# Congestion-Aware Serving of Agentic LLM Applications

Mouheb Ben Nasr
KAUST

Muhammad Bilal
KAUST

Alessandro Cornacchia
KAUST

Boris Radovič
KAUST, University of Ljubljana

Marco Canini
KAUST

## Abstract

Agentic LLM workflows issue many dependent calls with unpredictable resource demand, causing queue buildup and latency degradation on shared serving backends when left unmanaged. In this paper, we propose CALM-MAS, a congestion-aware serving framework for LLM applications that treats LLM test-time computation as an elastic resource, dynamically adjusting the compute profile of admitted tasks to tame congestion. CALM-MAS detects early signals of backend saturation, and leverages the flexibility of LLM applications to regulate load. During spikes of requests, the system downgrades agent topology and reasoning depth; during low-utilization periods, it allocates additional reasoning effort to maximize task accuracy. Compared with a static serving baseline based on vLLM, CALM-MAS reduces shared-backend tail latency by **77%** with the accuracy degradation remaining confined to **6.1** pps relative to the native agent configuration.

## CCS Concepts

• **Software and its engineering** → **Software performance**;
• **Computing methodologies** → *Multi-agent systems.*

## Keywords

LLM serving, LLM applications, multi-agent systems, congestion control, test-time scaling, SLO management

**ACM Reference Format:**
Mouheb Ben Nasr, Muhammad Bilal, Alessandro Cornacchia, Boris Radovič, and Marco Canini. 2026. Congestion-Aware Serving of Agentic LLM Applications. In *ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '26), September 17–18, 2026, Bangkok, Thailand*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3838177.3841735


*APSys '26, Bangkok, Thailand*

ACM ISBN 979-8-4007-2912-6/26/09
https://doi.org/10.1145/3838177.3841735

## 1 Introduction

Large Language Model (LLM) workflows are rapidly transitioning from stateless request-response services into dynamic, agentic systems. In traditional single-turn queries or multi-turn chats, each user message maps to one bounded generation with a predictable compute profile. Recently, however, the field has shifted toward test-time scaling, leveraging iterative reasoning, multi-step planning, and multi-agent systems (MAS) to solve complex tasks [19, 22, 33]. Agentic systems can dynamically expand thought trajectories [19, 33] and scale the inference topology from a single agent to a network of specialist agents [10], substantially improving response quality at the cost of unpredictable per-request compute.

This evolution introduces complex resource management challenges for backend infrastructure [8, 24, 44]. Modern inference clusters are now tasked with serving a bifurcated workload on shared compute. At one extreme are traditional *non-agentic* clients, which issue lightweight, single-turn requests. At the other extreme are *agentic* clients, which decompose tasks into dozens of dependent LLM calls, intermediate coordination steps, and external tool invocations [8, 22]. A single agentic task can consume millions of tokens and maintain large key-value (KV) cache entries for extended periods, injecting substantial compute overhead into the inference cluster.

When these diverse workloads compete for shared compute under highly variable traffic, the resulting resource contention is severe. Because agentic requests can vary substantially in reasoning complexity, treating them as static workloads can cause request queues to grow uncontrollably [8, 22, 44] and increased contention for KV cache. This leads to congestion and tail latency degradation for all requests.

Existing approaches have significant limitations when handling inference engine congestion. Rate-limiting quotas and admission control throttle or drop requests but cannot regulate how much computation an admitted request consumes. Elastic scaling provisions additional GPUs to add model replicas [36], but it takes minutes to bring new capacity online, often slower than the timescale over which congestion develops [6]. Scheduling policies [1, 28, 32, 40]

prioritize requests but cannot prevent queue buildup when the offered load exceeds processing capacity. In a nutshell, a common thread runs through these limitations: existing mechanisms control which requests are admitted, when they execute, or where they run, but treat each admitted request as a fixed-cost unit. Yet, this is precisely the dimension that has exploded with test-time scaling.

We propose CALM-MAS, a *congestion-aware* serving framework that closes this gap by dynamically adapting the test-time computation performed per admitted request. The key intuition behind CALM-MAS is to treat test-time computation as an *elastic* configuration that the system adjusts in response to load, rather than a static, load-agnostic parameter fixed at request time. Specifically, CALM-MAS operates along two control dimensions: *reasoning effort*, i.e., the number of thinking tokens allocated per request, and *agent architecture*, i.e., the number of collaborating MAS stages that decompose a task. Both dimensions impact the computational cost of a request. At the same time, their effect on response quality is task-dependent: not every problem benefits from deeper reasoning or additional agents. This makes them attractive control knobs: by treating them as elastic, dynamic configurations, CALM-MAS can gracefully degrade the computational footprint of LLM requests during congestion events and substantially reduce queue buildup, while restoring full quality when resources become available.

We evaluate CALM-MAS on a vLLM [16] backend serving concurrent MAS and non-MAS workloads under a uniform latency SLO. CALM-MAS reduces P99 end-to-end latency by 1.18× relative to a static MAS baseline and by 1.97× relative to per-client rate limiting with the accuracy degradation remaining confined to 6.1 percentage points (pps) relative to the unconstrained multi-agent configuration.

## 2 Background & Motivation

### 2.1 Serving Agentic LLM Workflows

The rapid deployment of LLMs in production has driven the development of dedicated inference serving systems such as vLLM [16], SGLang [46], and TGI [12]. These engines are workload-agnostic by design: they accept a prompt, dynamically allocate Key-Value (KV) cache memory during autoregressive decoding, and evict entries as needed using policies such as LRU, with no awareness of the higher-level task a request belongs to. This architecture was well matched to the traditional paradigm of single-turn, stateless requests, where each prompt maps to one bounded generation with a predictable compute and memory footprint.

However, the paradigm is shifting toward multi-turn, stateful LLM workloads. Multi-turn chatbots maintain conversation context across human interactions [20, 43]; agentic systems go further, using LLMs as central reasoning engines embedded within autonomous loops [38]. These workflows feature tool utilization [31], iterative self-correction [37], and multi-agent orchestration [41] to accomplish complex, long-running objectives. Multi-agent systems, in particular, are capable of achieving better response quality while consuming more KV cache [3]. This operational shift introduces three core architectural challenges that break the assumptions of traditional, stateless serving infrastructure:

1) ***Dynamic and unpredictable lifecycles.*** Single-turn requests have predictable compute profiles bounded by a maximum token count. In contrast, an autonomous agent may iterate for many rounds based on intermediate results and environment feedback, introducing high variance in request lifetimes and token generation volume [14].

2) ***Request amplification.*** A single user-facing MAS task decomposes into multiple dependent LLM calls (one per MAS stage). The serving engine sees a burst of correlated requests from one logical task, rather than a stream of independent queries [24, 42].

3) ***Stateful resource holding.*** Unlike single-turn requests, agentic workflows *(i)* may hold multiple concurrent cache slots due to parallel LLM calls within the same workflow, and *(ii)* retain those slots for the workflow's entire execution. This persistent state amplifies memory pressure and contention on the serving backend.

### 2.2 Congestion in LLM serving

LLM serving engines face two distinct pressure points under load. On the *prefill* side, when the incoming token rate exceeds prefill throughput, the waiting queue grows and time-to-first-token (TTFT) degrades. On the *decode* side, concurrent generations compete for finite KV cache memory; once capacity is saturated, the engine must preempt requests (e.g., via swap or recompute in vLLM [16]) or shrink the running batch, inflating time-between-tokens (TBT). Past a utilization threshold, both latencies degrade non-linearly with load [15, 27], creating *congestion*.

Congestion is not merely a transient response to overload: it can be self-sustaining. Preempted requests re-enter the queue carrying KV state that must be swapped back in or recomputed from scratch, paying a penalty proportional to their already-generated context. A short input burst can thus elevate effective load well beyond the burst's duration, producing a metastable regime in which queue depth and tail latency remain high after the arrival rate has subsided (quantified in § 4).

Agentic workloads are uniquely prone to entering this regime, with each property from § 2.1 contributing a distinct pathology. *Stateful resource holding* makes preemption disproportionately expensive, as evicting a multi-turn request discards or forces recomputation of accumulated context.

*Request amplification* converts a single user-facing spike into a burst of correlated, context-sharing calls that hit the engine in lockstep. And *unpredictable lifecycles* prevent the scheduler from sizing batches or admission decisions against any reliable estimate of remaining work. Together, these properties make agentic serving not merely susceptible to congestion, but liable to persisting in a congestion-prone operating regime.

## 2.3 Limitations of current approaches

We review previous work on handling overload in LLM serving and discuss shortcomings under persistent congestion. Current approaches operate on different levers—e.g., capacity, admission, timing, or model choice—but none modulates the *amount of test-time computation* a workflow performs once admitted, which is precisely the knob that dominates cost in reasoning models and multi-agent systems.

**Elastic scaling**. When compute demand exceeds the capacity of the serving system, a natural response is to provision additional GPUs. However, scaling cannot respond on the timescale of congestion events. First, the requested GPU type may be unavailable in the target cluster, in which case provisioning can take on the order of hours [35]. Second, even when hardware is available, several minutes elapse between the time a GPU is requested and the time it is ready to accept traffic, owing to container startup, CUDA initialization, and model weight loading [6, 23]. Congestion events that resolve within tens of seconds to a few minutes are therefore over before new capacity comes online.

**Rate limiting and admission control**. Rate-limiting quotas (commonly used by all major LLM API providers [2, 9, 26]) and admission policies [28] regulate *which* requests enter the system or at *what rate*, but treat every admitted request as a fixed-cost unit. The server has no mechanism to renegotiate once the request is admitted. A heavy MAS workflow consumes the same compute regardless of current load, potentially exacerbating congestion for all users. We demonstrate in § 4 that per-client rate-limiting leads to severe queue buildup and latency degradation under congestion.

**Scheduling and model routing**. Recent agentic serving frameworks expose workflow structure to the runtime and use it for scheduling and model selection. Scheduling policies [1, 17, 24, 28, 32, 34, 36, 44] decide *when* a request is served relative to others. However, during congestion, even the most efficient scheduling cannot prevent queue buildup if the offered load exceeds the backend's processing capacity. Alternatively, model routing policies [8, 25, 45] decide *which* model configuration serves a request. For example, Aragog [8] routes each workflow stage to the cheapest accuracy-preserving model configuration based on live system load. These systems control *which* resources serve a request but do not control *how much computation* a request performs on a given model.

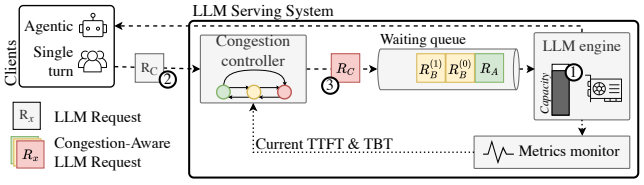


**Figure 1: *System overview.* Upon congestion (①), CALM-MAS controller adjusts the test-time computation knobs of incoming requests (②) to regulate the amount of admitted computation (③).**

# 3 CALM-MAS

Building on the previous observations, we instead propose to explicitly reduce congestion by regulating the *amount of test-time computation per admitted request*. Accordingly, we present CALM-MAS, a congestion-aware serving framework that dynamically adapts the computational behavior of inference workloads based on backend congestion signals.

## 3.1 Overview

Fig. 1 synthesizes the idea. A *metrics monitor* continuously tracks congestion signals from the serving engine, and a *congestion controller* reacts to these signals by adjusting two test-time scaling knobs—*reasoning budget* and *agent fan-out*—for the incoming request to dynamically modify its computational demands. During periods of low utilization, CALM-MAS prioritizes response quality by enabling more computationally intensive inference configurations. Under congestion, CALM-MAS progressively reduces the computational footprint of requests across the serving system, thereby lowering overall resource consumption. We discuss the congestion signals and detail the reaction mechanisms in the next sections.

Analogous to congestion-control mechanisms in computer networks, CALM-MAS does not eliminate SLO violations during transient overload; rather, it bounds their duration by actively driving the system back toward a sustainable operating point, while continuing to serve requests with the best possible accuracy given the current load.

**Congestion signals**. CALM-MAS leverages readily-available signals from the inference engine that reflect the current system load to detect backend saturation. *Time To First Token (TTFT)* measures the delay from the moment a request arrives at the serving engine, inclusive of any time spent waiting in the request queue, to the emission of the first generated token. As requests accumulate faster than the engine can process them, new arrivals wait longer before the prefill

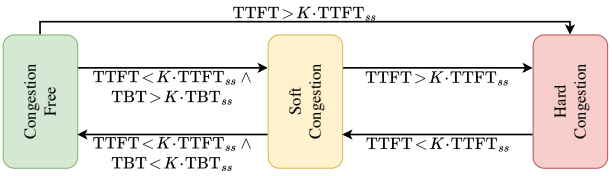


**Figure 2: Congestion control state machine. The controller transitions between three states based on observed TTFT and TBT relative to steady-state baselines scaled by threshold $K$.**

phase begins. A rising TTFT is therefore an early indicator of backend overload. *Time Between Tokens (TBT)* measures inter-token latency during autoregressive generation. Elevated TBT indicates that the decoding pipeline is saturated, for instance because large KV caches from many concurrent requests compete for GPU memory bandwidth.

Using both signals allows the controller to distinguish prefill-dominated congestion (high TTFT) from decoding-dominated congestion (high TBT) and to respond proportionally with corresponding congestion reduction mechanisms. We elaborate on this in the next section.

## 3.2 Congestion reaction mechanisms

The controller maps congestion signals to three operating states (Fig. 2), each enforcing progressively stricter adjustments to the compute requirements of new requests by acting on the two test-time scaling knobs introduced above.

We note that these knobs are typically hard to define a priori for a given task by users. Moreover, the optimal configuration for a given task may vary with the current load on the system: a large reasoning budget may be beneficial when the system is uncongested, but lead to severe latency degradation when the system is congested. We therefore treat these knobs as dynamic configurations that the system adjusts on the fly based on current congestion levels, rather than as static parameters that users must set at request time.

The provider defines a steady-state baseline value for each signal, which represents the expected value of the metric when the system is fully utilized but not yet congested (we suggest a heuristic method in § 4). Thresholds for each signal are then set relative to these baselines and can depend on the SLOs. In this paper, we use a simple multiplicative factor $K$ to define the thresholds, and reserve more sophisticated mechanisms for future work.

**Congestion-free regime**. Both TTFT and TBT remain below the congestion threshold. The system honors the full user-declared reasoning budget and the native multi-agent architecture.

**Soft congestion regime**. TBT exceeds its threshold while TTFT remains acceptable, indicating decoding-phase saturation. Mirroring multiplicative decrease in network congestion control, the system reduces the reasoning budget of subsequently admitted requests by a *multiplicative* factor $\beta$ (set to 0.1 in our experiments), reducing thinking tokens to relieve generation-side pressure. A proportional cut is task-agnostic and yields proportional fairness across users.

**Hard congestion regime**. TTFT exceeds its threshold, signaling prefill phase overhead. The controller therefore applies a workflow-specific collapse operator that maps the multi-agent workflow to a registered single-agent fallback. Depending on the workflow, this fallback may execute the original task prompt, a merged prompt summarizing the intended stages, or a designated fallback role. Thus, collapse is not assumed to be universal: applications must define the degraded execution path for workflows such as debate or critique pipelines that lack a natural single-agent form. The controller also disables reasoning tokens, removing their computational cost and reducing decoding pressure.

These actions eliminate request amplification and stateful resource holding while lowering per-request compute. The asymmetry between the soft and hard regimes is deliberate, motivated by the self-reinforcing nature of congestion. When both metrics violate the desired range, we take more drastic measures to escape congestion. Finer gradations of intervention indexed to the extent of SLO violation are a natural extension we leave to future work.

***Remark #1: MAS scope***. CALM-MAS targets the subclass of MAS workloads where multi-agency primarily serves as a test-time scaling technique for accuracy (e.g., debate, ensemble, multi-perspective review), rather than workloads where agents play semantically distinct roles with specialized tool access or capabilities; for the latter, collapsing to a single agent is not functionally equivalent and lies outside the scope of this work. Nevertheless, unlike early approaches valid only for a limited subset of tasks (e.g., BeLLMan [29]), CALM-MAS generalizes across a broad class of tasks.

***Remark #2: Per-task contract and in-flight requests***. Congestion control decisions apply only to *subsequently admitted* tasks and do not affect in-flight requests, e.g., requests belonging to a previously admitted MAS task. We deliberately keep the execution configuration of in-flight tasks unchanged, since modifying the agent topology mid-task may invalidate accumulated workflow state or waste compute and memory already invested. We show in § 4 that this does not undermine the effectiveness of CALM-MAS: the bulk of congestion relief comes from the reduced specification applied to the stream of newly arriving requests, which dominates aggregate load during sustained congestion episodes. Adapting ongoing workflows, e.g by modifying their reasoning

effort or execution topology while perserving accumulated state, is an interesting direction we leave to future work.

**Use of Artifical Intelligence**. AI-assisted tools were used during implementation to help identify and debug software errors. All resulting code changes were reviewed and validated by the authors. AI tools were not used to design the experiments, select the evaluation methodology, or interpret the experimental results presented in this work.

## 4 Evaluation

We run all experiments using the `Qwen3.5-35B-A3B` model in FP8 precision deployed on a single NVIDIA A100 GPU and served by the vLLM [16] inference engine. We consider two heterogeneous workloads: *single-turn* tasks and *agentic* tasks. Single-turn tasks correspond to conventional single-request inference workloads derived from the GPQA [30], a benchmark comprising graduate-level science questions designed to test advanced reasoning. For agentic tasks we consider a benchmark consisting of 50 requests obtained by sampling 25 requests from both BrowseComp+ [5] and PlanCraft [7]. Task arrivals follow a non-stationary Poisson process with time-varying rates. During burst periods, agentic and single-turn tasks arrive at 2 and 5 tasks/min, respectively; during low-load periods, both rates are reduced by a factor of three.

**Steady-state configuration**. The steady state represents the operating point at which the system is fully utilized but not yet overloaded: all three key resources—prefill throughput, decode throughput, and KV-cache—are simultaneously saturated without any single bottleneck dominating. We identify this point empirically by profiling the system under pure single-turn and pure agentic workloads to establish each workload's saturation limit, then evaluating mixed workloads near the combined saturation boundary. Each candidate mix is scored as $S = (u_{\text{in}}\, u_{\text{out}}\, u_{\text{kv}})^{1/3} - \lambda \sum_{i \neq j} |u_i - u_j|$, where $u_{\text{in}}, u_{\text{out}}, u_{\text{kv}} \in [0, 1]$ are the input throughput, output throughput, and KV-cache utilization, each normalized by its observed maximum. The first term rewards high joint utilization; the second is a regularization term, weighted by $\lambda$ (set to 0.15 in all experiments), that penalizes imbalance across resource dimensions. The resulting values serve as the baseline thresholds for the congestion controller.

**Congestion control tames TTFT spikes**. We examine the effects of CALM-MAS's congestion control mechanisms on TTFT in Fig. 3. When considering the initial stages of the experiments, $K$=1 causes the congestion controller to enter the soft-congestion state after just 1.5s, whereas $K$=5 sustains the congestion-free state for approximately 5s. At $K$=10, the system remains congestion-free for nearly 20s, during which the request queue grows to the point that TTFT spikes, forcing a transition to the hard-congestion state. As a result, the peak TTFT (i.e., the largest TTFT we register

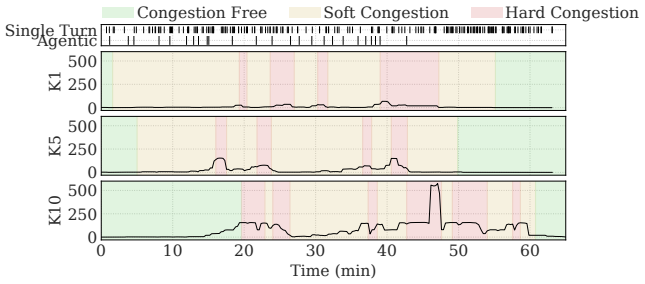


**Figure 3: Evolution of the congestion states in CALM-MAS and its impact on TTFT for varying $K$. The top bar indicates arrival times for the two types of requests.**

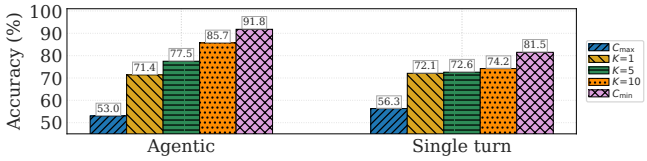


**Figure 4: CALM-MAS's accuracy vs baselines.**

throughout the course of the experiment) values are 67.8, 152.9, and 574.0 for $K$=1, 5, and 10, respectively. Running the same workload without constraints on reasoning and architecture, regardless of system load, yields a peak TTFT of 2524.8 (omitted from the figure for clarity), indicating that CALM-MAS reduces peak TTFT by up to 94% in this scenario—a reduction that grows as $K$ decreases.

**Impact on accuracy**. To isolate the impact of the congestion control mechanism on accuracy, we compare CALM-MAS against two baselines: $C_{\text{min}}$, which never applies constraints on reasoning budget or agent architecture; and $C_{\text{max}}$, which enforces maximum constraints on all requests. These correspond to the congestion-free and hard-congestion states of CALM-MAS, respectively.

Fig. 4 shows that, as expected, the $C_{\text{min}}$baseline consistently achieves the highest accuracy, while reducing test-time scaling as in $C_{\text{max}}$causes a marked drop. Thus, the parameter $K$ serves as an effective control knob mediating the latency-accuracy trade-off: as shown above, lower $K$ reduces peak TTFT, but at the cost of accuracy, since more requests are stripped of reasoning capabilities to keep the system load under control, increasing the number of inaccuracies. Conversely, higher $K$ mitigates this effect; at $K$=10, the accuracy degradation remains confined to 6.1 and 7.3 pps for agentic and single-turn requests, respectively.

**End-to-end performance**. On top of the previous baselines ($C_{\text{max}}$ and $C_{\text{min}}$), we compare CALM-MAS against per-client rate limiting. Each client is associated with a dedicated rate limiter upstream of the inference engine. All rate limiters have a pre-defined credit budget $c$. Each request $r$ deducts

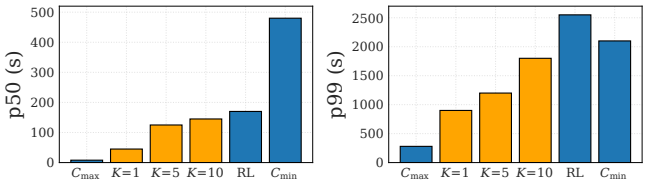


**Figure 5: E2E latency (50th and 99th percentile).**

credit from the budget as $c \leftarrow c - in(r) - 2 \times \texttt{max\_out\_tokens}$, where $in(r)$ is the prompt length in tokens, `max_out_tokens` is the maximum output length, and the factor 2 accounts for the higher cost of the auto-regressive decoding phase relative to prefill, similar to [32]. If credits are not available, the request is deferred until sufficient credits are replenished. Credits are re-filled at a rate derived from the steady-state, and unused credits[1] are refunded upon request completion. We set `max_out_tokens` = 16,000 and initialize each client's budget to $2 \times$ `max_out_tokens`.

We compare in Fig. 5 the baselines against varying values of $K$ with respect to the end-to-end (E2E) latency, defined as the wall-clock time elapsed from task submission to response receipt. As expected, the $C_{\text{max}}$baseline achieves the lowest E2E latency, followed by CALM-MAS with increasing $K$. Rate limiting and $C_{\text{min}}$differ only in that the former may defer requests until sufficient credits are available; this deferral prevents agentic tasks from monopolizing system resources, improving median (p50) E2E latency over $C_{\text{min}}$. At the 99th percentile, however, the effect is reversed: agentic tasks inevitably appear in the tail, and deferring them increases E2E latency relative to $C_{\text{min}}$.

## 5 Open challenges

This work is a first step towards congestion-aware LLM serving, but several challenges remain open.

**Control policy design**. In CALM-MAS, the relationship between computational budget and task accuracy depends on the workload. Systematically exploring the cost-accuracy Pareto front across diverse task distributions is needed to inform tighter control policies. Currently, the congestion controller operates on three coarse-grained regimes (§ 3.2), where both the state transitions and the congestion reaction policies are fixed. This design is simple and effective, however, more sophisticated mechanisms could yield better trade-offs. Future work will explore finer-grained methods similar to self-adaptive [21] and budget-aware reasoning [39] to regulate the reasoning effort. These methods are currently agnostic to the underlying serving infrastructure, and an open challenge is how to condition them on congestion signals coming from the serving engine. Similarly, methods for automatic MAS architecture discovery [11, 13] and meta-harness optimization [18] can also be conditioned on a congestion signal to synthesize agent control loops tailored to different load conditions.

**Service contract and quality degradation**. CALM-MAS introduces a quality trade-off for requests admitted *after* congestion is detected, while not modifying in-flight workflows (§ 3.2). The question of who bears the cost of quality degradation deserves further attention. While this preliminary work contributes primarily to the technical feasibility of congestion-aware serving, we envision that LLM service providers may explicitly expose the quality trade-off to users (similar to prior work in serverless computing on flexible resource-allocation interfaces [4]). For example, a provider could offer cheaper APIs in exchange for accepting reduced test-time computation during congestion, while still giving users the option to wait longer and be served under the native configuration. The definition of the corresponding interfaces for pricing, admission, and fallback strategies remains an open problem.

**LLM serving generality**. We evaluated CALM-MAS in a co-located prefill-decode configuration, and using a single inference serving engine. Extending CALM-MAS to multi-instance model serving or architectures with disaggregated prefill and decode (PD disaggregation) may require coordinating congestion signals across serving components, and represents a natural next step for our work. Lastly, our vision for CALM-MAS is not as a standalone component, but as a general congestion-control layer that should coexist with other LLM serving components, such as batching, scheduling, routing, admission control, and load balancing. Understanding the interplay between CALM-MAS and these techniques, and integrating into a unified framework, is another key challenge ahead.

## 6 Conclusion

CALM-MAS is a congestion-aware serving framework that treats test-time computation as an elastic knob rather than a fixed parameter. Unlike previous work, CALM-MAS regulates the amount of computation per admitted request based on real-time congestion signals. Under congestion, CALM-MAS can substantially reduce tail latency and queue buildup while preserving most of the accuracy of native task configurations. Realizing our vision fully will require addressing several challenges: generalizing the control policy, resolving how load-dependent quality degradation should be exposed through service contracts, and evaluating the system under more realistic LLM serving architectures.

[1]The actual output tokens $out(r)$ for a request is known only at completion. We refund the unused credits as $c \leftarrow c + 2 \times (\texttt{max\_out\_tokens} - out(r))$.

## References


[1] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. In *OSDI*.

[2] Anthropic. 2026. Rate Limits. https://docs.anthropic.com/en/api/rate-limits. Accessed: 2026-05-20.

[3] Zhuohang Bian, Feiyang Wu, Chengrui Zhang, Hangcheng Dong, Yun Liang, and Youwei Zhuo. 2026. TokenDance: Scaling Multi-Agent LLM Serving via Collective KV Cache Sharing. arXiv:2604.03143 [cs.DC]

[4] Muhammad Bilal, Marco Canini, Rodrigo Fonseca, and Rodrigo Rodrigues. 2023. With Great Freedom Comes Great Opportunity: Rethinking Resource Allocation for Serverless Functions. In *EuroSys*.

[5] Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Sahel Sharifymoghaddam, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. 2025. BrowseComp-Plus: A More Fair and Transparent Evaluation Benchmark of Deep-Research Agent. arXiv:2508.06600 [cs.CL]

[6] Rob Cronin. 2026. Dissecting LLM Container Cold-Start: Where the Time Actually Goes. https://techcommunity.microsoft.com/blog/linuxandopensourceblog/dissecting-llm-container-cold-start-where-the-time-actually-goes/4508831

[7] Gautier Dagan, Frank Keller, and Alex Lascarides. 2025. Plancraft: an evaluation dataset for planning with LLM agents. arXiv:2412.21033 [cs.CL]

[8] Yinwei Dai, Zhuofu Chen, Anand Iyer, and Ravi Netravali. 2025. Aragog: Just-in-Time Model Routing for Scalable Serving of Agentic Workflows. arXiv:2511.20975 [cs.DC]

[9] Google. 2026. Rate Limits. https://ai.google.dev/gemini-api/docs/rate-limits. Accessed: 2026-05-20.

[10] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In *ICLR*.

[11] Shengran Hu, Cong Lu, and Jeff Clune. 2025. Automated Design of Agentic Systems. arXiv:2408.08435 [cs.AI]

[12] Hugging Face. 2025. *Text Generation Inference*. https://github.com/huggingface/text-generation-inference Accessed: 2026-05-19.

[13] Zixuan Ke, Austin Xu, Yifei Ming, Xuan-Phi Nguyen, Caiming Xiong, and Shafiq Joty. 2025. MAS-ZERO: Designing Multi-Agent Systems with Zero Supervision. In *Workshop on Scaling Environments for Agents*.

[14] Jiin Kim, Byeongjun Shin, Jinha Chung, and Minsoo Rhu. 2026. The cost of dynamic reasoning: Demystifying ai agents and test-time scaling from an ai infrastructure perspective. In *HPCA*.

[15] Abhishek Vijaya Kumar, Gianni Antichi, and Rachee Singh. 2025. Aqua: Network-Accelerated Memory Offloading for LLMs in Scale-Up GPU Domains. In *ASPLOS*.

[16] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *SOSP*.

[17] Marco Laju, Donghyun Son, Saurabh Agarwal, Nitin Kedia, Myungjin Lee, Jayanth Srinivasa, and Aditya Akella. 2026. Nalar: An agent serving framework. arXiv:2601.05109 [cs.DC]

[18] Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. 2026. Meta-Harness: End-to-End Optimization of Model Harnesses. In *COLM*.

[19] Junyan Li, Wenshuo Zhao, Yang Zhang, and Chuang Gan. 2025. Steering LLM Thinking with Budget Guidance. arXiv:2506.13752 [cs.CL]

[20] Yubo Li, Xiaobin Shen, Yidi Miao, Xinyu Yao, Xueying Ding, Ramayya Krishnan, and Rema Padman. 2026. Beyond Single-Turn: A Survey on Multi-Turn Interactions with Large Language Models. arXiv:2504.04717 [cs.CL]

[21] Zheng Li, Qingxiu Dong, Jingyuan Ma, Di Zhang, Kai Jia, and Zhifang Sui. 2026. SelfBudgeter: Adaptive Token Allocation for Efficient LLM Reasoning. arXiv:2505.11274 [cs.AI]

[22] Tengxiao Liu, Zifeng Wang, Jin Miao, I-Hung Hsu, Jun Yan, Jiefeng Chen, Rujun Han, Fangyuan Xu, Yanfei Chen, Ke Jiang, Samira Daruki, Yi Liang, William Yang Wang, Tomas Pfister, and Chen-Yu Lee. 2026. Budget-Aware Tool Use Enables Effective Agent Scaling. arXiv:2511.17006 [cs.AI]

[23] Chiheng Lou, Sheng Qi, Chao Jin, Dapeng Nie, Haoran Yang, Yu Ding, Xuanzhe Liu, and Xin Jin. 2025. HydraServe: Minimizing Cold Start Latency for Serverless LLM Serving in Public Clouds. arXiv:2502.15524 [cs.DC]

[24] Michael Luo, Xiaoxiang Shi, Colin Cai, Tianjun Zhang, Justin Wong, Yichuan Wang, Chi Wang, Yanping Huang, Zhifeng Chen, Joseph E. Gonzalez, and Ion Stoica. 2026. Agentix: An Efficient Serving Engine for LLM Agents as General Programs. In *NSDI*.

[25] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. 2025. RouteLLM: Learning to Route LLMs from Preference Data. In *ICLR*.

[26] OpenAI. 2026. Rate limits. https://platform.openai.com/docs/guides/rate-limits. Accessed 2026-05-20.

[27] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Í textasciitilde nigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *ISCA*.

[28] Archit Patke, Dhemath Reddy, Saurabh Jha, Haoran Qiu, Christian Pinto, Chandra Narayanaswami, Zbigniew Kalbarczyk, and Ravishankar K. Iyer. 2024. Queue Management for SLO-Oriented Large Language Model Serving. In *SoCC*.

[29] Tella Rajashekhar Reddy, Atharva Deshmukh, Karan Tandon, Rohan Gandhi, Anjaly Parayil, and Debopam Bhattacherjee. 2025. BeLLMan: Controlling LLM Congestion. arXiv:2510.15330 [cs.DC]

[30] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. GPQA: A Graduate-Level Google-Proof Q&A Benchmark. arXiv:2311.12022 [cs.AI]

[31] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *NeurIPS*.

[32] Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E. Gonzalez, and Ion Stoica. 2024. Fairness in Serving Large Language Models. In *OSDI*.

[33] Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Parameters for Reasoning. In *ICLR*.

[34] Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. 2025. Preble: Efficient Distributed Prompt Scheduling for LLM Serving. In *ICLR*.

[35] Foteini Strati, Zhendong Zhang, George Manos, Ixeia Sánchez Périz, Qinghao Hu, Tiancheng Chen, Berk Buzcu, Song Han, Pamela Delgado, and Ana Klimovic. 2025. Sailor: Automating Distributed Training over Dynamic, Heterogeneous, and Geo-distributed Clusters. In *SOSP*.

[36] Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. 2024. Llumnix: Dynamic Scheduling for Large Language Model Serving. In *OSDI*.

[37] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Voyager: An Open-Ended Embodied Agent with Large Language Models. *Trans. Mach. Learn. Res.* (2024).

[38] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024. A survey on large language model based autonomous agents. *Frontiers Comput. Sci.* (2024).

[39] Hao Wen, Xinrui Wu, Yi Sun, Feifei Zhang, Liye Chen, Jie Wang, Yunxin Liu, Yunhao Liu, Ya-Qin Zhang, and Yuanchun Li. 2025. BudgetThinker: Empowering Budget-aware LLM Reasoning with Control Tokens. arXiv:2508.17196 [cs.LG]

[40] Bingyang Wu, Yinmin Zhong, Zili Zhang, Shengyu Liu, Fangyue Liu, Yuanhang Sun, Gang Huang, Xuanzhe Liu, and Xin Jin. 2026. FastServe: Iteration-Level Preemptive Scheduling for Large Language Model Inference. In *NSDI*.

[41] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. 2024. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *COLM*.

[42] Yuxing Xiang, Xue Li, Kun Qian, Yan Zhang, Wenyuan Yu, Ennan Zhai, Xin Jin, and Jingren Zhou. 2026. ServeGen: Workload Characterization and Generation of Large Language Model Serving in Production. In *NSDI*.

[43] Chen Zhang, Xinyi Dai, Yaxiong Wu, Qu Yang, Yasheng Wang, Ruiming Tang, and Yong Liu. 2025. A Survey on Multi-Turn Interaction Capabilities of Large Language Models. arXiv:2501.09959 [cs.CL]

[44] Wei Zhang, Zhiyu Wu, Yi Mu, Rui Ning, Banruo Liu, Nikhil Sarda, Myungjin Lee, and Fan Lai. 2026. JITServe: SLO-aware LLM Serving with Imprecise Request Information. In *NSDI*.

[45] Yunxuan Zhang, Hao Wang, Han Tian, Liu Yang, Xudong Liao, Wenxue Li, Ping Yin, Bowen Liu, and Kai Chen. 2026. MFS: An Efficient Model Family Serving System for LLMs. In *EuroSys*.

[46] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. 2024. SGLang: Efficient Execution of Structured Language Model Programs. In *NeurIPS*.