STEADY-STATE QUEUE-LENGTH TAIL ANALYSIS OF LLM SERVING SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

The growing adoption of large language model (LLM) inference places increasing pressure on service providers. Although some serving systems can use coarse estimates to prevent unbounded request queue growth caused by overload, the lack of accurate estimates of LLM serving capacity may result in persistently long queues even in steady state, substantially degrading inference service quality. To address this issue, we combine batching mechanisms specific to LLM inference with classical queueing analysis and develop a framework for predicting arbitrary-time steady-state queue-length tail probabilities under different batching strategies. The framework accounts for random request lengths and service times associated with different batching strategies to estimate the probability that the number of requests awaiting service exceeds a specified threshold. It also enables comparisons across arrival rates, batch sizes, and service configurations. Validation experiments on a real server show that, as the system approaches full utilization, the proposed mathematical model accurately estimates queue-length tail probabilities across different arrival rates.

1 INTRODUCTION

Large language model (LLM) serving systems must continuously respond to inference requests from conversational applications, retrieval-augmented generation, and AI agents, and their responsiveness directly affects the user experience (Lin et al., 2024; Zheng et al., 2024; Liu et al., 2024). As demand for these services continues to grow, incoming requests place increasing pressure on serving platforms. Improving LLM services to use limited computational resources more effectively and provide more timely responses has therefore become an important challenge for service providers.

Evaluation-driven approaches (Agrawal et al., 2024a; Yang et al., 2024; Nie et al., 2026) characterize workloads and inference performance through prediction, simulation, and time modeling, providing stability conditions and guidance for configuration optimization. Execution-oriented studies (Agrawal et al., 2024b; Zhong et al., 2024; Patel et al., 2024; Sun et al., 2024) improve resource utilization and responsiveness through batching optimization, inference-stage disaggregation, and request migration. However, execution efficiency and stability alone do not ensure service quality. Even stable, well-optimized systems can accumulate excessive backlogs under resource constraints, causing users to lose patience and reducing their willingness to pay. Conversely, insufficient demand may leave preallocated resources idle and reduce economic returns.

Predicting queueing behavior across workloads and resource configurations can guide rate limiting, model configuration, and hardware upgrades, reducing both service degradation and resource waste. For LLMs, such prediction must jointly account for autoregressive generation and batched execution: requested generation lengths, batch formation rules, and execution policies all affect serving capacity. Without this joint characterization, providers cannot reliably anticipate request accumulation before deployment or compare queueing risks across configurations.

To accurately characterize request queueing, we build on existing models of batched inference time and combine classical queueing theory with probabilistic analysis to develop a steady-state queue-length tail probability analysis method for fixed batching and dynamic elastic batching. The method accounts for random request lengths and the service characteristics of different batching strategies to predict the probability that the number of requests awaiting service exceeds a specified queue-length threshold. It also supports comparisons across arrival rates, batch sizes, and service configurations,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

providing a quantitative basis for assessing request accumulation risks and selecting system parameters or hardware configurations.

The main contributions of this paper are as follows:

- We develop a queue-length analysis method for fixed batching and dynamic elastic batching that incorporates random request lengths and batched execution characteristics into a statistical characterization of the waiting queue.
- We derive expressions for arbitrary-time steady-state waiting queue-length tail probabilities under both batching strategies and provide numerical methods for their computation, enabling queueing risk assessment across different workloads and service configurations.
- We validate the theoretical predictions through experiments on a real server. The results show that the proposed method accurately estimates queue-length tail probabilities across different arrival rates, even as the system approaches full utilization.

2 Related Work

2.1 LLM Inference Serving Optimization

To improve LLM service quality, existing research enhances execution efficiency through better resource organization and request scheduling. Mooncake (Qin et al., 2025) organizes disaggregated computation and storage around the KV cache to reduce redundant computation and support request scheduling, while Preble (Srivatsa et al., 2025) coordinates prefix reuse and load balancing in distributed serving. For long-context requests, LoongServe (Wu et al., 2024) adjusts the computational resources allocated to requests through elastic sequence parallelism. ServerlessLLM (Fu et al., 2024) reduces service startup overhead through fast model loading and inference migration, supporting on-demand resource provisioning.

Beyond execution and resource management, several studies further consider the service requirements of different requests and applications. ConServe (Qiao et al., 2024) coordinates online and offline inference through fine-grained scheduling and preemption, utilizing idle resources while preserving the latency requirements of online requests. VTC (Sheng et al., 2024) allocates processing opportunities based on the service costs of input and output tokens to improve fairness across users. Autellix (Luo et al., 2025) uses program-level execution information to prioritize LLM calls, reducing cumulative waiting times in agentic applications.

2.2 LLM Workload Modeling and Performance Analysis

Modeling request execution costs and resource requirements is fundamental to evaluating LLM serving performance. LLMCompass (Zhang et al., 2024) provides a hardware evaluation framework for LLM inference that predicts execution latency and cost across different hardware designs. QLM (Patke et al., 2024) estimates request waiting times and meets service objectives through queue reordering and model management. TRAIL (Shahout et al., 2025) uses internal model representations to predict remaining output lengths and reduces request response times through limited preemption. These studies inform resource selection and scheduling decisions by characterizing hardware costs and request execution characteristics.

On the theoretical side, Mitzenmacher & Shahout (2025) systematically discuss LLM queueing and scheduling problems involving length prediction, batching, and GPU memory constraints, highlighting the challenges of adapting conventional queueing models to autoregressive inference. Ao et al. (2025) use a fluid model to analyze serving capacity under memory constraints and design online scheduling policies accordingly. Such studies connect request processing costs with system capacity, providing a theoretical basis for understanding how workload variations affect serving performance.

3 Preliminaries

We review established inference-time models for fixed and elastic batching. These models characterize within-batch execution and connect random request lengths to the steady-state queue-length analysis in the next section.

3.1 Factors Affecting Inference Time

For a typical autoregressive LLM, request inference consists of two consecutive stages. First, the prefill stage processes the input and generates the first token. Subsequently, the decode stage proceeds iteratively based on the existing context, generating one new token for each unfinished request at each step. Prefill latency is primarily affected by the batch size and the number of input tokens, and also includes overhead associated with GPU kernel launches, input data transfers, and operator scheduling. Decode latency primarily depends on the batch size and the number of decoding iterations, together with overhead similar to that incurred during prefill.

Based on the factors discussed above, Yang et al. (2024) use linear approximations to model the execution time of the two LLM inference stages in terms of the batch size and the maximum output token length within the batch. Notably, their model does not explicitly account for input token length, as their validation experiments show that its effect on overall inference latency is relatively small when the output is sufficiently long. They also describe three batching strategies to which their modeling approach applies: dynamic batching, fixed batching, and elastic batching. To obtain a more general batching strategy, we combine the rules of dynamic and elastic batching to better reflect the batching behavior of contemporary inference services. We refer to this combined strategy as dynamic elastic batching. The following subsections describe the two batching strategies considered in this work and their corresponding inference time models in detail.

3.2 Inference Time under Fixed Batching

Fixed batching forms each batch from exactly b requests at the head of the queue, where b is a given positive integer. If the server is idle but fewer than b requests are waiting, it remains idle until enough requests have accumulated. Batch execution is neither suspended nor preempted, and no new requests are admitted to a batch during execution. To represent the inputs and outputs as rectangular tensors, fixed batching uses padding to align their token lengths. Shorter inputs are padded to match the longest input in the batch, while requests that complete decoding earlier remain in the computation by generating padding tokens until the longest output is completed.

Yang et al. (2024) express the inference time of a single batch under fixed batching as follows:

$$H^l = k_1 b + k_2 + k_3 bl + k_4 l, \tag{1}$$

where $k_1 > 0$, $k_2 \geq 0$, $k_3 \geq 0$, and $k_4 \geq 0$. The prefill time required to generate the first token is represented by $k_1 b + k_2$, with k_1 denoting the incremental prefill time per additional request in the batch and k_2 denoting the fixed prefill overhead. The time required to generate subsequent tokens through iterative decoding is represented by $k_3 bl + k_4 l$. The coefficient k_3 denotes the incremental time per decoding step per additional request in the batch, while k_4 denotes the fixed overhead per decoding step. The variable l denotes the maximum output token length within the batch, excluding the first output token. Since b remains fixed during inference service, we treat it as a constant.

3.3 Inference Time under Dynamic Elastic Batching

Dynamic elastic batching relaxes the batch formation and request departure rules of fixed batching. The dynamic rule allows the batch size b to vary over positive integers: whenever requests are present in the waiting queue, the system can group them into a batch for joint processing. The elastic rule (Yu et al., 2022) eliminates padding during decoding, allowing shorter requests to leave immediately upon completion, so that subsequent decoding steps process only the remaining requests.

Yang et al. (2024) provide the following expression for the inference time of a single batch under elastic batching, which also applies to dynamic elastic batching:

$$\begin{aligned} H^{[b,\boldsymbol{n}]} &= k_1 b + k_2 + (k_3 b + k_4)\, n_1 + (k_3(b-1) + k_4)\,(n_2 - n_1) \\ &\quad + \cdots + (k_3 1(b-(b-1)) + k_4)\,(n_b - n_{b-1}) \\ &= k_1 b + k_2 + k_3 \sum_{i=1}^{b} n_i + k_4 n_b, \end{aligned} \tag{2}$$

where $\boldsymbol{n} = (n_1, n_2, \ldots, n_b)$ is the vector of per-request decode lengths sorted in nondecreasing order, with n_i denoting the number of tokens generated during decoding, excluding the first output

token, by the ith request in this ordering. This expression captures the inference time under dynamic elastic batching by excluding completed requests from the computation of the time required for subsequent decoding steps.

4 Steady-State Queue-Length Tail Analysis

Building on the inference time formulas established in prior work, this section develops stochastic models of request arrivals and batch inference times for the two batching strategies described in Section 3. We then derive expressions for the steady-state tail probabilities of the waiting queue length observed at an arbitrary continuous-time instant under each strategy.

4.1 System Model

We consider an LLM inference node with a single waiting queue operating under the setting described above. The node follows a first-come-first-served (FCFS) discipline, forming batches from requests at the head of the queue for LLM inference. Requests arrive according to a homogeneous Poisson process with rate $\lambda > 0$, measured in requests per second. The decode-stage output lengths of requests are independent and identically distributed, following the discrete uniform distribution $\mathrm{Unif}\{n_{\min}, n_{\min}+1, \ldots, n_{\max}\}$. To simplify the analysis, we assume that requests awaiting service join an infinite-capacity queue and do not leave the queue because of failures or user impatience.

4.2 Fixed Batching

4.2.1 Inference Time Distribution

Under fixed batching, the batch size b remains constant. Let L_b denote the maximum decode-stage output length within a batch. The variability in inference time across batches is therefore determined solely by L_b. Since the decode-stage output lengths of individual requests follow a uniform distribution, the probability $p_{l'}$ that L_b equals l' for a given batch is obtained as follows:

$$p_{l'} := \Pr(L_b = l') = \frac{(l' - n_{\min} + 1)^b - (l' - n_{\min})^b}{(n_{\max} - n_{\min} + 1)^b}, \qquad l' \in \{n_{\min}, \ldots, n_{\max}\}. \tag{3}$$

4.2.2 Stationary Distribution of the Embedded Chain

Let Y_n denote the number of new requests arriving during the service of the n-th batch, from service initiation to completion. Conditional on $L_b = l'$, Y_n follows a Poisson distribution with parameter $\lambda H^{l'}$. Applying the law of total probability over all possible values of l' yields the probability α_k that exactly k requests arrive during a batch service:

$$\alpha_k := \Pr(Y_n = k) = \sum_{l'=n_{\min}}^{n_{\max}} p_{l'} e^{-\lambda H^{l'}} \frac{(\lambda H^{l'})^k}{k!}, \qquad k \in \{0, 1, \ldots\}. \tag{4}$$

To describe queue-length changes between consecutive batches, let X_n denote the number of waiting requests immediately after the n-th batch completes and before requests are removed for the next batch. Under fixed batching, the queue length X_{n+1} at the next batch completion equals the number of requests remaining in the queue when that batch starts plus the number of arrivals during its service, yielding $X_{n+1} = (X_n - b)^+ + Y_{n+1}$. For $X_n = i$ and $X_{n+1} = j$, the queue-length transition probability is given by:

$$\begin{aligned} P_{ij} &:= \Pr\left(X_{n+1} = j \mid X_n = i\right) \\ &= \Pr\left(Y_{n+1} = j - (i-b)^+\right) \\ &= \alpha_{j-(i-b)^+}, \end{aligned} \tag{5}$$

which is the probability that exactly $j - (i - b)^+$ new requests arrive during service. Given the current waiting queue length, the queue-length distribution at the next batch completion does not depend on earlier states. Thus, the sequence $\{X_n\}$ forms an embedded Markov chain. From the preceding results, the mean batch inference time is $\bar{h}_b = \sum_{l'=n_{\min}}^{n_{\max}} p_{l'} H^{l'}$. When $\lambda \bar{h}_b < b$, the mean

queue-length increment is strictly negative outside the finite state set $\{0, \ldots, b-1\}$. Under Poisson arrivals, the chain can reach state 0 from any state and can reach any nonnegative integer state from state 0, establishing irreducibility. Moreover, the positive probability of returning to state 0 in one transition establishes aperiodicity. Together with the drift condition and finite expected increments within the finite set, these properties imply that the chain has a unique stationary distribution to which its state distribution converges.

Assuming that the system is in steady state, let π_i denote the probability that the waiting queue length equals i at a batch completion. These probabilities satisfy the balance equations $\pi_j = \sum_{i=0}^{\infty} \pi_i P_{ij}$ and the normalization condition $\sum_{i=0}^{\infty} \pi_i = 1$. Starting from the initial distribution concentrated at the empty queue, we solve these equations using power iteration, updating the state probabilities until the sum of their absolute differences between consecutive iterations falls below 10^{-13}. To accommodate memory constraints, we truncate the queue-length state space at a finite threshold and temporarily aggregate probability mass beyond this threshold into the last retained state. We progressively increase the truncation threshold and recompute the distribution until the maximum absolute change in the queue-length tail probabilities does not exceed 10^{-8} and both the boundary probability and the unaccounted probability mass do not exceed 10^{-10}.

4.2.3 Queue-Length Tail Probability at an Arbitrary Time

Let Q denote the steady-state waiting queue length observed at an arbitrary point in continuous time. Our objective is to compute $\Pr(Q > x)$, where x is a specified nonnegative integer queue-length threshold. We refer to the interval between two consecutive batch completions as a cycle. Each cycle consists of two phases: an idle phase during which the system waits for the queue to accumulate b requests, and a service phase during which b requests are removed from the queue to form a batch and processed to completion. The first phase occurs only when the initial queue length i is less than b. We obtain $\Pr(Q = x)$ by dividing the expected time spent at queue length x within a cycle by the expected cycle length.

During the first phase, for an initial queue length $i \le x < b$, the queue reaches length x as requests arrive and remains there until the next arrival. Since the expected time between consecutive arrivals is $1/\lambda$, the expected time spent at queue length x during this phase, averaged over the stationary initial-state distribution and denoted by $E_x[W_1]$, is given by:

$$E_x[W_1] = \frac{\mathbf{1}_{\{x<b\}}}{\lambda} \sum_{i=0}^{x} \pi_i, \tag{6}$$

where $\mathbf{1}_{\{x<b\}}$ is an indicator function that equals 1 when $x < b$ and 0 otherwise.

Next, to calculate the occupation time during the second phase, let t denote the time elapsed since inference begins and u the cumulative number of arrivals during inference. Under Poisson arrivals, the probability that exactly u requests have arrived by time t is $e^{-\lambda t}(\lambda t)^u/u!$. Integrating this probability over the inference duration yields the expected time during which the cumulative arrival count equals u. As established earlier, the batch inference time takes the value $H^{l'}$ with probability $p_{l'}$. Therefore, the expected occupation time T_u can be expressed as a weighted sum of these integrals over all possible inference durations:

$$\begin{aligned} T_u &= \sum_{l'=n_{\min}}^{n_{\max}} p_{l'} \int_0^{H^{l'}} e^{-\lambda t} \frac{(\lambda t)^u}{u!} \mathrm{d}t \\ &= \sum_{l'=n_{\min}}^{n_{\max}} p_{l'} \frac{1}{\lambda} \left[1 - e^{-\lambda H^{l'}} \sum_{k=0}^{u} \frac{(\lambda H^{l'})^k}{k!} \right] \\ &= \frac{1}{\lambda} \left[1 - \sum_{k=0}^{u} \alpha_k \right]. \end{aligned} \tag{7}$$

Given an initial queue length i at the beginning of a cycle, $(i-b)^+$ requests remain waiting when inference begins. For the waiting queue length to equal x, exactly $x - (i-b)^+$ new requests must arrive during service. When $i \le x + b$, the queue may spend time at length x during service.

Weighting by π_i, the expected time spent at queue length x during the service phase, denoted by $E_x[W_2]$, is given by:

$$E_x[W_2] = \sum_{i=0}^{x+b} \pi_i T_{x-(i-b)^+}. \tag{8}$$

Using equation 4 and equation 5, the above expression can be further simplified as:

$$\begin{aligned} E_x[W_2] &= \frac{1}{\lambda}\left(\sum_{i=0}^{x+b}\pi_i - \sum_{i=0}^{x+b}\pi_i \sum_{j=0}^{x} P_{ij}\right) \\ &= \frac{1}{\lambda}\left(\sum_{i=0}^{x+b}\pi_i - \sum_{i=0}^{x}\pi_i\right) \\ &= \frac{1}{\lambda}\sum_{i=x+1}^{x+b}\pi_i. \end{aligned} \tag{9}$$

To determine the expected cycle length, note that, under stability and without request loss, the system's long-run average request completion rate equals the arrival rate λ. Since each fixed batch completes exactly b requests, the long-run average batch completion rate is λ/b. The expected cycle length, namely the mean interval between consecutive batch completions, is therefore b/λ. Combining this result with the preceding derivations yields:

$$\begin{aligned} \Pr(Q = x) &= \frac{E_x[W_1] + E_x[W_2]}{b/\lambda} \\ &= \frac{1}{b}\left[\mathbf{1}_{\{x<b\}}\sum_{i=0}^{x}\pi_i + \sum_{i=x+1}^{x+b}\pi_i\right]. \end{aligned} \tag{10}$$

The tail probability can be written as $\Pr(Q > x) = 1 - \sum_{q=0}^{x}\Pr(Q = q)$. Substituting the above expression gives:

$$\Pr(Q > x) = 1 - \frac{1}{b}\sum_{q=0}^{x}\left[\mathbf{1}_{\{q<b\}}\sum_{i=0}^{q}\pi_i + \sum_{i=q+1}^{q+b}\pi_i\right]. \tag{11}$$

4.3 Dynamic Elastic Batching

Since practical LLM serving systems impose an upper limit on the number of requests processed in a single batch, we denote the maximum batch size under dynamic elastic batching by B and derive $\Pr(Q > x)$ under this constraint.

4.3.1 Inference Time Distribution

For a batch with size $b \in \{1, \ldots, B\}$, let N_j denote the decode output length of the jth request before sorting. Define $D_b = \sum_{j=1}^{b} N_j$ and $L_b = \max_{1\le j\le b} N_j$ as the total output length and the maximum individual output length within the batch, respectively. According to equation 2, the randomness of inference time under dynamic elastic batching depends on both D_b and L_b. Since the request lengths are independent and uniformly distributed, the joint probability $w_b(d, l')$ that $D_b = d$ and $L_b = l'$ is given by:

$$\begin{aligned} w_b(d, l') &:= \Pr(D_b = d, L_b = l') \\ &= \frac{[z^d]\left[\left(\sum_{v=n_{\min}}^{l'} z^v\right)^b - \left(\sum_{v=n_{\min}}^{l'-1} z^v\right)^b\right]}{(n_{\max} - n_{\min} + 1)^b}, \end{aligned} \tag{12}$$

where z is a formal variable, z^v represents a request with decode length v, and $[z^d]$ extracts the coefficient of z^d from the polynomial expression that follows. The numerator counts the request-length combinations whose total length is d and whose maximum individual length is exactly l'. The expression in equation 2 can also be rewritten in terms of b, d, and l' as follows:

$$H^{[b,d,l']} = k_1 b + k_2 + k_3 d + k_4 l'. \tag{13}$$

4.3.2 Stationary Distribution of the Embedded Chain

Let $\alpha_{b,k}$ denote the probability that exactly k new requests arrive during the inference of a batch with size b. Reusing the conditional Poisson weighting in equation 4, we replace the length probabilities and service durations with $w_b(d, l')$ and $H^{[b,d,l']}$, respectively, yielding:

$$\alpha_{b,k} = \sum_{d,l'} w_b(d, l') e^{-\lambda H^{[b,d,l']}} \frac{(\lambda H^{[b,d,l']})^k}{k!}. \tag{14}$$

The transition probability under dynamic elastic batching can also be expressed in a form similar to equation 5: $P_{ij} = \alpha_{b,j-(i-B)^+}$. Following a derivation similar to that in Section 4.2.2, we conclude that, when $\lambda \bar{h}_b < B$, the system has a unique stationary distribution to which its state distribution converges, where $\bar{h}_b = \sum_{d,l'} w_b(d, l') H^{[b,d,l']}$.

4.3.3 Queue-Length Tail Probability at an Arbitrary Time

Unlike fixed batching, dynamic elastic batching waits for the first request only when the initial queue length of a cycle is 0, with an expected waiting time of $1/\lambda$. Following the preceding derivation, the expected occupation time during the first phase, $E_x[W_1]$, is given by:

$$E_x[W_1] = \frac{\pi_0}{\lambda} \mathbf{1}_{\{x=0\}}. \tag{15}$$

For the second phase, the integral relationship in equation 7 remains valid when the batch size b varies. Thus, the same derivation and simplification yield $E_x[W_2] = \frac{1}{\lambda} \sum_{i=x+1}^{x+B} \pi_i$. Since the number of requests processed per cycle varies, we calculate its mean, denoted by $\bar{b}$, by weighting three possible cases. When the initial waiting queue length is $i = 0$, the system waits for the first request and then starts a batch containing that single request. When $1 \le i < B$, all waiting requests form a batch of size i. When $i \ge B$, the system processes B requests. Weighting these cases by their stationary probabilities gives:

$$\bar{b} = \pi_0^{\mathrm{E}} + \sum_{i=1}^{B-1} i\pi_i + B \left(1 - \sum_{i=0}^{B-1} \pi_i \right). \tag{16}$$

Reusing the flow-balance argument for fixed batching, the expected cycle length is $\bar{b}/\lambda$. Finally, following equation 11, we obtain:

$$\Pr(Q > x) = 1 - \frac{1}{\bar{b}} \left[\pi_0 + \sum_{q=0}^{x} \sum_{i=q+1}^{q+B} \pi_i \right]. \tag{17}$$

5 Reliability Validation

We validate theoretical $P(Q > x)$ against empirical estimates obtained from queue-length logs during long-running LLM serving experiments. We also measure the calibration time and accuracy of H^l and $H^{[b,d,l']}$, assessing the service-trial overhead required before queue-length prediction.

5.1 Experimental Setup

We conduct the serving simulations on a Linux server equipped with a single RTX 4090 GPU. We deploy Qwen3.5-9B-INT4 using vLLM and follow the FCFS service discipline. Input

and total output token lengths are independently sampled from the discrete uniform distribution $\mathrm{Unif}\{10, \ldots, 200\}$. Fixed batching uses a constant batch size $b = 4$, whereas dynamic elastic batching uses a maximum batch size $B = 8$.

Requests arrive according to the Poisson process with rate λ described in Section 4. We set $\lambda = \rho\lambda_{\max}$, where $0 < \rho < 1$ and $\lambda_{\max}$ denotes the average number of requests processed per second when the system continuously operates at full load. Varying ρ adjusts the request arrival pressure while allowing the system to reach steady state in the long run. We use $\rho \in \{0.90, 0.94, 0.97\}$.

5.2 Results Analysis

5.2.1 Reliability of Theoretical Queue-Length Tail Probabilities

We generate random requests at the three arrival rates described above and monitor the waiting queue length during inference serving. Each experiment consists of two stages. First, we perform a warm-up to exclude the initial queue buildup and allow server hardware operation to stabilize, so that measurements begin after the inference service has entered a stable operating regime. Specifically, warm-up lasts 2–8 hours and ends when the monitored mean queue length fluctuates around a stable level without abrupt changes. We then observe the queueing system for 24, 48, and 72 hours at $\rho = 0.90$, 0.94, and 0.97, respectively, recording the time spent at each queue length. We repeat this procedure for both batching strategies and compare the $P(Q > x)$ curves fitted to the empirical observations with the theoretical $P(Q > x)$ curves produced by our method. The results are shown in Figures 1 and 2.

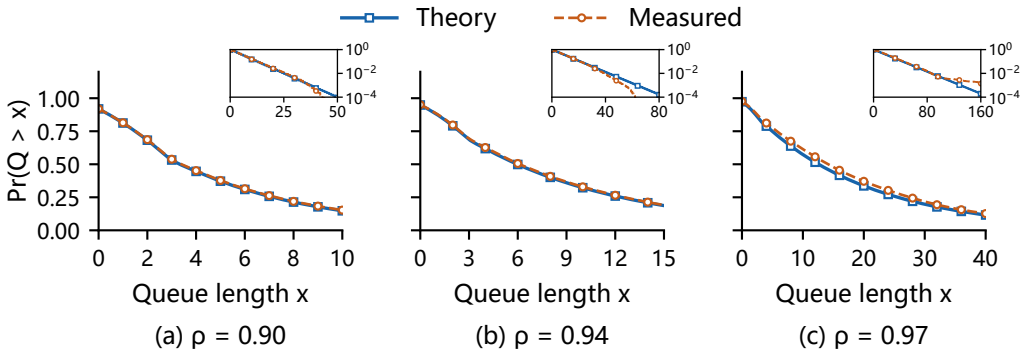

Figure 1: Comparison of theoretical and empirical queue-length tail probabilities under fixed batching.

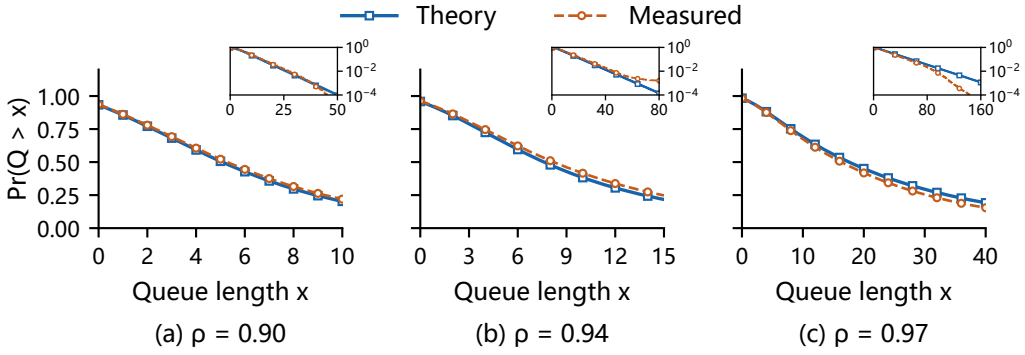

Figure 2: Comparison of theoretical and empirical queue-length tail probabilities under dynamic elastic batching.

Figures 1 and 2 compare theoretical and empirical queue-length tail probabilities in the high-tail-probability region under fixed and dynamic elastic batching, respectively, for different values of

ρ. The upper-right inset in each panel extends the comparison across the full displayed probability range using a logarithmic vertical scale corresponding to $\log_{10} \Pr(Q > x)$. The theoretical curves accurately estimate tail probabilities at thresholds associated with frequent queue-length exceedances. Larger discrepancies occur only over $x \in [40, 80]$ at $\rho = 0.94$ and $x \in [100, 160]$ at $\rho = 0.97$, where the corresponding tail probabilities are below 1%. These discrepancies therefore do not compromise providers' estimates in the high-probability region.

Table 1 further reports total variation distance (TV), maximum absolute error (MaxAE), mean absolute error (MAE), and relative error in mean queue length ($\mathrm{RE}_{\mathbb{E}[Q]}$). Although both batching strategies exhibit some discrepancies at $\rho = 0.97$, TV bounds the probability difference for any queue-length event to within 7 percentage points. Only dynamic elastic batching exhibits a mean-queue relative error slightly above 10%. Because these errors are primarily concentrated in the low-probability region, their impact on providers' estimates of the queue lengths most frequently observed during operation is limited.

Table 1: Errors in theoretical $P(Q > x)$ under different batching strategies

	ρ	TV	MaxAE	MAE	$\mathrm{RE}_{\mathbb{E}[Q]}$
Fixed	0.90	0.007	0.006	0.005	1.84%
	0.94	0.013	0.008	0.005	0.18%
	0.97	0.045	0.041	0.023	7.25%
Dynamic elastic	0.90	0.018	0.017	0.013	4.08%
	0.94	0.033	0.033	0.022	8.37%
	0.97	0.040	0.038	0.022	10.09%

5.2.2 Inference-Time Fitting Overhead

We fit H^l using randomly generated request batches of fixed size 4, and $H^{[b,d,l']}$ using randomly generated request batches whose sizes repeatedly cycle through 1–8 in order. Table 2 reports the training durations and fitting accuracy metrics, including the coefficient of determination (R^2), mean absolute percentage error (MAPE), 95th percentile of absolute percentage error (P95 APE), and absolute mean signed percentage error ($|\mathrm{MSPE}|$).

Table 2: Fitting error metrics for H^l and $H^{[b,d,l']}$ across training durations

	Time	R^2	MAPE	P95 APE	$\lvert\mathrm{MSPE}\rvert$
H^l	25.849s	0.980	2.461%	3.519%	2.292%
	35.597s	0.988	1.919%	2.858%	1.661%
	55.159s	0.993	1.338%	2.404%	0.947%
$H^{[b,d,l']}$	15.510s	0.998	0.730%	1.975%	0.330%
	62.797s	0.998	0.509%	1.368%	0.214%
	141.049s	0.999	0.470%	1.260%	0.158%

Both models improve rapidly during short service trials: overall and upper-percentile prediction errors and systematic bias decrease with training time. Within approximately 55s and 141s, H^l and $H^{[b,d,l']}$ attain $R^2 \geq 0.993$ and MAPE values of 1.338% and 0.470%, respectively. Together with the high-tail-probability results in Section 5.2.1, these measurements support brief calibration trials as a practical basis for backlog prediction.

6 Conclusion

We combine batch inference time models with queueing analysis to derive arbitrary-time steady-state queue-length tail probabilities for LLM serving under fixed and dynamic elastic batching. Real server queueing experiments demonstrate that the proposed method requires only modest model-fitting overhead to accurately estimate tail probabilities in the high-probability region while maintaining high overall accuracy across the full range of queue-length thresholds.

AI USE STATEMENT

In this work, we used generative AI tools to check grammar and spelling and to translate portions of the manuscript. We did not use generative AI tools to develop theoretical models, formulate mathematical claims, write proofs or their key steps, propose research methods, design experiments, generate synthetic datasets, implement methods, or interpret results. Qualitative and thematic data analysis is not applicable to this work.

We have manually reviewed all sentences in the manuscript that were revised or translated using AI tools. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Amey Agrawal, Nitin Kedia, Jayashree Mohan, Ashish Panwar, Nipun Kwatra, Bhargav S Gulavani, Ramachandran Ramjee, and Alexey Tumanov. Vidur: A large-scale simulation framework for llm inference. *Proceedings of Machine Learning and Systems*, 6:351–366, 2024a.

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 117–134, 2024b.

Ruicheng Ao, Gan Luo, David Simchi-Levi, and Xinshang Wang. Optimizing llm inference: Fluid-guided online scheduling with memory constraints. *arXiv preprint arXiv:2504.11320*, 2025.

Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. {ServerlessLLM}:{Low-Latency} serverless inference for large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 135–153, 2024.

Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. Parrot: Efficient serving of {LLM-based} applications with semantic variable. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 929–945, 2024.

Jiachen Liu, Jae-Won Chung, Zhiyu Wu, Fan Lai, Myungjin Lee, and Mosharaf Chowdhury. Andes: Defining and enhancing quality-of-experience in llm-based text streaming services. *arXiv preprint arXiv:2404.16283*, 2024.

Michael Luo, Xiaoxiang Shi, Colin Cai, Tianjun Zhang, Justin Wong, Yichuan Wang, Chi Wang, Yanping Huang, Zhifeng Chen, Joseph E Gonzalez, et al. Autellix: An efficient serving engine for llm agents as general programs. *arXiv preprint arXiv:2502.13965*, 2025.

Michael Mitzenmacher and Rana Shahout. Queueing, predictions, and large language models: Challenges and open problems. *Stochastic Systems*, 15(3):195–219, 2025.

Chengyi Nie, Nian Si, and Zijie Zhou. A queueing-theoretic framework for stability analysis of llm inference with kv cache memory constraints. *arXiv preprint arXiv:2605.04595*, 2026.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 118–132. IEEE, 2024.

Archit Patke, Dhemath Reddy, Saurabh Jha, Haoran Qiu, Christian Pinto, Chandra Narayanaswami, Zbigniew Kalbarczyk, and Ravishankar Iyer. Queue management for slo-oriented large language model serving. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*, pp. 18–35, 2024.

Yifan Qiao, Shu Anzai, Shan Yu, Haoran Ma, Shuo Yang, Yang Wang, Miryung Kim, Yongji Wu, Yang Zhou, Jiarong Xing, et al. Conserve: Fine-grained gpu harvesting for llm online and offline co-serving. *arXiv preprint arXiv:2410.01228*, 2024.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation—a {KVCache-centric} architecture for serving {LLM} chatbot. In *23rd USENIX conference on file and storage technologies (FAST 25)*, pp. 155–170, 2025.

Rana Shahout, Eran Malach, Chunwei Liu, Weifan Jiang, Minlan Yu, and Michael Mitzenmacher. Don't stop me now: Embedding based scheduling for llms. In *International Conference on Learning Representations*, volume 2025, pp. 63345–63368, 2025.

Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E Gonzalez, and Ion Stoica. Fairness in serving large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 965–988, 2024.

Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. Preble: Efficient distributed prompt scheduling for llm serving. In *International conference on learning representations*, volume 2025, pp. 37057–37082, 2025.

Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. Llumnix: Dynamic scheduling for large language model serving. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 173–191, 2024.

Bingyang Wu, Shengyu Liu, Yinmin Zhong, Peng Sun, Xuanzhe Liu, and Xin Jin. Loongserve: Efficiently serving long-context large language models with elastic sequence parallelism. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pp. 640–654, 2024.

Yuqing Yang, Lei Jiao, and Yuedong Xu. A queueing theoretic perspective on low-latency llm inference with variable token length. In *2024 22nd International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, pp. 273–280. IEEE, 2024.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 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, 2022.

Hengrui Zhang, August Ning, Rohan Baskar Prabhakar, and David Wentzlaff. Llmcompass: Enabling efficient hardware design for large language model inference. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 1080–1096. IEEE, 2024.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. {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, 2024.