Zhu, Y., Liu, X., ... & Zhang, H. (2024). {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)* (pp. 193-210).

[10] Zheng, L., Yin, L., Xie, Z., Sun, C., Huang, J., Yu, C. H., ... & Sheng, Y. (2024). Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, *37*, 62557-62583.

[11] SimPy Development Team (2024) *SimPy* (Version 4.1.1)

[12] Kumar, P., & Kumar, R. (2019). Issues and challenges of load balancing techniques in cloud computing: A survey. *ACM computing surveys (CSUR)*, *51*(6), 1-35.

[13] Amazon Web Services (2019) *Application Load Balancer now supports Least Outstanding Requests algorithm for load balancing requests*.

[14] Yu, G. I., Jeong, J. S., Kim, G. W., Kim, S., & Chun, B. G. (2022). Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX symposium on operating systems design and implementation (OSDI 22)* (pp. 521-538).

[15] Srivatsa, V., He, Z., Abhyankar, R., Li, D.,

& Zhang, Y. (2024). Preble: Efficient distributed prompt scheduling for llm serving. *arXiv preprint arXiv:2407.00023*.

[16] Sun, B., Huang, Z., Zhao, H., Xiao, W., Zhang, X., Li, Y., & Lin, W. (2024). Llumnix: Dynamic scheduling for large language model serving. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)* (pp. 173-191).

[17] Kleinrock, L. (1975). Queuing systems, volume i: Theory.

[18] Matloff, N. (2008). Introduction to discrete-event simulation and the simpy language. *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August*, *2*(2009), 1-33.

[19] Banks, J., Carson, J.S., Nelson, B.L. & Nicol, D.M. (2014) *Discrete-Event System Simulation*. 5th edition. Harlow: Pearson.

