FLOW-CONTROLLED SCHEDULING FOR LLM INFERENCE WITH PROVABLE STABILITY GUARANTEES

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) have been widely adopted due to their great performance across a wide range of applications. ChatGPT and Gemini now serve hundreds of millions of active users and handle billions of user requests per day, which puts optimizing LLM inference into the spotlight. A key challenge in LLM inference is that decode lengths are unknown. The memory usage for each request grows with generated tokens, which may lead to overflow and cause system instability. To address this concern, we propose a simple flow-control framework that controls the rate at which prompts join the active set. For known output lengths, we prove that, for every arrival rate that is strictly inside the stability region, there exists a periodic policy that can be stable. We derive a necessary condition that any stable system must satisfy and establish sufficient conditions under which our algorithm can provably achieve stability. Experiments show that, compared to commonly used strategies in practice, our approach achieves higher request throughput and lower average latency.

1 INTRODUCTION

Large language models (LLMs) have become a core component of modern AI applications, enabling a wide range of capabilities, from creative writing and translation to code generation and reasoning. Built on the transformer architecture (Vaswani et al., 2017), these models have rapidly scaled in both size and performance. This rapid growth has brought unprecedented serving demand: Chatterji et al. (2025) documented that, by July 2025, 700 million users were sending 18 billion messages per week on ChatGPT. Therefore, the efficiency and reliability of LLM inference (serving) are critical bottlenecks for both cost and user experience. Characterized by its autoregressive generation behavior, LLM inference systems process customer requests in two distinct stages: a *prefill* stage that processes the input prompt, followed by a *decode* stage that generates output tokens sequentially.

To improve GPU utilization, modern inference systems apply a batch policy, where requests are batched and handled simultaneously. Under this batching paradigm, a central challenge is the efficient management of the GPU memory, specifically the key-value (KV) cache. During inference, the system caches the keys and values computed from the processed prefill and generated decode tokens to generate subsequent tokens. The memory footprint of a request grows accordingly during token generation and is released only when the request is completed, introducing dynamic memory consumption. In addition, real-world request arrivals are highly stochastic, with variable input and output lengths. Therefore, the aggregate memory usage under the batching policy may exhibit significant fluctuation. When memory usage approaches the limit, the system may be forced to evict requests and repeat computation, which degrades latency and user experience.

To mitigate these challenges, researchers in computer systems have proposed inference engines with continuous batching mechanisms, such as Orca and vLLM (Yu et al., 2022; Kwon et al., 2023). Subsequent work employed chunked prefills to reduce interference between prefill and decoding (Agrawal et al., 2024b). These developments have also motivated queueing-theoretic studies of the loads that inference systems can sustain (Mitzenmacher & Shahout, 2025).

We study LLM inference through the lens of queueing theory. We model the inference engine as a stochastic service system in which GPU memory is the main constraint. Our key insight is to focus on request admission, i.e., when a request enters the active set and begins prefilling. When many requests are processed simultaneously, their memory usage tends to increase simultaneously, leading

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.

to concentrated memory peaks. Building on this insight, we control the number of requests activated in each time slot through activation budgets that repeat periodically. This raises a natural question: can such a simple policy achieve good performance?

For known output lengths, we show that periodic activation budgets can stabilize every request workload in the interior of the region achievable by any scheduling policies. Building on this result, we further develop methods to select the budgets for a particular request workload. Our contributions are as follows.

1. We show that, under a hard KV cache constraint, the closure of the stability region (request workload that can be stable under a particular scheduling policy) can be characterized by periodic flow-controlled policies.
2. We derive an integer optimization formulation for selecting the budgets for a periodic flow-controlled policy. We also show that such a policy can stabilize the system under a particular load condition. These results offer practical implications for LLM inference.
3. For unknown output lengths, we propose a periodic algorithm with LIFO eviction and bound its overflow probability over a finite horizon.
4. We conduct comprehensive experiments using both synthetic and real-world datasets, where our scheduling algorithm outperforms benchmark algorithms in both request throughput and latency.

1.1 Related Work

LLM inference scheduling. Modern inference engines improve throughput and tail latency under tight GPU memory constraints. Early systems such as FasterTransformer (NVIDIA, 2024) allow request-level scheduling while prioritizing the decoding stage. Recently, continuous batching allows requests to enter and leave a running batch (Yu et al., 2022), and PagedAttention improves the allocation of the KV cache (Kwon et al., 2023). On the theoretical side, recent studies analyze online scheduling, throughput, and robustness under different memory and processing models (Jaillet et al., 2025; Dai et al., 2025; Wang et al., 2025). Moreover, Ao et al. (2025) use a fluid model to design WAIT and Nested WAIT, which enjoy asymptotic performance guarantees under threshold and memory conditions. Ao et al. (2026) study service-induced memory congestion and rate-limited admission. Nie et al. (2026) analyze the stability of work-conserving policies in a model where KV caches can be swapped to the CPU at no cost. In contrast, we concern the entire class of scheduling policies. Our analysis exploits a feature specific to LLM inference. Since the KV footprint of a request grows as it decodes, any successful execution can be transformed into continuous decoding without increasing the memory usage in any slot. By connecting general policies to periodic flow-controlled policies, we show that the latter can stabilize every request workload in the interior of the stability region.

Queueing Theory. The queueing community has established an immense body of works on studying the behavior of stochastic service systems under resource constraints. This line of literature characterizes when such systems can operate stably and analyzes the system performance (e.g., throughput or delay) with multiple request types (Chen & Yao, 2001; Hu et al., 2022; Chen et al., 2025a). These results are translated and applied into broad application domains, including service systems (Guo et al., 2023; Zhong et al., 2025), healthcare (Hu et al., 2022), and cloud systems (Lu et al., 2011).

2 Model and Problem Formulation

We study an online LLM inference problem on a single computational worker (e.g., GPU) with a limited key-value (KV) cache budget $M \in \mathbb{N}$. Time is slotted and indexed by $t \in \mathbb{N}$, and the system starts empty. Each request p_i is characterized by (l_i, o_i, t_i), where $l_i \in \mathbb{N}$ is the input length, $o_i \in \mathbb{N}$ is the output length, and t_i is its arrival slot. The decision-maker observes a request only when it arrives. Upon arrival at time t_i, the prompt length l_i is revealed, while the output length o_i may not be known in advance. We first study known output lengths; Section 5 considers unknown lengths.

At the beginning of each slot t, new requests join a waiting queue, and the decision-maker chooses which requests to activate, placing their KV cache in memory. We assume no KV cache is stored outside the computational worker. In our paper, we assume that the prefill stage takes zero time slots. The decision-maker then selects active requests to decode, each generating one token in that slot. Once activated, the request joins the active set and is admitted to the KV cache. A request with j generated tokens occupies $l_i + j$ units of memory. It remains in the active set until it either completes generating all output tokens or is evicted. Once a request is evicted, its generated tokens and KV cache are completely reset to zero and will return to the waiting queue.

Let $\mathcal{P}_t$ denote the active set right after decoding at time t, before completed requests release their memory. Let $j_i(t)$ denote the number of tokens generated by request i at t. Because the overall KV cache usage cannot exceed the limit M, we have

$$U_t = \sum_{p_i \in \mathcal{P}_t} (l_i + j_i(t)) \leq M.$$

In particular, when request i generates its last token, its memory footprint reaches its maximum $l_i + o_i$. The KV cache is then released at the end of that slot. We call a policy feasible if its KV cache usage never exceeds M.

In this paper, we classify requests into m classes according to their input and output lengths. or each class $k \in [m] = \{1, 2 \ldots, m\}$, requests share the same prefill length l_k and output length o_k. Let $N_{t,k}$ denote the number of type k arrivals in slot t, and let $\lambda_{t,k}$ denote its mean. Here we impose the following assumption.

Assumption 2.1. $(N_{t,1}, \ldots, N_{t,m})$ are independent and identically distributed over time, with $\mathbb{E}[N_{t,k}] = \lambda_{t,k} := \lambda_k > 0$ and $\mathbb{E}[N_{t,k}^2] < \infty$. Arrivals of different classes in the same slot may be dependent.

This assumption holds, for example, when each class has independent Poisson arrivals. Let $X_{t,k}$ count all type k requests that have arrived but remain unfinished at the end of slot t, including all waiting, active, and restarted requests.

Definition 2.2 (Stable system)**.** A policy stabilizes the system if

$$\limsup_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{k=1}^{m} \mathbb{E}[X_{t,k}] < \infty.$$

Intuitively, stability means that the expected backlog remains finite on average over time. For each class k, we define the *cumulative memory workload* of one request by

$$w_k = \sum_{q=1}^{o_k} (l_k + q) = l_k o_k + \frac{o_k(o_k + 1)}{2}. \tag{1}$$

Then we obtain the following proposition, which characterize a necessary condition for our flow-controlled LLM inference scheduling policy to achieve stability.

Proposition 2.3 (Necessary condition)**.** *Given an arrival-rate vector $(\lambda_1, \ldots, \lambda_m)$, if there exists a scheduling policy that stabilizes the system, we have $\sum_k \lambda_k w_k \leq M$.*

Proposition 2.3 characterizes a necessary condition that any stable system must satisfy. In Equation (1), w_k represents the minimum cumulative KV cache usage required to complete a type k request, summed over all of its decoding slots. When the inequality does not hold, the request arrival load exceeds the service capacity, meaning that *no* LLM inference scheduling algorithms can keep the system stable. This proposition bounds the system's ability to complete requests from above, and the proof is provided in Appendix C.

3 Periodic Flow Control

The core principle of our flow-control design is to control the activation rate. At each time step, we impose an activation budget and allow a limited number of requests to join the active set. This mechanism prevents transient surges in memory usage, which could inflate KV cache usage and destabilize the system. In this section, we allow these budgets to repeat over a period of P slots. A budget that is not used in a slot does not carry over to later slots.

Algorithm 1 Periodic Flow-Controlled LLM Inference

Input: period P, budgets $(b_{r,k})$, arrival stream.
Initialize class waiting queues $\mathcal{Q}_k$ and active set $\mathcal{P}$.
for $t = 1, 2, \ldots$ **do**
  Add new arrivals to their class queues; set $r \leftarrow \langle t \rangle_P$.
  for $k = 1, \ldots, m$ **do**
    Activate the first $\min\{b_{r,k}, |\mathcal{Q}_k|\}$ requests in FIFO order.
    Remove them from $\mathcal{Q}_k$ and add them to $\mathcal{P}$.
  end for
  Decode every request in $\mathcal{P}$ by one token.
  Remove completed requests from $\mathcal{P}$ at the end of the slot.
end for

3.1 Memory Safety and Stability

We begin with a simplified setting, where the decision-maker observes both the prefill and decode lengths of a request upon arrival. For each phase $r \in \{1, \ldots, P\}$ and class k, let $b_{r,k} \in \mathbb{Z}_+$ denote the activation budget, i.e., the maximum number of type k requests that can be activated in phase r. We write $\langle t \rangle_P = 1 + ((t-1) \bmod P)$ for the phase of slot t, and denote the total budget of class k in a period and its average activation budget per slot by

$$B_k = \sum_{r=1}^{P} b_{r,k}, \quad \mu_k = \frac{B_k}{P} = \frac{1}{P} \sum_{r=1}^{P} b_{r,k}. \tag{2}$$

A type k request activated in slot s and decoded continuously occupies $l_k + q$ units of memory in slot $s + q - 1$, for $q = 1, \ldots, o_k$. Therefore, the memory usage in phase r is bounded by $G_r(b)$, defined as

$$G_r(b) = \sum_{k=1}^{m} \sum_{q=1}^{o_k} (l_k + q) b_{\langle r-q+1 \rangle_P, k}. \tag{3}$$

When $o_k > P$, this sum also counts requests activated in earlier periods.

For notational convenience, let $Q_{t-1,k}$ denote the type k waiting queue at the end of slot $t-1$. Then we are able to obtain the following result.

Theorem 3.1 (Stability condition)**.** *Under Assumption 2.1, suppose we have the following inequality:*

$$G_r(b) \le M, \quad \mu_k > \lambda_k. \tag{4}$$

Then Algorithm 1 is eviction-free and stable.

The design principle of Algorithm 1 is to control the rate at which requests join the active set. In slot t, the algorithm activates at most $b_{\langle t \rangle_P, k}$ requests of class k. By Inequality (4), the memory usage never exceeds the limit and no eviction is needed. Moreover, Inequality (4) ensures that the average activation budget of each class exceeds its arrival rate. The detailed proof is provided in Appendix B.

When $P = 1$, Theorem 3.1 reduces to a constant activation budget with integers $b_k > \lambda_k$ and $\sum_k b_k w_k \le M$. Longer periods allow fractional average activation budgets. For example, with $(l, o, M) = (1, 2, 3)$, the period-two budget $(1, 0)$ stabilizes the system for every $\lambda < 1/2$, while no positive constant integer budget satisfies the memory bound.

3.2 The Stability Region of General Policies

We next consider how much of the stability region periodic schedules can cover, compared with general policies.

Definition 3.2 (Stability region)**.** The stability region Λ is the set of arrival-rate vectors λ such that some feasible causal policy stabilizes the system under some arrival process with mean λ satisfying Assumption 2.1. Such a policy may activate, pause, and restart requests.

We then establish the following theorem.

Theorem 3.3. *For known output lengths, every arrival-rate vector in the interior of Λ can be stabilized by a finite periodic flow-controlled policy without eviction.*

Proof sketch. The proof rests on two observations. First, because the KV footprint of a request grows as it decodes, moving its tokens as late as possible while keeping its completion slot never increases the memory usage in any slot. Consider any feasible schedule up to slot T, and keep only the last run of each request completed by T. For a class-k request completing in slot c, we move its q-th token to slot $c - o_k + q$. Since a request generates at most one token per slot, the original run has generated at least q tokens by that slot and remains in memory until c, so the memory usage indeed does not increase in any slot. Each request is still activated no earlier than its arrival and now decodes continuously until its original completion slot. As all retained requests start and finish within slots $1, \ldots, T$, the resulting activation counts can be repeated periodically.

Second, we approximate the rates of a stable policy by such periodic budgets. Under strong stability, the expected completion counts per slot approach the arrival rates along a sequence of horizons. For each horizon, the memory constraint permits only finitely many budgets, so these rates are a convex combination of the average budgets. Approximating the weights by rational numbers and concatenating copies of the corresponding schedules yields deterministic periodic budgets with arbitrarily close average rates. For a target load in the interior of Λ, we apply this construction to a slightly larger stabilizable load and choose the approximation so that every class with positive arrivals receives an average budget above its arrival rate. Theorem 3.1 then gives strong stability for every arrival process satisfying Assumption 2.1 with the target rates. The full characterization and proof are provided in Appendix C.

4 Selecting Periodic Activation Budgets

In this section, we select the activation budgets using an integer optimization formulation and a simpler construction for selecting these budgets.

4.1 A Fixed-Period Integer Formulation

By Theorem 3.1, a budget vector b yields a stable policy if every phase uses at most M memory and every class receives, on average, more activations than it has arrivals. For a given period P, we search for such a vector by solving the following optimization problem, as follows:

$$\begin{aligned} \max_{b,\eta} \quad & \eta \\ \text{subject to} \quad & G_r(b) \le M && r \in [P], \\ & P^{-1} \sum_{r=1}^{P} b_{r,k} \ge (1+\eta)\lambda_k && k \in [m], \\ & b_{r,k} \in \mathbb{Z}_+, \quad \eta \in \mathbb{R}. \end{aligned} \tag{5}$$

Here, the first constraint bounds the memory usage in every phase, and the second requires μ_k to exceed λ_k by $\eta\lambda_k$, which the objective maximizes.

Proposition 4.1. *The optimization problem (5) attains a finite optimum. This optimum is positive if and only if there exists a period-P schedule whose average activation budget exceeds the arrival rate of every request class.*

Maximizing η gives every class the same relative slack: each class can be activated at least $1 + \eta$ times as fast as it arrives. By Theorem 3.1, a positive optimum therefore yields a stable policy. The resulting policy tends to have short latency, because a larger slack lets the waiting queues drain quickly after bursts of arrivals. However, it need not minimize the average or tail latency among all flow-controlled policies.

4.2 Evenly Spaced Activation Budgets

In practice, choosing a suitable period and budgets can be complicated. We now consider a simple alternative that spreads the budget of each class evenly over time. Specifically, for each class k, we

choose a rational target rate $\beta_k > \lambda_k$ as its average budget μ_k, and a phase $\varphi_k \in [0, 1)$, and set

$$b_{t,k} = \lfloor \beta_k t + \varphi_k \rfloor - \lfloor \beta_k (t-1) + \varphi_k \rfloor, \tag{6}$$

which counts the integers that $\beta_k t + \varphi_k$ passes in slot t. Class k thus receives activation budgets close to β_k for every slot. For convenience, one can simply take $\varphi_k = 0$. Choosing a common period P with $P\beta_k \in \mathbb{Z}_+$ makes the budgets periodic.

Theorem 4.2. *The budgets in (6) satisfy*

$$G_r(b) < \sum_k \beta_k w_k + \sum_k (l_k + o_k) \quad (r \in [P]). \tag{7}$$

Consequently, if

$$\sum_k \beta_k w_k + \sum_k (l_k + o_k) \le M, \tag{8}$$

then these budgets yield an eviction-free and stable periodic policy.

Theorem 4.2 implies that every request workload satisfying $\sum_k \lambda_k w_k < M - \sum_k (l_k + o_k)$ can be stabilized by evenly spaced budgets. It is worth noting that this is not a necessary condition for the system to be stable.

Even spacing also keeps the waiting queues short. Intuitively, this strategy smooths the memory usage: when the system admits many prompts in a short interval, it may follow a burst of simultaneous completions, leading to a drop of memory and causing volatility. In contrast, controlling the activation rate stabilizes the inference system, yields a more predictable KV cache usage trajectory, and improves stability.

5 Unknown Output Lengths

We now consider a more realistic setting, where the output length is unknown upon request arrival. Upon arrival of a request p_i, its input length l_i is observed while the output length o_i is unknown. We now group requests into classes $k \in [m]$ according to their observed input lengths, where class k has input length l_k. Conditional on the input classes, the output lengths are independent across requests, as well as independent of the arrival counts. The output lengths of class k follow a distribution supported on a finite set $[C_k] = \{1, \ldots, C_k\}$. We assume that the arrival vectors satisfy Assumption 2.1 and that $l_k + C_k \le M$. For each class k, we write

$$p_{k,q} = \mathbb{P}(O_i \ge q \mid i \text{ belongs to class } k), \quad q = 1, \ldots, C_k. \tag{9}$$

Intuitively, $p_{k,q}$ is the probability that a type k request is still decoding at its q-th step.

Without the output lengths, the memory usage of a scheduling policy is no longer deterministic, indicating that overflow events may happen at a time. We therefore add an eviction rule to handle such overflows: When the next decoding step would exceed the memory limit, we evict active requests following the last-in, first-out (LIFO) mechanism. Eviction takes place before decoding, thus the actual memory usage never exceeds M. Intuitively, LIFO eviction prioritizes keeping requests that have been decoding the longest, which have already consumed significant KV cache resources during their decoding. Thus, evicting them would waste substantial computational effort. This makes LIFO a natural choice for balancing overflow recovery with resource efficiency. In what follows, we formally propose our policy Algorithm 2 for the scenario where the output length is unknown. Here, j_i denotes the number of tokens that request i has generated before decoding in slot t.

Algorithm 2 Periodic Flow Control with Unknown Output Lengths

Require: Memory M; period P; budgets $b_{r,k}$; empty FIFO queues $\mathcal{Q}_k$ and active set.
1: **for** each slot $t = 1, 2, \ldots$ **do**
2: Add new arrivals to their input-class FIFO queues.
3: Remove $\min\{b_{\langle t \rangle_P,k}, |\mathcal{Q}_k|\}$ FIFO requests from each class queue $\mathcal{Q}_k$.
4: Tentatively add them to the active set and compute $\widetilde{U}_t = \sum_{i \text{ active}} (l_i + j_i + 1)$.
5: **while** $\widetilde{U}_t > M$ **do**
6: Evict the most recently activated request (largest ID first for ties).
7: Reset its decoded progress, return it to the tail of its class queue, and update $\widetilde{U}_t$.
8: **end while**
9: Commit the remaining activations, decode one token per active request, and release completions.
10: **end for**

Now we consider the probability of overflow under the above algorithm. Let τ denote the time of the first overflow in Algorithm 2. For $\theta > 0$, we define

$$\Psi_r(\theta; b) = \sum_{k=1}^{m} \sum_{q=1}^{C_k} b_{\langle r-q+1 \rangle_P,k} \log\left(1 - p_{k,q} + p_{k,q} e^{\theta(l_k+q)}\right), \tag{10}$$

which is the log moment-generating function.

Theorem 5.1. *For any deterministic periodic budgets and any $T \in \mathbb{N}$, we have*

$$\mathbb{P}(\tau \le T) \le \min\left\{1,\ T \max_{r \in [P]} \inf_{\theta > 0} e^{-\theta(M+1)+\Psi_r(\theta;b)}\right\}. \tag{11}$$

Theorem 5.1 bounds the probability that an overflow occurs within the first T slots. The proof is provided in Appendix F, where we consider the memory usage before the first eviction using a virtual system in which all activation budgets are fully used.

The theorem also provides clues on how to select the budgets b. With unknown output lengths, the memory constraints $G_r(b) \le M$ in program (5) can no longer be guaranteed in advance. Instead, we can fix the parameter $\theta > 0$ of the bound and replace the memory constraints by

$$\Psi_r(\theta; b) \le \theta(M+1) + \log(\delta/T), \quad r \in [P], \tag{12}$$

where $\delta \in (0, 1)$ is a risk tolerance. Every feasible activation budget b that satisfies these constraints guarantees $\mathbb{P}(\tau \le T) \le \delta$, Since $\Psi_r(\theta; b)$ is linear in b, the result is still an integer program, and a positive slack η ensures that the average budgets exceed the arrival rates. In practice, one can try several values of P and θ and keep the best schedule.

So far, we have assumed that the output-length distributions are known. In practice, they can be estimated from historical requests. If we use upper confidence bounds on $p_{k,q}$ in (12), we can still bound the overflow probability within T slots from above (see Appendix F). Alternatively, if the budgets satisfy

$$\sum_{k} \sum_{q=1}^{C_k} (l_k + q)\, b_{\langle r-q+1 \rangle_P,k} \le M, \quad \forall r \in [P],$$

that is, condition (4) with each output length replaced by its bound C_k, then no eviction ever occurs, and the policy is strongly stable whenever $\mu_k > \lambda_k$ for every class with arrivals (see Appendix E).

6 Numerical Simulations

In this section, we evaluate our flow-controlled algorithms through numerical simulations. We measure its performance using four metrics: the average and the tail end-to-end latency, i.e., the time from arrival to completion, and the request and workload throughput, i.e., the number and the total workload $\sum_i \big(l_i o_i + o_i(o_i+1)/2\big)$ of completed requests per unit time. We consider two simulation settings. In Section 6.1, we conduct simulations on synthetic data and compare the performance of our algorithm against some baseline heuristic scheduling algorithms. In Section 6.2, we conduct simulations on a large-scale real-world dataset.

6.1 Synthetic Data Simulations

We first evaluate the performance of our algorithm on synthetic datasets. We consider a simulation setting where the simulated prompts can be classified into $m = 3$ types according to the input and output lengths:

$$(l_1, o_1) = (10, 20), \quad (l_2, o_2) = (10, 40), \quad (l_3, o_3) = (10, 60).$$

These classes share the same input length but differ in output length. Each type of request arrives as an independent Poisson process with the same intensity λ. We allow the time horizon T to vary with $T \in \{100, 200, \ldots, 1000\}$, and let $\lambda \in \{3, 4, 5\}$. For brevity, we report the results for $\lambda \in \{3, 4\}$ in Appendix G. We simulate an inference system with a hard KV cache capacity M. We set $M = 16492$, which matches the deployment scale for Llama2-70B on two A100 GPUs. In this scenario, we set $(b_1, b_2, b_3) = (4, 4, 4)$ so that no eviction will happen. This choice makes sure that the memory usage will never exceed the limit.

Known output lengths. To begin with, we assume that the request type is known upon request arrival. We compare our algorithm against four baselines: (1) *α-protection greedy* algorithm (Jaillet et al., 2025), which activates a new request only if the memory usage stays within $(1 - \alpha)M$ and evicts all active requests upon overflow, with $\alpha = 0.72$ chosen by a grid search; (2) *MC benchmark*, an FCFS-style algorithm, in which the algorithm only activates requests if the future memory usage remains within the memory limit; (3) *MC-SF* (Jaillet et al., 2025), which prioritizes adding requests with smaller decoding lengths to allow more requests to be packed into batches. The algorithm activates requests when the future memory usage is within the limit; and (4) *hindsight optimal* value of each metric, which we compute on one sample path by solving a separate integer program with Gurobi.

The results are reported in the first row of Figure 1, which reveal that our method outperforms the α-protection greedy, MC, and MC-SF algorithms across all metrics and achieves performance close to the hindsight optimal. This demonstrates the value of flow control. Similar benefits also appears at $\lambda = 4$. In Figure 1, optimal denotes the hindsight optimum, and each curve is the average results over several runs.

Unknown output lengths. Next, we consider the setting in which the output length of each request is unknown upon arrival. In this setting, the MC-SF algorithm becomes less applicable because we do not have access to the decoding length of each request. To this end, we benchmark against the Amin algorithm proposed by Chen et al. (2025b). Concretely, the Amin algorithm treats each output length as the shortest possible value, greedily activates requests, and upon overflow evicts requests in ascending order of their predicted output lengths while updating these predictions. We also adapt the MC benchmark by treating each output length as the longest possible value.

The results are reported in the second row of Figure 1, showing that our method again achieves near-optimal performance and outperforms the benchmarks across all metrics. In particular, at $T = 1000$, its average latency is 189 slots, compared with 277 for Amin and 383 for MC, and its throughput is only about 7% below the hindsight optimum.

6.2 Real-World Data Simulations

In this section, we evaluate the performance of our algorithm on a real-world dataset from Zheng et al. (2023a), which contains over 210000 prompts. We randomly select 50000 prompts from this dataset and treat each as an inference request: the prompt serves as an input, and the response serves as an output. We measure token lengths in tokens of the Llama-2 tokenizer and retain the 42,727 requests whose lengths are both at most 500. The requests are simulated to arrive according to a Poisson process with intensity λ. We allow λ to vary with $\lambda \in \{10, 50\}$ (requests per second) to simulate both high- and low-demand settings. For brevity, we report the results for $\lambda = 50$ in Appendix G. The output length is unknown upon arrival. Moreover, in our model, we discretize the processing time of a batch into a time period. However, in practical LLM serving, it depends on the batch composition (e.g., the number of prefill and decode tokens). Therefore, in our numerical results, we simulate a continuous-time serving system and adopt Microsoft Vidur (Agrawal et al., 2024a) to calculate the processing time of each batch for Llama-2-70B on two A100 GPUs, with a KV cache of 16492 tokens and at most 512 requests per batch. Our algorithm follows Algorithm 2 with the constant

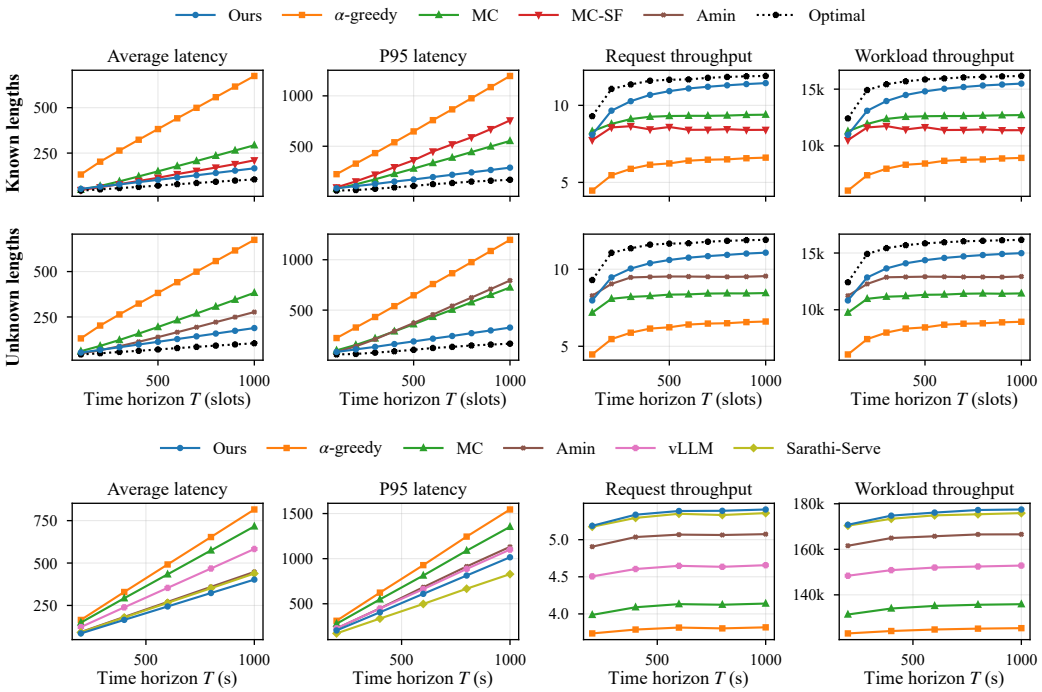

Figure 1: Performance metrics across scheduling algorithms. (Top) Synthetic data with $\lambda = 5$ when the output length is known upon arrival. (Middle) Synthetic data with $\lambda = 5$ when the output length is unknown upon arrival. (Bottom) Real-world data with $\lambda = 10$ requests per second. The performance metrics (from left to right) are average latency, 95% latency, request throughput, and workload throughput.

budget $b = 1$, i.e., it activates at most one request per iteration. In addition to the former benchmarks (i.e., the α-protection greedy ($\alpha = 0.5$), MC, and Amin algorithms), we further compare with vLLM (Kwon et al., 2023) and Sarathi (Agrawal et al., 2024b) as implemented in Vidur.

As shown in the last row of Figure 1 and Appendix G, our algorithm achieves the lowest average latency and the highest request and workload throughput at both arrival rates. At $T = 1000$, our average latency is 7–8% lower than that of Sarathi and 21–31% lower than that of vLLM. Meanwhile, our throughput is 16% higher than that of vLLM. However, Sarathi achieves a lower 95% latency, by 18% at $\lambda = 10$ and 1% at $\lambda = 50$.

7 Discussion

In this paper, we study the LLM serving system with stochastic and heterogeneous request arrivals. Motivated by the results in queueing theory, we propose an easy-to-implement periodic flow-controlled scheduling algorithm, which limits the rate that requests are activated. For known output lengths, these policies stabilize every request workload in the interior of the stable region, and we provide constructions for selecting them from target arrival rates. For unknown output lengths, we combine periodic activation with LIFO eviction and select the budgets using a finite-horizon bound on the overflow probability. Several directions merit future work. First, incorporating the output length predictions from Zheng et al. (2023b) with LLM serving can potentially improve the performance of the serving system and allows for more informative decision-making. Second, extending our framework to multi-turn interactions, where consecutive prompts share context and prompt caches, is an intriguing direction.

AI Use Statement

Large language models were used to assist with literature discovery, manuscript drafting and editing, mathematical derivations and proof checking, and implementation of simulation code. The authors are responsible for the correctness and attribution of the final manuscript and code.

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. In *Proceedings of Machine Learning and Systems*, volume 6, pp. 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. USENIX Association, 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.

Ruicheng Ao, Jing Dong, Gan Luo, and David Simchi-Levi. Service-induced congestion in memory-constrained LLM serving. arXiv preprint arXiv:2606.15555, 2026.

Aaron Chatterji, Thomas Cunningham, David J. Deming, Zoe Hitzig, Christopher Ong, Carl Yan Shan, and Kevin Wadman. How people use ChatGPT. Working Paper 34255, National Bureau of Economic Research, September 2025.

Hong Chen and David D Yao. *Fundamentals of queueing networks: Performance, asymptotics, and optimization*, volume 46. Springer Science & Business Media, 2001.

Jinsheng Chen, Jing Dong, and Pengyi Shi. Optimal routing under demand surges: The value of future arrival rates. *Operations Research*, 73(1):510–542, 2025a.

Zixi Chen, Yinyu Ye, and Zijie Zhou. Adaptively robust LLM inference optimization under prediction uncertainty. arXiv preprint arXiv:2508.14544, 2025b.

J. G. Dai, Tianze Deng, Yueying Li, and Tianyi Peng. Throughput-optimal scheduling algorithms for LLM inference and AI agents. arXiv preprint arXiv:2504.07347, 2025.

Pengfei Guo, Moshe Haviv, Zhenwei Luo, and Yulan Wang. Signaling service quality through queue disclosure. *Manufacturing & Service Operations Management*, 25(2):543–562, 2023.

Yue Hu, Carri W Chan, and Jing Dong. Optimal scheduling of proactive service with customer deterioration and improvement. *Management Science*, 68(4):2533–2578, 2022.

Patrick Jaillet, Jiashuo Jiang, Konstantina Mellou, Marco Molinaro, Chara Podimata, and Zijie Zhou. Online scheduling for LLM inference with KV cache constraints. arXiv preprint arXiv:2502.07115, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626. Association for Computing Machinery, 2023.

Yi Lu, Qiaomin Xie, Gabriel Kliot, Alan Geller, James R Larus, and Albert Greenberg. Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services. *Performance Evaluation*, 68(11):1056–1071, 2011.

Pascal Massart. The tight constant in the Dvoretzky–Kiefer–Wolfowitz inequality. *The Annals of Probability*, 18(3):1269–1283, 1990.

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. In *International Conference on Machine Learning*, 2026.

NVIDIA. Fastertransformer, 2024. URL https://github.com/NVIDIA/FasterTransformer.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Meixuan Wang, Yinyu Ye, and Zijie Zhou. LLM serving optimization with variable prefill and decode lengths. arXiv preprint arXiv:2508.06133, 2025.

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. USENIX Association, 2022.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric P. Xing, Joseph E. Gonzalez, Ion Stoica, and Hao Zhang. LMSYS-Chat-1M: A large-scale real-world LLM conversation dataset. arXiv preprint arXiv:2309.11998, 2023a.

Zangwei Zheng, Xiaozhe Ren, Fuzhao Xue, Yang Luo, Xin Jiang, and Yang You. Response length perception and sequence scheduling: An LLM-empowered LLM inference pipeline. In *Advances in Neural Information Processing Systems*, volume 36, pp. 65517–65530, 2023b.

Yueyang Zhong, John R Birge, and Amy R Ward. Learning to schedule in multiclass many-server queues with abandonment. *Operations Research*, 73(6):3085–3103, 2025.

A QUEUE IDENTITIES AND STABILITY

We establish the queue identities and drift bounds used in the stability proofs. Throughout the appendix, we refer to each unit of an activation budget as an activation opportunity. For notational convenience, let the number of type k requests activated in slot t is

$$a_{t,k} = \min\{b_{\langle t \rangle_P,k}, Q_{t-1,k} + N_{t,k}\}. \tag{13}$$

Lemma A.1 (Persistent-budget recursion)**.** *Fix one class, a deterministic budget sequence $c_t \in \mathbb{Z}_+$, and an arrival sequence $n_t \in \mathbb{Z}_+$. Let $Q_0 = 0$. In slot t, after adding n_t, activate $a_t = \min\{c_t, Q_{t-1} + n_t\}$. Then*

$$Q_t = (Q_{t-1} + n_t - c_t)^+. \tag{14}$$

For every $t \geq 0$,

$$Q_t = \max_{0 \leq u \leq t} \left\{ \sum_{v=u+1}^{t} (n_v - c_v) \right\}^+, \tag{15}$$

where the term $u = t$ equals zero.

Proof. The number left after activation is $Q_{t-1} + n_t - \min\{c_t, Q_{t-1} + n_t\}$, which is (14). We prove (15) by induction. It holds at $t = 0$. If it holds at $t - 1$, then

$$\begin{aligned} Q_t &= \left(\max_{0 \leq u \leq t-1} \left\{ \sum_{v=u+1}^{t-1} (n_v - c_v) \right\}^+ + n_t - c_t \right)^+ \\ &= \max\left\{ 0, \max_{0 \leq u \leq t-1} \sum_{v=u+1}^{t} (n_v - c_v) \right\}, \end{aligned}$$

which is exactly the claimed formula. □

Lemma A.2 (A frame inequality)**.** *For nonnegative q, arrivals $n_1, \ldots, n_P$, and budgets $c_1, \ldots, c_P$, let Q_P be obtained by iterating (14) from $Q_0 = q$. Put $A = \sum_{r=1}^{P} n_r$ and $B = \sum_{r=1}^{P} c_r$. Then*

$$Q_P \leq (q - B)^+ + A. \tag{16}$$

Consequently,

$$Q_P^2 - q^2 \leq A^2 + B^2 + 2q(A - B). \tag{17}$$

Proof. We first prove (16) by induction on P. For $P = 1$,

$$(q + n_1 - c_1)^+ \leq (q - c_1)^+ + n_1.$$

Assume the result for $P - 1$. Monotonicity of $x \mapsto (x + n_P - c_P)^+$ gives

$$\begin{aligned} Q_P &\leq \big((q - B + c_P)^+ + A - n_P + n_P - c_P\big)^+ \\ &\leq \big((q - B + c_P)^+ - c_P\big)^+ + A \\ &= (q - B)^+ + A. \end{aligned}$$

The second line uses $(x + y - c)^+ \leq (x - c)^+ + y$ for $x, y, c \geq 0$, and the last identity uses $((x - a)^+ - b)^+ = (x - a - b)^+$ for $a, b \geq 0$.

For (17), use

$$((q - B)^+ + A)^2 \leq q^2 + A^2 + B^2 + 2q(A - B).$$

If $q \geq B$, this follows by expanding $(q - B + A)^2$ and dropping $-2AB$. If $q < B$, the left side is A^2, while the right side minus A^2 equals $(B - q)^2 + 2qA \geq 0$. □

Lemma A.3 (Stability from frame drift)**.** *Let $Z_n = (Z_{n,1}, \ldots, Z_{n,m}) \in \mathbb{Z}_+^m$ and suppose*

$$Z_{n+1,k} \leq (Z_{n,k} - B_k)^+ + A_{n,k}, \tag{18}$$

where $B_k \in \mathbb{Z}_+$, the vectors $A_n \in \mathbb{Z}_+^m$ are i.i.d., independent of Z_n, $\mathbb{E}[A_{n,k}] = \alpha_k$, $\mathbb{E}[A_{n,k}^2] < \infty$, and $\mathbb{E}[\sum_k Z_{0,k}^2] < \infty$. Let $\mathcal{K} := \{k : \alpha_k < B_k\}$. For every $k \notin \mathcal{K}$, assume $B_k = \alpha_k = 0$ and $Z_{0,k} = 0$ almost surely. If $\mathcal{K}$ is nonempty, then

$$\limsup_{R\to\infty} \frac{1}{R} \sum_{n=0}^{R-1} \sum_{k=1}^{m} \mathbb{E}[Z_{n,k}] \le \frac{\sum_{k\in\mathcal{K}} (\mathbb{E}[A_{0,k}^2] + B_k^2)}{2 \min_{k\in\mathcal{K}} (B_k - \alpha_k)}. \tag{19}$$

If $\mathcal{K}$ is empty, then $Z_n = 0$ almost surely for every n. Suppose, in addition, that a slot-level process Q_t satisfies $Q_{nP,k} = Z_{n,k}$ and, for some nonnegative $H_{n,k}$,

$$Q_{nP+j,k} \le Z_{n,k} + H_{n,k}, \quad 0 \le j < P, \quad \sup_n \mathbb{E}[H_{n,k}] < \infty.$$

Then frame-boundary stability implies slot-level stability.

Proof. For $k \notin \mathcal{K}$, nonnegativity, $A_{n,k} = 0$ almost surely, and (18) imply inductively that $Z_{n,k} = 0$ almost surely for every n. If $\mathcal{K}$ is empty, this proves the stated conclusion. Otherwise, the finite second moments and the recursion imply inductively that $\mathbb{E}[\sum_k Z_{n,k}^2] < \infty$ for every finite n. Let $V(z) = \sum_{k\in\mathcal{K}} z_k^2$. Applying (17), conditioning on Z_n, and using independence gives

$$\mathbb{E}[V(Z_{n+1}) - V(Z_n) \mid Z_n] \le K - 2 \sum_{k\in\mathcal{K}} (B_k - \alpha_k) Z_{n,k},$$

where $K = \sum_{k\in\mathcal{K}} (\mathbb{E}[A_{0,k}^2] + B_k^2) < \infty$. Take expectations, sum from $n = 0$ to $R-1$, use $V(Z_R) \ge 0$, and retain the finite term $\mathbb{E}[V(Z_0)]$. After division by $2R$, that term vanishes; taking a limsup gives (19).

For the last statement, average $Q_{nP+j,k} \le Z_{n,k} + H_{n,k}$ over $j = 0, \dots, P-1$, then over frames. The frame-boundary term is stable by (19), and the uniform bound on $\mathbb{E}[H_{n,k}]$ controls the interpolation term. For an arbitrary horizon T, take $R = \lceil (T+1)/P \rceil$. Nonnegativity bounds the sum over slots $1, \dots, T$ by the sum over $0, \dots, RP-1$, and $RP/T \to 1$. This proves slot-level stability in the convention of Definition 2.2. □

B Proof of Periodic Safety and Stability

Proof of Theorem 3.1

Proof. Set $a_{s,k} = 0$ for $s \le 0$. A class-k request activated in slot $t - q + 1$ contributes $l_k + q$ in slot t if it has reached decoding stage q. Every request under the algorithm is decoded continuously, hence the actual post-token, pre-release memory in slot t is exactly

$$U_t = \sum_{k=1}^{m} \sum_{q=1}^{o_k} (l_k + q) a_{t-q+1,k}. \tag{20}$$

By (13), $a_{t-q+1,k} \le b_{\langle t-q+1\rangle_P,k}$; therefore $U_t \le G_{\langle t\rangle_P}(b) \le M$. This proves pathwise safety. Since overflow never occurs, no eviction occurs.

For every class, Lemma A.1 gives

$$Q_{t,k} = (Q_{t-1,k} + N_{t,k} - b_{\langle t\rangle_P,k})^+. \tag{21}$$

At frame boundaries $Z_{n,k} := Q_{nP,k}$, Lemma A.2 gives

$$Z_{n+1,k} \le (Z_{n,k} - B_k)^+ + A_{n,k}, \quad A_{n,k} := \sum_{t=nP+1}^{(n+1)P} N_{t,k}.$$

The vectors A_n are i.i.d., independent of Z_n, $\mathbb{E}[A_{n,k}] = P\lambda_k$, and have finite second moments. Every coordinate with $\lambda_k = 0$ has $N_{t,k} = 0$ almost surely because it is nonnegative, and can be omitted. Within frame n, the waiting queue can increase by at most $A_{n,k}$, so the slot-interpolation condition of Lemma A.3 holds with $H_{n,k} = A_{n,k}$. That lemma proves stability of the waiting queues at frame boundaries and in slots. The number of active requests is at most $M/\min_k(l_k + 1)$ on every sample path, because every active request has generated at least one token in its activation slot. Adding this deterministic bound proves stability of X_t.

□

C Completeness of periodic admission

For a fixed period, define the loads supported by safe budgets as follows.

Definition C.1.

$$\mathcal{R}_P = \left\{ x \in \mathbb{R}_+^m : \exists b \in \mathbb{Z}_+^{P \times m},\ G_r(b) \le M\ (r \in [P]),\ x_k \le \mu_k\ (k \in [m]) \right\}. \tag{22}$$

For $\lambda \in \mathbb{R}_+^m$, let $\mathfrak{A}(\lambda)$ be the collection of all arrival laws satisfying Assumption 2.1 with mean λ. This collection is nonempty: independently across coordinates and slots one may take $N_{t,k} = \lfloor \lambda_k \rfloor + Z_{t,k}$, where $Z_{t,k}$ is Bernoulli with mean $\lambda_k - \lfloor \lambda_k \rfloor$. Define

$$\Lambda_{\text{stab}}^{\exists} := \{\lambda : \exists \mathsf{A} \in \mathfrak{A}(\lambda) \exists \text{ a feasible causal policy stabilizing } \mathsf{A}\}, \tag{23}$$

$$\Lambda_{\text{stab}}^{\forall} := \{\lambda : \forall \mathsf{A} \in \mathfrak{A}(\lambda) \exists \text{ a feasible causal policy stabilizing } \mathsf{A}\}. \tag{24}$$

The order in (24) is $\forall \mathsf{A}\, \exists \pi_{\mathsf{A}}$. A fixed periodic budget works for every arrival law whose mean has strict slack below its service rates. The region Λ in the main text is $\Lambda_{\text{stab}}^{\exists}$.

Theorem C.2 (Closure characterization)**.** *For known output lengths,*

$$\operatorname{cl}(\Lambda_{\text{stab}}^{\forall}) = \operatorname{cl}(\Lambda_{\text{stab}}^{\exists}) = \operatorname{cl}\left(\bigcup_{P \ge 1} \mathcal{R}_P \right), \tag{25}$$

Here $\operatorname{cl}(S)$ *denotes the closure of* S *in* $\mathbb{R}_+^m$*, the set of limits of sequences of points in* S*.*

Lemma C.3 (A deterministic completion-rate subsequence)**.** *Suppose a causal policy stabilizes an arrival law in* $\mathfrak{A}(\lambda)$*. If* $D_k(T)$ *is the number of class-*k *completions by the end of slot* T*, then there is a deterministic sequence* $T_j \to \infty$ *such that*

$$\frac{\mathbb{E}[D_k(T_j)]}{T_j} \longrightarrow \lambda_k, \quad k = 1, \dots, m. \tag{26}$$

Proof. By stability, there is a finite K such that, for every sufficiently large T,

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{k} \mathbb{E}[X_{t,k}] \le K.$$

There must be infinitely many deterministic slots t for which

$$\sum_{k} \mathbb{E}[X_{t,k}] \le K + 1;$$

otherwise every sufficiently large term would exceed $K+1$ and the time-average liminf would exceed K. Select an increasing sequence of such slots and call it T_j. Then

$$0 \le \frac{\mathbb{E}[X_{T_j,k}]}{T_j} \le \frac{K+1}{T_j} \longrightarrow 0$$

for every class.

Let $A_k(T) = \sum_{t=1}^{T} N_{t,k}$. The system starts empty, requests cannot be dropped, and an evicted request remains unfinished. Therefore the pathwise flow-conservation identity is

$$A_k(T) = D_k(T) + X_{T,k}.$$

Taking expectations and using $\mathbb{E}[A_k(T)] = T\lambda_k$ proves (26). □

Proof of Proposition 2.3

Proof. Every class-k request completed by slot T has a final successful attempt that generates its o_k tokens in distinct slots. Its footprint when token q is generated is $l_k + q$. Summing these successful-token footprints over completed requests counts only part of the memory occupied over slots $1, \dots, T$. Feasibility therefore gives

$$\sum_{k} w_k D_k(T) \le \sum_{t=1}^{T} U_t \le MT.$$

If the policy is stable, Lemma C.3 provides deterministic $T_j \to \infty$ with $\mathbb{E}[D_k(T_j)]/T_j \to \lambda_k$ for every class. Taking expectations, dividing by T_j, and passing to the limit yields $\sum_k \lambda_k w_k \le M$. □

Lemma C.4 (Finite-horizon right-justification)**.** *Fix a feasible sample path of an arbitrary causal policy and a horizon T. The requests completed by T can be transformed into an eviction-free schedule in slots $1, \dots, T$ with all of the following properties:*

(i) every retained request is activated no earlier than its arrival;

(ii) after activation it generates one token in each consecutive slot and completes at its original completion slot;

(iii) transformed memory in every slot is no larger than original memory; and

(iv) the transformed active set is empty before slot 1 and after slot T.

Proof. For each request completed by T, delete every failed attempt and retain its final successful attempt. Consider a retained class-k request completed in slot c. Let

$$u_1 < u_2 < \cdots < u_{o_k} = c$$

be the slots in which that final attempt generated its output tokens. After token q, the remaining $o_k - q$ tokens must occupy distinct later slots, so

$$u_q \le c - o_k + q. \tag{27}$$

Move token q to $\widetilde{u}_q = c - o_k + q$. Since $\widetilde{u}_1 \ge u_1$, the moved activation is no earlier than the final attempt's activation and hence no earlier than the request's arrival.

At slot $\widetilde{u}_q$, inequality (27) implies that the original final attempt had already generated at least q tokens and had not completed before that slot. Its original footprint in that slot was therefore at least $l_k + q$, exactly the moved footprint. For $q = o_k$, both schedules count the last-token footprint before release. The comparison holds request by request, and all retained final attempts were part of the original active set. Summing gives $\widetilde{U}_t \le U_t \le M$ for each t. Every moved request completes at its original completion slot by T, proving the empty terminal state and all four claims. □

Lemma C.5 (Finite safe blocks)**.** *For fixed T, the right-justified schedule induces an integer array $B^T = (B^T_{t,k} : 1 \le t \le T, 1 \le k \le m)$ taking values in a finite set $\mathcal{B}_T$. Every $b \in \mathcal{B}_T$ is a feasible empty-to-empty block, and its periodic repetition is a feasible period-T reservation. With*

$$\rho_k(b) := \frac{1}{T} \sum_{t=1}^{T} b_{t,k},$$

we have

$$\frac{\mathbb{E}[D(T)]}{T} = \sum_{b \in \mathcal{B}_T} \mathbb{P}(B^T = b)\rho(b). \tag{28}$$

Proof. Let $B^T_{t,k}$ count the moved class-k activations in slot t. Every completed request gives one moved activation, so

$$\sum_{t=1}^{T} B^T_{t,k} = D_k(T). \tag{29}$$

Every new class-k activation contributes at least $l_k + 1$ in its activation slot. Feasibility therefore gives $0 \le B^T_{t,k} \le \lfloor M/(l_k + 1) \rfloor$. Only finitely many integer arrays satisfy these coordinatewise bounds, hence the support $\mathcal{B}_T$ is finite. Taking expectations in (29) proves (28).

If $b_{t,k} > 0$, each associated request completes within the block, so $t + o_k - 1 \le T$. Hence no activated request crosses a block boundary. Repeating the block reproduces the same within-block memory path in every period. Its cyclic convolution $G_r(b)$ is exactly that memory path, which is at most M by Lemma C.4. Thus the repetition is a feasible period-T reservation. □

Lemma C.6 (Rational empty-to-empty time sharing)**.** *Let $b^1, \dots, b^J$ be feasible empty-to-empty blocks of a common length T. For any probability vector $p = (p_1, \dots, p_J)$ and $\epsilon > 0$, there is a feasible deterministic periodic reservation with mean budget $\widehat{\rho}$ such that*

$$\left\| \widehat{\rho} - \sum_{j=1}^{J} p_j \rho(b^j) \right\|_\infty < \epsilon. \tag{30}$$

Proof. Put $R_T = 1 + \max_j \|\rho(b^j)\|_\infty$. Choose L large enough and nonnegative integers $a_1, \dots, a_J$ with $\sum_j a_j = L$ and

$$\sum_{j=1}^{J} |a_j/L - p_j| < \epsilon/R_T. \tag{31}$$

Such integers exist: take $a_j = \lfloor Lp_j \rfloor$ for $j < J$ and assign the remaining mass to a_J; the total error is at most $2(J-1)/L$.

Concatenate a_j copies of b^j for all j, in a fixed order. Every block starts and ends empty, so no request crosses an internal boundary. The concatenation has length LT, is feasible, starts and ends empty, and its periodic repetition is feasible. Its mean budget is $\widehat{\rho} = \sum_j (a_j/L)\rho(b^j)$. Inequality (30) follows from (31) and the definition of R_T. □

PROOF OF THEOREM C.2

Proof. We prove both inclusions.

First fix P, $x \in \mathcal{R}_P$, and a witnessing safe budget b with mean $\mu \geq x$. For $\eta \in (0,1)$ put $x_k^{(\eta)} = (1-\eta)x_k$ when $x_k > 0$, and zero otherwise. Every positive coordinate satisfies $x_k^{(\eta)} < x_k \leq \mu_k$. A nonnegative integer arrival count with mean zero is zero almost surely. Theorem 3.1 therefore shows that the same periodic policy stabilizes every arrival law in $\mathfrak{A}(x^{(\eta)})$. Hence $x^{(\eta)} \in \Lambda^{\forall}_{\text{stab}}$ and $x^{(\eta)} \to x$. Taking the union over P and closures gives

$$\operatorname{cl}\left(\bigcup_{P \geq 1} \mathcal{R}_P\right) \subseteq \operatorname{cl}(\Lambda^{\forall}_{\text{stab}}) \subseteq \operatorname{cl}(\Lambda^{\exists}_{\text{stab}}). \tag{32}$$

For the reverse inclusion, take $\lambda \in \Lambda^{\exists}_{\text{stab}}$. Some law in $\mathfrak{A}(\lambda)$ is stabilized by a causal policy. Lemma C.3 gives deterministic $T_j \to \infty$ such that $d^{(j)} := \mathbb{E}[D(T_j)]/T_j \to \lambda$. At each T_j, Lemmas C.4 and C.5 express $d^{(j)}$ as a finite convex combination of rates of feasible empty-to-empty blocks of length T_j. Apply Lemma C.6 with error $1/j$. It produces a deterministic periodic budget with mean $\widehat{d}^{(j)}$ satisfying

$$\widehat{d}^{(j)} \in \bigcup_{P \geq 1} \mathcal{R}_P, \quad \|\widehat{d}^{(j)} - d^{(j)}\|_\infty < 1/j.$$

Thus $\widehat{d}^{(j)} \to \lambda$, which proves $\Lambda^{\exists}_{\text{stab}} \subseteq \operatorname{cl}(\cup_P \mathcal{R}_P)$. Taking closures and combining with (32) proves (25). □

The closure identity yields the following stability guarantee for scaled-down loads in the known-output model.

Corollary C.7 (Strictly supportable loads)**.** *Let $\mathcal{K} := \operatorname{cl}(\Lambda^{\exists}_{\text{stab}})$. For every $\lambda \in \mathcal{K}$ and every $\eta \in (0,1)$, there are a finite period P and a deterministic safe budget b such that the corresponding periodic policy is eviction-free and stabilizes every arrival law in $\mathfrak{A}((1-\eta)\lambda)$. The same budget works for all such laws. Consequently, if $(1+\epsilon)\lambda \in \mathcal{K}$ for some $\epsilon > 0$, there is a periodic policy stabilizing every law in $\mathfrak{A}(\lambda)$.*

Proof. If $\lambda = 0$, nonnegativity and zero means imply zero arrivals almost surely, and the zero budget suffices. Otherwise set $I := \{k : \lambda_k > 0\}$ and $d := \frac{\eta}{2} \min_{k \in I} \lambda_k > 0$. Theorem C.2 provides $x \in \bigcup_{P \geq 1} \mathcal{R}_P$ with $\|x - \lambda\|_\infty < d$. Choose a witnessing finite-period budget with mean $\mu \geq x$. For every $k \in I$,

$$\mu_k \geq x_k > \lambda_k - d \geq (1 - \eta/2)\lambda_k > (1-\eta)\lambda_k.$$

The zero coordinates have no arrivals almost surely. The budget satisfies all cyclic memory constraints, so Theorem 3.1 applies to every arrival law with the stated reduced mean. Its conditions depend on that mean and finite second moments, giving the same budget for all of these laws. For the final assertion apply the first assertion to $(1+\epsilon)\lambda$ with $\eta = \epsilon/(1+\epsilon)$. □

The period and budget in the corollary depend on the target load and on η.

Proof of Theorem 3.3

Proof. For $\lambda = 0$, the zero budget suffices. For a nonzero load in the interior of Λ, choose $\epsilon > 0$ such that $(1+\epsilon)\lambda \in \Lambda$. In particular, an interior point admits such a sufficiently small radial step. Since $\Lambda = \Lambda^{\exists}_{\mathrm{stab}}$, the enlarged load belongs to $\mathcal{K} = \mathrm{cl}(\Lambda^{\exists}_{\mathrm{stab}})$. Corollary C.7 supplies one finite periodic budget stabilizing every law in $\mathfrak{A}(\lambda)$. Its memory safety and eviction freedom follow from Theorem 3.1. □

D Constructing periodic budgets

This appendix proves the results of Section 4.

D.1 Proof of Proposition 4.1

Proof. The $q = 1$ term of the memory constraints in (5) gives $0 \le b_{r,k} \le \lfloor M/(l_k+1) \rfloor$. Thus there are finitely many integer arrays. Since K_+ is nonempty and $\lambda_k > 0$ for every $k \in K_+$, (5) bounds η above for each array, so an optimum is attained and finite. Feasibility follows by choosing the zero budget and $\eta \le -1$.

If the optimum has $\eta > 0$, the optimal budget has mean at least $(1+\eta)\lambda_k > \lambda_k$ on every positive coordinate. Conversely, if a feasible budget has mean $\mu_k > \lambda_k$ on every $k \in K_+$, setting $\eta = \min_{k \in K_+}(\mu_k - \lambda_k)/\lambda_k > 0$ makes it feasible in the program. Setting the budgets of zero-rate classes to zero can only reduce memory usage and does not affect positive-rate service. The variable and constraint counts are immediate. □

For an all-zero target, the zero budget is already safe and stable. A positive optimum supplies a stability certificate at the chosen period. A feasible period can be repeated an integer number of times without changing its mean or safety.

D.2 Balanced budgets and backlog guarantees

The next results compare integer periodic budgets with an ideal constant fractional service rate. The fractional queue supplies a lower benchmark for integer periodic service at the same mean.

For a nonnegative deterministic service sequence $c = (c_t)$, possibly real-valued, and a fixed nonnegative arrival path $n = (n_t)$, write $Q_t(c; n)$ for the recursion

$$Q_t(c; n) = (Q_{t-1}(c; n) + n_t - c_t)^+, \quad Q_0(c; n) = 0.$$

Lemma D.1 (Interval discrepancy controls backlog)**.** *Suppose two service sequences c and d satisfy*

$$\left| \sum_{v=u+1}^{t} (c_v - d_v) \right| \le \Gamma \quad \textit{for every } 0 \le u < t. \tag{33}$$

Then, on every arrival path and at every t,

$$|Q_t(c; n) - Q_t(d; n)| \le \Gamma. \tag{34}$$

Proof. By the proof of Lemma A.1, the reflection formula remains valid for nonnegative real service. For each u, the two affine expressions inside the maxima differ by at most Γ under (33). If two finite collections of real numbers differ coordinatewise by at most Γ, their maxima differ by at most Γ. This proves (34). □

Fix a rational target rate $\beta \ge 0$ and a phase $\varphi \in [0, 1)$. Write

$$b_t^{\beta,\varphi} = \lfloor \beta t + \varphi \rfloor - \lfloor \beta(t-1) + \varphi \rfloor$$

for the single-class version of (6). It is a nonnegative integer. If $P\beta$ is an integer, it is P-periodic. For every integer a and $h \ge 1$, telescoping gives

$$\left| \sum_{t=a+1}^{a+h} b_t^{\beta,\varphi} - \beta h \right| < 1. \tag{35}$$

Indeed, the left-hand expression before taking absolute values is the difference of two fractional parts.

Definition D.2 (Phase-averaged finite-horizon backlog)**.** Let c be a P-periodic service sequence with total $\sum_{r=1}^{P} c_r = P\beta$. Its rotation by h phases is $c_t^{(h)} = c_{t+h}$, with periodic indices. For i.i.d. arrivals with finite second moment and mean λ, define

$$J_T(c) := \frac{1}{P}\sum_{h=0}^{P-1}\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[Q_t(c^{(h)}; N)]. \tag{36}$$

For the constant real service sequence $c_t \equiv \beta$, write $J_T(\beta)$.

Theorem D.3 (Additive-one backlog optimality)**.** *Fix $P, B \in \mathbb{N}$ and let $\beta = B/P > \lambda$. Among all nonnegative integer P-periodic service sequences with period total B, the ideal fractional queue is a lower bound and every mechanical sequence with rate β is within one request of that lower bound:*

$$J_T(\beta) \le \inf_{c\in\mathbb{Z}_+^P:\,\sum_r c_r = B} J_T(c) \le J_T(b^{\beta,\varphi}) < J_T(\beta) + 1 \quad (T \in \mathbb{N}). \tag{37}$$

Moreover,

$$\limsup_{T\to\infty} J_T(\beta) \le \frac{\mathbb{E}[(N_1 - \beta)^2]}{2(\beta - \lambda)}, \tag{38}$$

so the mechanical queue is stable and its long-run average expected backlog is at most the right-hand side of (38) plus one.

Proof. Fix t and a deterministic arrival vector $(n_1, \dots, n_t)$. By the reflection formula,

$$(c_1, \dots, c_t) \longmapsto Q_t(c; n)$$

is the maximum of finitely many affine functions and zero, and is therefore convex. At each slot v, the average over all rotations is

$$\frac{1}{P}\sum_{h=0}^{P-1} c_v^{(h)} = \frac{1}{P}\sum_{r=1}^{P} c_r = \beta.$$

Jensen's inequality for a finite average follows directly from convexity by induction on the number of points; applying it here yields

$$Q_t(\beta; n) \le \frac{1}{P}\sum_{h=0}^{P-1} Q_t(c^{(h)}; n).$$

Taking expectations and averaging over t proves the first inequality in (37). The middle inequality is the definition of an infimum. Every rotation of a mechanical sequence is again mechanical, with a different phase. Equations (35) and (34) therefore give $|Q_t(b^{(h)}; n) - Q_t(\beta; n)| < 1$ for every h, t, n. Averaging proves the last inequality: at each fixed t, the reflection formula involves only finitely many intervals, whose largest service discrepancy is strictly below one independently of the arrival path.

It remains to prove (38). Let $R_t = Q_t(\beta; N)$. Since $(x^+)^2 \le x^2$,

$$R_t^2 - R_{t-1}^2 \le (N_t - \beta)^2 + 2R_{t-1}(N_t - \beta).$$

The current arrival is independent of R_{t-1}. Take expectations, sum from 1 to T, use $\mathbb{E}[R_T^2] \ge 0$, and divide by $2T(\beta - \lambda)$ to obtain

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[R_{t-1}] \le \frac{\mathbb{E}[(N_1 - \beta)^2]}{2(\beta - \lambda)}.$$

The same summed inequality, before dropping its nonnegative drift term, gives

$$\mathbb{E}[R_T^2] + 2(\beta - \lambda)\sum_{t=1}^{T}\mathbb{E}[R_{t-1}] \le T\mathbb{E}[(N_1 - \beta)^2].$$

In particular,

$$0 \le \frac{\mathbb{E}[R_T]}{T} \le \frac{\sqrt{\mathbb{E}[R_T^2]}}{T} \le \sqrt{\frac{\mathbb{E}[(N_1-\beta)^2]}{T}} \longrightarrow 0.$$

The difference between averaging R_{t-1} and R_t is exactly $(\mathbb{E}[R_T]-\mathbb{E}[R_0])/T$, so the two conventions have the same limsup. This establishes the displayed limsup and the mechanical bound follows from the pathwise additive-one comparison. □

Lemma D.4 (Exact queue-area identity)**.** *On a fixed sample path of one FIFO waiting queue, let request i arrive in slot r_i and activate in slot $s_i \ge r_i$, taking $s_i = \infty$ if it never activates. For every T,*

$$\sum_{t=1}^{T} Q_t = \sum_{i:s_i \le T} (s_i - r_i) + \sum_{i:r_i \le T < s_i} (T - r_i + 1), \tag{39}$$

where Q_t is the queue after slot-t activation.

Proof. Count pairs (i,t) such that request i is waiting after the activation in slot t. Summing first over t gives the left side. A request activated by T contributes once for $t = r_i, \dots, s_i - 1$, namely $s_i - r_i$ times. A request still waiting after slot T contributes once for $t = r_i, \dots, T$, namely $T - r_i + 1$ times. Summing first over requests gives the right side. □

Remark D.5 (Precise latency interpretation). Equation (39) connects backlog and waiting time on each sample path. Whenever the second term in (39) is denoted by $R(T)$ and the number of activations by $D(T) > 0$, the finite-horizon average waiting time is exactly

$$\frac{\sum_{t=1}^{T} Q_t - R(T)}{D(T)}.$$

If $R(T)/T \to 0$, $D(T)/T \to \lambda > 0$, and $T^{-1}\sum_{t=1}^{T} Q_t \to L < \infty$, this average converges to L/λ. Under continuous decoding, class-k end-to-end latency then adds exactly o_k slots. Theorem D.3 itself is a backlog guarantee; a latency comparison additionally requires these convergence conditions for each schedule being compared.

D.3 Proof of Theorem 4.2

Proof. Fix class k and write $d_t = \lfloor \beta_k t + \varphi_k \rfloor - (\beta_k t + \varphi_k)$. Then $-1 < d_t \le 0$ and $b_{t,k} - \beta_k = d_t - d_{t-1}$. Put $c_q = l_k + q$. Summation by parts gives

$$\begin{aligned}\sum_{q=1}^{o_k} c_q (b_{r-q+1,k} - \beta_k) &= c_1 d_r + \sum_{q=2}^{o_k} (c_q - c_{q-1}) d_{r-q+1} - c_{o_k} d_{r-o_k} \\ &< (l_k + o_k),\end{aligned}$$

because the first two terms are nonpositive and $-c_{o_k} d_{r-o_k} < c_{o_k} = l_k + o_k$. Adding $\beta_k \sum_q c_q = \beta_k w_k$ and summing over k proves (7). Condition (8) implies the cyclic safety constraints, while each mean budget is $\beta_k > \lambda_k$; Theorem 3.1 completes the proof. □

Corollary D.6 (Additive gap to the workload outer bound)**.** *Let $H := \sum_k (l_k + o_k)$. Every arrival-rate vector satisfying*

$$\sum_{k=1}^{m} \lambda_k w_k < M - H \tag{40}$$

is stabilized eviction-free by a balanced periodic policy. On the other hand, every stabilizable vector satisfies

$$\sum_{k=1}^{m} \lambda_k w_k \le M. \tag{41}$$

Thus, with the classes and their lengths fixed, the balanced construction has an additive memory gap at most H, and its radial relative gap vanishes as $M \to \infty$.

Proof. Under (40), density of the rationals permits rational $\beta_k > \lambda_k$ sufficiently close to λ_k that (8) holds. This proves the inner result.

The outer inequality is Proposition 2.3. This proves the claimed additive memory gap. With the classes and their lengths fixed, dividing this gap by M gives the stated asymptotic comparison. □

E UNKNOWN OUTPUT LENGTHS: ROBUST RESERVATIONS AND SAFE FILL

Index the observed input classes by $\ell \in \mathcal{L} = \{1, \dots, d\}$. A request of class ℓ has input length l_ℓ and an unknown output length $1 \le O_i \le C_\ell$, where C_ℓ is a known finite upper bound. Write $C_{\max} = \max_\ell C_\ell$. The arrival vectors satisfy Assumption 2.1 with these input classes. The baseline uses its deterministic periodic activation opportunities in FIFO order and discards an opportunity when its queue is empty, and every activated request is decoded continuously. The safety statements below hold pathwise for every realization of the output lengths within the bounds C_ℓ.

Replacing each output length by its class bound gives

$$G_r^{\max}(b) = \sum_{\ell=1}^{d} \sum_{q=1}^{C_\ell} (l_\ell + q) b_{\langle r-q+1 \rangle_P, \ell}. \tag{42}$$

Theorem E.1 (Periodic reservations for unknown lengths)**.** *Suppose*

$$G_r^{\max}(b) \le M \quad (r \in [P]), \tag{43}$$

$$P^{-1} \sum_{r=1}^{P} b_{r,\ell} > \lambda_\ell \quad (\lambda_\ell > 0). \tag{44}$$

The FIFO periodic policy is memory-safe and eviction-free for every output realization within these bounds, and stable under Assumption 2.1.

E.1 PROOF OF THEOREM E.1

Proof. Let $a_{s,\ell}$ be the actual class-ℓ activations at slot s, and set $a_{s,\ell} = 0$ for $s \le 0$. Since $a_{s,\ell} \le b_{\langle s \rangle_P, \ell}$, the realized post-token, pre-release footprint obeys

$$\begin{aligned} U_t &\le \sum_{\ell \in \mathcal{L}} \sum_{q=1}^{C_\ell} (l_\ell + q) a_{t-q+1,\ell} \\ &\le G_{\langle t \rangle_P}^{\max}(b) \le M. \end{aligned}$$

The first inequality replaces each actual lifetime by its class upper bound. This proves pathwise safety and eviction freedom.

The waiting queue is independent of output marks because a request leaves it upon activation. Its exact recursion is

$$Q_{t,\ell} = (Q_{t-1,\ell} + N_{t,\ell} - b_{\langle t \rangle_P, \ell})^+.$$

At period boundaries, Lemma A.2 gives

$$Q_{(n+1)P,\ell} \le (Q_{nP,\ell} - B_\ell)^+ + A_{n,\ell},$$

where

$$B_\ell = \sum_r b_{r,\ell}, \quad A_{n,\ell} = \sum_{t=nP+1}^{(n+1)P} N_{t,\ell}.$$

If $\lambda_\ell = 0$, nonnegativity implies $N_{t,\ell} = 0$ almost surely. A coordinate with $B_\ell = 0$ may therefore be omitted, while a coordinate with $B_\ell > 0$ satisfies $0 = P\lambda_\ell < B_\ell$. Within frame n, the waiting queue can increase by at most $A_{n,\ell}$, so the slot-interpolation condition holds with $H_{n,\ell} = A_{n,\ell}$. The strict rate condition and Lemma A.3 prove waiting-queue stability. Pathwise memory safety bounds the active population by $M / \min_\ell(l_\ell + 1)$, proving full-system stability. □

Proposition E.2 (Necessity for a saturated robust baseline)**.** *Assume additionally that each C_ℓ is the maximum of its class output support. Fix a period and budget, and suppose every input-class queue is saturated so that every opportunity is used. Safety for every coordinatewise supported output realization holds if and only if $G_r^{\max}(b) \le M$ for every phase r. The necessity statement is exact within this fixed-period, saturated, open-loop policy class.*

Proof. Sufficiency follows from the pathwise memory comparison in the preceding proof. For necessity, after $C_{\max} - 1$ warm-up slots choose the supported realization in which every request contributing to a violated phase has output length C_ℓ. Saturation makes every opportunity used, so the memory in that phase is exactly $G_r^{\max}(b)$. Every phase occurs infinitely often; if one value exceeds M, the required pathwise safety fails. □

E.2 Rolling-calendar implementation of safe fill

Safe fill retains every future baseline reservation and uses calendar slack, including capacity released by unused baseline opportunities or early completions. The complete future trajectory must be checked; current headroom alone is insufficient.

For each actual baseline opportunity $e = (s, \ell, j)$, $1 \le j \le b_{\langle s \rangle_P, \ell}$, define its upper-bound reservation vector

$$v_e(u) := \begin{cases} l_\ell + q, & u = s + q - 1, \quad q = 1, \dots, C_\ell, \\ 0, & \text{otherwise.} \end{cases} \tag{45}$$

An extra class-ℓ request activated at s receives the same vector.

Definition E.3 (Reservation ledger and rolling calendar)**.** At the beginning of slot t, the conceptual ledger contains:

(i) every unprocessed baseline opportunity with activation slot $s \ge t$, carrying its vector (45); and

(ii) every request activated before t that has not completed by the end of $t - 1$, carrying all unconsumed entries of its upper-bound vector.

For $u \ge t$, let $R_t(u)$ be the sum of all ledger entries at u. Its finite rolling representation is

$$r_t(h) := R_t(t + h), \quad h = 0, \dots, C_{\max} - 1. \tag{46}$$

At $t = 1$, the ledger contains all actual baseline opportunities from slot one onward and no request. Hence

$$r_1(h) = \sum_{\ell \in \mathcal{L}} \sum_{q=1}^{\min\{C_\ell, h+1\}} (l_\ell + q) b_{\langle 1+h-q+1 \rangle_P, \ell} \le G^{\max}_{\langle 1+h \rangle_P}(b) \le M. \tag{47}$$

Algorithm E.4 (Rolling safe fill)**.** In slot t, execute the following operations in order.

(S1) Add arrivals to their input-class FIFO queues.

(S2) Process each current baseline opportunity separately. If its queue is nonempty, remove the opportunity object, activate the FIFO head, and replace the object by the request's identical upper-bound vector; the calendar values do not change. If its queue is empty, remove that one opportunity and its entire vector. Thus a partially used budget releases only the individually unused opportunity vectors.

(S3) Visit classes in a fixed order. For a class-ℓ FIFO head, test atomically

$$r_t(q - 1) + (l_\ell + q) \le M, \quad q = 1, \dots, C_\ell. \tag{48}$$

If every inequality holds, add all entries of the vector immediately and activate the request. The updated calendar is used to test the next request. If any inequality fails, make no change and stop testing that class in this slot. Continue to the next class.

(S4) Decode every active request by one token and measure post-token, pre-release memory.

(S5) When a request activated at s completes after realized length d in slot $t = s + d - 1$, retain its current-slot entry as a completed-pending-discard object. If $d < C_\ell$, remove exactly its unused future suffix $l_\ell + d + 1, \ldots, l_\ell + C_\ell$ from slots $t + 1, \ldots, s + C_\ell - 1$. Release nothing before the end-of-sequence signal. Let r_t^{end} denote the calendar after all such completion updates.

(S6) Discard every entry for absolute slot t, including the current entries of completed-pending-discard objects, and shift

$$r_{t+1}(h) = r_t^{\mathrm{end}}(h+1), \quad h = 0, \ldots, C_{\max} - 2, \tag{49}$$

and set the new frontier $u = t + C_{\max}$ to

$$r_{t+1}(C_{\max} - 1) = G^{\max}_{\langle u \rangle_P}(b). \tag{50}$$

The frontier formula is exact. A request activated no later than t has reserved completion no later than $t + C_{\max} - 1$, so no accepted request reaches the new frontier. Every baseline opportunity that can contribute at $u = t + C_{\max}$ has activation time $u - q + 1 \geq t + 1$ and remains unprocessed; their sum is exactly (50).

Lemma E.5 (Rolling-ledger invariant)**.** *Assume* (43)*. The following statements hold at the beginning of every slot, after each ledger-changing operation in (S2) and (S3), after all completion updates in (S5), and after the shift-and-frontier update in (S6):*

(I1) each undiscarded calendar value is at most M;

(I2) every unprocessed baseline opportunity retains its complete trajectory;

(I3) every accepted incomplete request retains every unconsumed entry of its upper-bound trajectory, while every request completed in the current slot retains exactly its current entry as a completed-pending-discard object until (S6); and

(I4) the calendar is exactly the entrywise sum of the objects in (I2)–(I3).

Proof. Initialization follows from (47). Assume the invariant at the beginning of slot t. Replacing a used baseline opportunity by an identical request vector changes only ownership, not any value. Removing an unused opportunity decreases exactly its entries. This also covers $b_{\langle t \rangle_P, \ell} > 1$ and partial use because opportunities are processed one at a time.

An extra request is accepted only after its complete vector passes (48). Acceptance and insertion are one atomic operation; therefore every changed value remains at most M before the next candidate is tested. Rejection changes nothing. Decoding changes neither ownership nor reservation values. An end-of-sequence signal certifies that the completed request cannot use its reserved suffix; deleting exactly that suffix decreases the calendar and leaves only its current entry as a completed-pending-discard object. Step (S6) then discards that current entry.

The rolling shift only changes relative indices. As shown after Algorithm E.4, the new frontier consists of all and only the unprocessed baseline entries reaching that slot, has value $G^{\max}_{\langle u \rangle_P}(b) \leq M$, and contains no accepted-request entry. Thus all four statements hold at slot $t + 1$. Induction over slots and over the sequential atomic operations proves the lemma. □

Theorem E.6 (Safe-fill safety and baseline preservation)**.** *Under* (43)*, Algorithm E.4 is eviction-free for every locally finite arrival path and supported output realization. An extra request can never block a current or future baseline opportunity: whenever such an opportunity is processed and its queue is nonempty, it can activate the FIFO head without another memory test.*

Proof. Immediately before decoding in slot t, every active request owns its current upper-bound reservation. Its realized footprint is no larger than that reservation, so Lemma E.5 gives $U_t \leq r_t(0) \leq M$. Hence no eviction occurs. A future baseline opportunity is inserted before runtime and remains in the ledger until its activation slot. Extras are admitted only after this vector has already been counted. When the opportunity is used, it is replaced by an identical request vector and needs no new capacity. Thus every baseline opportunity activates its FIFO head whenever its queue is nonempty. □

Run a baseline system B and safe-fill system S on the same arrival path. Give simultaneous same-class arrivals a common deterministic FIFO tie break and couple equal labels to the same output marks. Let $Q^X_{t,\ell}$ be the waiting queue after slot-t activations in system $X \in \{B, S\}$. An activation or completion time is $+\infty$ if that event never occurs.

Theorem E.7 (FIFO queue and requestwise dominance)**.** *For every t and ℓ,*

$$Q^S_{t,\ell} \le Q^B_{t,\ell}. \tag{51}$$

Consequently, cumulative class-ℓ activations in S are no smaller than in B, and every FIFO-labelled request is activated and completed in S no later than its equal-labelled request in B.

Proof. After arrivals, let $W^X_{t,\ell} = Q^X_{t-1,\ell} + N_{t,\ell}$. The baseline leaves $(W^B_{t,\ell} - b_{\langle t\rangle_P,\ell})^+$. By Theorem E.6, safe fill honors the same baseline opportunities, so before extras it leaves $(W^S_{t,\ell} - b_{\langle t\rangle_P,\ell})^+$; extras only reduce this quantity. Starting from equal empty queues, monotonicity of $x \mapsto (x-b)^+$ proves (51) by induction.

Let $A_\ell(t) = \sum_{s=1}^t N_{s,\ell}$ and let $D^X_\ell(t)$ be cumulative activations. Eviction freedom and queue conservation give $Q^X_{t,\ell} = A_\ell(t) - D^X_\ell(t)$, hence $D^S_\ell(t) \ge D^B_\ell(t)$. The activation slot of FIFO label j is $\inf\{t : D^X_\ell(t) \ge j\}$, so it is no later in S. Both systems decode continuously without eviction. If the common output mark is d, completion occurs exactly $d-1$ slots after activation, proving completion dominance. □

Corollary E.8 (Safe fill)**.** *Under (43)–(44), the safe-fill policy in Appendix E is eviction-free and stable. On a common arrival path with matched output lengths, every request starts and completes no later than under the periodic baseline.*

Proof of Corollary E.8

Proof. Theorem E.1 stabilizes the baseline. Theorem E.7 transfers its waiting-queue bound to safe fill. Theorem E.6 bounds the active population by $M/\min_\ell(l_\ell + 1)$, so the full system is stable. Requestwise latency dominance was proved in Theorem E.7. □

Remark E.9 (Why the full calendar is necessary). Full-calendar checks account for current and future memory use. With one class, $l = 1$, $C = 2$, and $M = 3$, an extra request needs 2 now and 3 next slot. It fits an empty current cache, but a protected next-slot baseline activation would require another 2, creating footprint $5 > 3$. Sequential atomic tests as in (S3) also account for the reservations of extras accepted earlier in the slot.

F Periodic activation with a finite-horizon risk tolerance

We prove the risk guarantee for Algorithm 2 and connect it to the fixed-parameter optimization in (12). There are m observed input classes. Class k has input length l_k, and the output length of a class-k request, which we call its mark, has distribution F_k supported on $\{1, \dots, C_k\}$, with $l_k + C_k \le M$. Conditional on the input classes, marks are independent across external requests, identically distributed within each class, and independent of arrival counts. The counts satisfy Assumption 2.1. The system starts empty, and its period and integer budgets are fixed before serving traffic. We use the tails $p_{k,q}$ and log-mgf $\Psi_r(\theta; b)$ defined in (9) and (10).

F.1 Projected overflow and the first eviction

Immediately after the budget assignments in slot t, let $\mathcal{A}_t$ be the candidate active set, which consists of the existing active requests together with the tentatively selected requests. Let j_i be the progress of request i before decoding, which equals zero for a new attempt. Every candidate has a next token to generate because outputs are positive and previous completions have been removed. The projected footprint after decoding and before release is

$$\widetilde{U}_t = \sum_{i \in \mathcal{A}_t} (l_{k_i} + j_i + 1).$$

We define the first overflow time by

$$\tau := \inf\{t \geq 1 : \widetilde{U}_t > M\}, \quad \inf \varnothing = \infty. \tag{52}$$

The projection is formed before memory is allocated to the tentative new attempts. LIFO eviction removes the most recently activated candidate, counting tentative attempts as activated in the current slot and breaking ties by the largest request ID, and repeats until the projection fits. Each evicted request loses its generated state and joins the tail of its class waiting queue with the same mark. Its attempted activation has used its activation opportunity, and the slot has no refill after the evictions. The periodic phases, which are indexed by absolute time, continue during evictions and idle slots. Since removing all active requests makes the projection zero, this eviction procedure terminates. Committing the retained new activations and decoding all retained requests then respects the memory budget. The probability bound below concerns the event that such evictions are needed.

Label every activation opportunity of the periodic schedule by

$$e = (s, k, j), \quad s \in \mathbb{Z}, \quad k \in [m], \quad 1 \leq j \leq b_{\langle s \rangle_P, k}.$$

Attach independent virtual marks $O_e \sim F_k$ to these opportunities, which are indexed by absolute time. The opportunities with $s \leq 0$ are fictitious: their marks are used only to obtain a periodic upper bound, and they do not activate real requests.

Lemma F.1 (Deferred marks up to the first overflow)**.** *Up to the projection that triggers the first overflow, the process can be constructed without changing its law so that each request first assigned through a used opportunity e receives mark O_e. Unused opportunities retain independent marks that are ignored by the actual system.*

Proof. Order opportunities by absolute slot and then by a fixed within-slot order. Before the first overflow, every queued request is an external request that has never been attempted. Which class-k FIFO head is assigned to the next used opportunity is determined by the arrivals and the history of earlier opportunities. Its unrevealed mark is independent of that history and has distribution F_k. Assigning the fresh mark O_e therefore preserves its conditional law. The assignment decision does not use O_e; when an opportunity is unused, ignoring its mark preserves the law of the remaining process.

Induction over opportunities gives this construction through all tentative assignments in the slot of the first overflow, before any eviction in that slot. The induction may condition on previously assigned (unobserved) marks as well as their observed completion events, since the next external mark remains independent. Thus it also covers residents whose final lengths have not yet been revealed. Assigned marks remain attached to their requests on later retries. The lemma concerns the process through the first projection in (52). □

For $t \geq 1$, define the finite-start virtual footprint

$$V_t^{(0)} = \sum_{k=1}^{m} \sum_{q=1}^{\min\{C_k, t\}} \sum_{j=1}^{b_{\langle t-q+1 \rangle_P, k}} (l_k + q)\mathbf{1}\{O_{t-q+1,k,j} \geq q\}. \tag{53}$$

Its log-mgf is

$$\Psi_t^{(0)}(\theta; b) = \sum_{k=1}^{m} \sum_{q=1}^{\min\{C_k, t\}} b_{\langle t-q+1 \rangle_P, k} \log\bigl(1 - p_{k,q} + p_{k,q} e^{\theta(l_k+q)}\bigr). \tag{54}$$

Lemma F.2 (Virtual domination before the first overflow)**.** *On $\{\tau \geq t\}$, the projection before any eviction satisfies $\widetilde{U}_t \leq V_t^{(0)}$.*

Proof. No request has been evicted before slot t on this event. A request assigned at $s = t-q+1 < t$ has decoded continuously and remains in the candidate set precisely when its mark is at least q. For $s = t$, every tentative assignment has $q = 1$ and a positive mark. Its projected footprint is then $l_k + q$. Lemma F.1 assigns it the corresponding virtual mark. Expression (53) includes its term and a nonnegative term for every unused opportunity. Summing proves the claim, including the projection at $t = \tau$ before any request is removed. □

PROOF OF THEOREM 5.1

Proof. For a fixed t, every summand in (53) has a different absolute opportunity label. In particular, different ages q correspond to different slots $t-q+1$, even if the same phase occurs more than once because $C_k > P$. The summands are therefore independent, and

$$\mathbb{E}\left[e^{\theta(l_k+q)\mathbf{1}\{O_e \geq q\}}\right] = 1 - p_{k,q} + p_{k,q}e^{\theta(l_k+q)}.$$

Multiplication gives $\mathbb{E}[e^{\theta V_t^{(0)}}] = e^{\Psi_t^{(0)}(\theta;b)}$.

The footprint and M are integers. For any $\theta > 0$, Markov's inequality gives

$$\mathbb{P}(V_t^{(0)} > M) = \mathbb{P}(V_t^{(0)} \geq M+1) \leq \exp\{-\theta(M+1) + \Psi_t^{(0)}(\theta;b)\}.$$

If $\tau = t$, then $\widetilde{U}_t > M$ and Lemma F.2 implies $V_t^{(0)} > M$. Consequently,

$$\begin{aligned}
\mathbb{P}(\tau \leq T) &= \sum_{t=1}^{T} \mathbb{P}(\tau = t) \\
&\leq \sum_{t=1}^{T} \inf_{\theta>0} \exp\{-\theta(M+1) + \Psi_t^{(0)}(\theta;b)\} \\
&\leq \sum_{t=1}^{T} \inf_{\theta>0} \exp\{-\theta(M+1) + \Psi_{\langle t \rangle_P}(\theta;b)\} \\
&\leq T \max_{r\in[P]} \inf_{\theta>0} \exp\{-\theta(M+1) + \Psi_r(\theta;b)\}.
\end{aligned}$$

For the third line, each fictitious opportunity with $s \leq 0$ contributes a nonnegative log-mgf term, so $\Psi_t^{(0)}(\theta;b) \leq \Psi_{\langle t \rangle_P}(\theta;b)$ for every $\theta > 0$. Combining this bound with $\mathbb{P}(\tau \leq T) \leq 1$ proves (11). Independence was used for the unconditional virtual marks, while the stopped event was used only for the pathwise inclusion $\{\tau = t\} \subseteq \{V_t^{(0)} > M\}$. □

Proposition F.3 (Nominal queues and agreement before the first overflow)**.** *Run the same budgets and FIFO activations in a nominal system with no memory limit, no eviction, and continuous decoding. If $\mu_k > \lambda_k$ for every positive-rate class, this nominal system is stable. Coupling it to Algorithm 2 with the same arrivals and output marks, their complete trajectories agree through slot T on $\{\tau > T\}$.*

Proof. The nominal waiting queue satisfies

$$Q_{t,k}^0 = (Q_{t-1,k}^0 + N_{t,k} - b_{\langle t \rangle_P,k})^+.$$

Lemmas A.2 and A.3, applied with $B_k = P\mu_k$ and period arrivals of mean $P\lambda_k$, give stability of its waiting queues. Zero-rate coordinates have no arrivals almost surely. A nominal active request remains for at most C_k slots, so its active population is bounded by $\sum_k C_k \max_r b_{r,k}$. Adding this finite deterministic bound proves full-system stability.

Before the first overflow, the actual and nominal systems use the same activations, decoding operations, and completion marks. Induction over slots proves equality of their states and request trajectories on $\{\tau > T\}$. After the first overflow, the actual system additionally contains restarted attempts, whose marks retain their original values. □

F.2 THE FIXED-PARAMETER INTEGER PROGRAM

Proposition F.4 (Risk certificate from the integer formulation)**.** *Fix $P, T \in \mathbb{N}$, $\delta \in (0,1)$, and $\theta > 0$. Any budget feasible for (12) satisfies $\mathbb{P}(\tau \leq T) \leq \delta$. A positive service margin also gives the nominal stability condition in Proposition F.3. The risk constraints are linear in the integer budgets. If the positive-rate class set is nonempty and $\theta(M+1) \geq \log(T/\delta)$, the maximum-margin formulation has a finite attained optimum.*

Proof. For fixed θ, each coefficient

$$a_{k,q}(\theta) := \log\bigl(1 - p_{k,q} + p_{k,q}e^{\theta(l_k+q)}\bigr)$$

is a known nonnegative number, and $\Psi_r(\theta; b) = \sum_{k,q} a_{k,q}(\theta) b_{\langle r-q+1\rangle_P,k}$ is linear in b. The risk constraint in (12) makes $\exp\{-\theta(M+1)+\Psi_r(\theta; b)\} \le \delta/T$ for every phase. Theorem 5.1, evaluated at this same θ, proves the risk claim. Positive margin implies $\mu_k > \lambda_k$ on the positive coordinates, proving the nominal queueing claim.

Since $p_{k,1} = 1$, we have $a_{k,1}(\theta) = \theta(l_k+1)$. Writing $L = \theta(M+1) - \log(T/\delta) \ge 0$, every feasible entry obeys

$$0 \le b_{r,k} \le \frac{L}{\theta(l_k+1)}.$$

There are finitely many integer budget arrays. The zero budget and a sufficiently negative margin are feasible. Positive margin weights and at least one positive-rate class bound the margin above for each array; hence the maximum is finite and attained. □

For any fixed θ, deleting activation opportunities decreases every risk left-hand side. Thus fixing the period totals to $\lfloor P\lambda_k \rfloor + 1$ for positive-rate classes is also an equivalent test for a positive margin within these risk constraints. One may solve the formulation over a finite list of periods and Chernoff parameters and retain a verified feasible budget. Its guarantee concerns the selected horizon, risk tolerance, and output laws. Such a search optimizes over its specified candidates; it does not require an additional union bound over candidates when the final choice is made before traffic. The phasewise exponential expressions can be recomputed directly from the selected integer array.

Proposition F.5 (Known-output specialization)**.** *If each class output is deterministic, $O_k = o_k$ almost surely, then $\Psi_r(\theta; b) = \theta G_r(b)$ and*

$$\inf_{\theta>0} e^{-\theta(M+1)+\Psi_r(\theta;b)} = \begin{cases} 0, & G_r(b) \le M, \\ 1, & G_r(b) \ge M+1. \end{cases}$$

Moreover, for $\theta \ge \log(T/\delta)$ the fixed-parameter risk constraint is equivalent to $G_r(b) \le M$.

Proof. The tail probabilities equal one for $q \le o_k$ and zero for $q > o_k$. Thus the log-mgf is $\theta G_r(b)$. When $G_r(b) \le M$, the exponent has a negative coefficient of θ and the infimum is zero as $\theta \to \infty$. When $G_r(b) \ge M+1$, its coefficient is nonnegative and the infimum is one as $\theta \downarrow 0$.

The fixed-parameter constraint is

$$G_r(b) \le M + 1 - \frac{\log(T/\delta)}{\theta}.$$

Under the stated bound on θ, its right-hand side belongs to $[M, M+1)$. Since $G_r(b)$ is an integer, this is equivalent to $G_r(b) \le M$. In this case LIFO eviction is never needed, and the policy agrees with Algorithm 1. □

F.3 Calibration of the output tails

For class k, let $Y_{k,1}, \dots, Y_{k,n_k}$ be i.i.d. calibration outputs from F_k, with $n_k \ge 1$. The calibration sample is independent of future serving traffic. Choose $\alpha_k \in (0,1)$ with $\sum_k \alpha_k \le \alpha$, and define

$$\widehat{p}_{k,q} := \frac{1}{n_k}\sum_{i=1}^{n_k} \mathbf{1}\{Y_{k,i} \ge q\}, \quad \varepsilon_k := \sqrt{\frac{\log(2/\alpha_k)}{2n_k}}, \quad p^+_{k,q} := \min\{1, \widehat{p}_{k,q} + \varepsilon_k\}. \tag{55}$$

Let $\Psi^+_r(\theta; b)$ be obtained from (10) by replacing $p_{k,q}$ by $p^+_{k,q}$.

Theorem F.6 (A calibrated risk certificate)**.** *Fix $T \in \mathbb{N}$ and $\delta \in (0,1)$. Select a finite period, integer budget, and positive θ using only the calibration sample, such that*

$$\Psi^+_r(\theta; b) \le \theta(M+1) + \log(\delta/T), \quad r \in [P]. \tag{56}$$

Then, over calibration data and future traffic, $\mathbb{P}(\tau \le T) \le \delta + \alpha$.

Proof. Write $\widehat{F}_k$ for the empirical cdf of class k. The Dvoretzky–Kiefer–Wolfowitz inequality with the sharp constant (Massart, 1990) states

$$\mathbb{P}\left(\sup_x |\widehat{F}_k(x) - F_k(x)| > \varepsilon_k\right) \le 2e^{-2n_k\varepsilon_k^2} = \alpha_k.$$

Since $p_{k,q} = 1 - F_k(q-1)$, a union bound over classes gives an event $\mathcal{E}$ of probability at least $1-\alpha$ on which $p_{k,q} \le p^+_{k,q}$ simultaneously for every k, q. This simultaneous event covers every period and budget that may be selected from the calibration data.

Condition on a calibration sample in $\mathcal{E}$. Independence from future traffic preserves its original output laws F_k; the selected period, budget, and θ are now fixed. For $\theta > 0$, the function $p \mapsto \log(1 - p + pe^{\theta(l_k+q)})$ is nondecreasing. All budgets are nonnegative, hence $\Psi_r(\theta; b) \le \Psi^+_r(\theta; b)$ for every phase. Equation (56) and Theorem 5.1 therefore give $\mathbb{P}(\tau \le T \mid \text{calibration}) \le \delta$ on $\mathcal{E}$. On its complement the conditional probability is at most one. Averaging over calibration samples yields $\mathbb{P}(\tau \le T) \le \delta + \alpha$. □

Remark F.7 (Input bins and correlated lengths). The same risk construction applies to an observed input bin when its requests have i.i.d. pairs (L, O), independent of the bin arrival counts, with $L \le l_k$ and $O \le C_k$ almost surely. Use the bin's output marginal for $p_{k,q}$, its input upper bound l_k in the footprint, and FIFO activation within the bin. Activation opportunities are assigned without selecting on the finer value of L.

To see the coupling, condition on the arrival counts and the fixed FIFO order. In the nominal periodic system, the mapping from external request labels to the opportunities that first activate them depends only on these counts and the budgets, not on the pairs (L, O). Independent pairs can therefore be assigned to the used opportunities of the nominal system. The actual system has the same mapping before its first overflow, and each of its footprints $(L+q)\mathbf{1}\{O \ge q\}$ is at most $(l_k+q)\mathbf{1}\{O \ge q\}$. The virtual-footprint proof then applies to the independent output marginals. This argument preserves any dependence between L and O within a bin. Calibration outputs must come from the same bin distribution, independently of future requests.

G Additional Numerical Results

Figures 2 and 3 report the synthetic experiments with $\lambda = 4$ and $\lambda = 3$, and Figure 4 reports the real-world experiments with $\lambda = 50$. All figures show means over runs; for the curves in Figure 1, the standard deviation across runs is at most 10.4% of the mean, with a median below 5%.

At $\lambda = 4$, the latency of MC, MC-SF, and Amin grows with T, whereas that of our algorithm grows much more slowly: at $T = 1000$, our average latency is 48 slots with known output lengths (MC-SF: 96, MC: 124) and 60 slots with unknown output lengths (Amin: 106, MC: 209). At $\lambda = 3$, our algorithm, MC, MC-SF, and Amin keep the average latency close to its minimum of 40 slots, whereas the latency of the α-protection greedy algorithm grows with T. We compare with WAIT (Ao et al., 2025) at these two loads, selecting its thresholds by a grid search subject to $n_k \ge \lambda$ and $\sum_k n_k w_k \le M$, which gives $(n_1, n_2, n_3) = (4, 3, 3)$ at $\lambda = 3$ and $(4, 4, 4)$ at $\lambda = 4$. At $T = 1000$, its average latency is 49.2 slots at $\lambda = 3$ and 48.9 slots at $\lambda = 4$, compared with 40.3 and 48.3 slots for our algorithm. At $\lambda = 50$ and $T = 1000$, our average latency is 3,835 seconds, compared with 4,004 for Amin, 4,125 for Sarathi, and 4,831 for vLLM.

Implementation details. All policies face the same arrival sequence in each run. In the synthetic experiments, MC and MC-SF process at most 1,024 requests per batch with known output lengths, and MC and Amin process at most 512 with unknown output lengths. In the real-world experiments, vLLM and Sarathi run as the built-in schedulers of Vidur with 1,030 KV cache blocks of 16 tokens; vLLM batches at most 4,096 tokens per iteration, and Sarathi uses prefill chunks of 512 tokens.

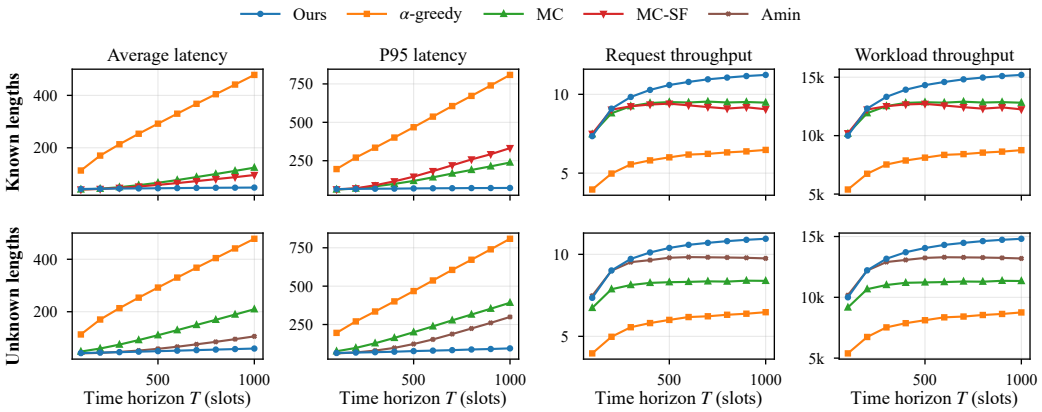

Figure 2: Synthetic data with $\lambda = 4$. Top: known output lengths; bottom: unknown output lengths. Units are as in Figure 1.

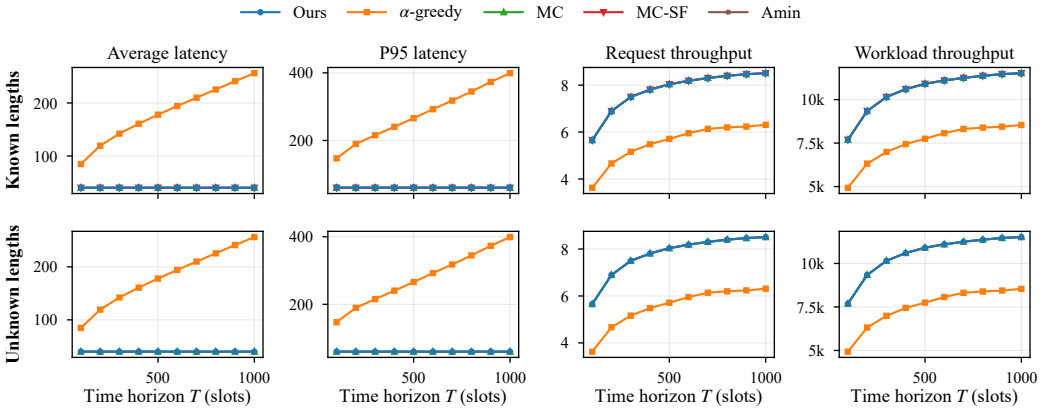

Figure 3: Synthetic data with $\lambda = 3$. Top: known output lengths; bottom: unknown output lengths. Units are as in Figure 1.

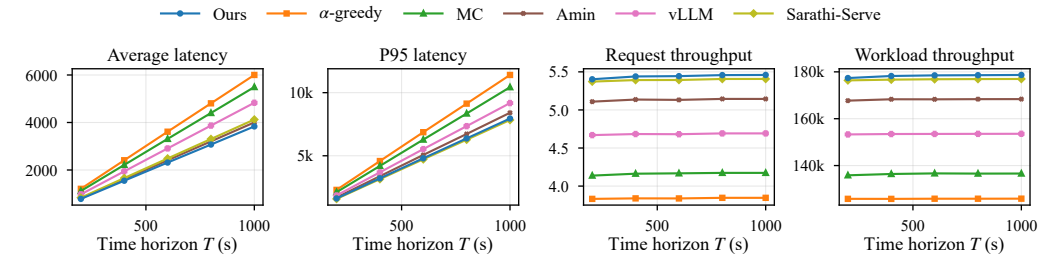

Figure 4: Real-world data with unknown output lengths and $\lambda = 50$ requests per second. Units are as in Figure 1.