H-MAS: Hierarchical Multi-Agent Scheduling for Multi-Tenant LLM Serving

Yuhan Liu[1,2], **Cong Xu**[2*], **Qi Jia**[2], **Yihua Wang**[1,2], **Feiyu Chen**[1,2], **Liang Jin**[2], **Lu Liu**[2], **Yaqian Zhao**[2], **Yuting Ding**[3], **Xiang Li**[4]

[1]Beijing Normal University [2] Inspur (Beijing) Electronic Information Industry Company

[3] Southeast University [4] Key Laboratory of Computing Power Network and Information Security, Ministry of Education, Qilu University of Technology (Shandong Academy of Sciences)

zxyuhan@mail.bnu.edu.cn, xucong@ieisystem.com

Abstract

Multi-tenant Model-as-a-Service (MaaS) LLM serving must maintain stringent quality of service (QoS) despite heterogeneous requests competing for constrained GPU resources. In practice, MaaS workloads exhibit non-stationarity across multiple time scales, including request bursts, request-composition drift, and persistent workload shifts. Because workloads change across multiple time scales, existing request schedulers often rely on a single fixed policy (e.g., First-Come-First-Served, FCFS) that remains unchanged at runtime, which can lead to unstable QoS. We propose H-MAS, a hierarchical multi-agent scheduler that operates in a layered closed loop: a perception/prediction layer infers lightweight request attributes and cost signals; a feedback layer summarizes runtime metrics into short- and long-horizon QoS states; a hierarchical control layer updates the active scheduling policy over longer horizons and tunes execution parameters over shorter horizons; and an execution layer applies these decisions during inference. Experiments with load scaling and Azure-trace replays show that H-MAS achieves 1.2×–3.0× higher Goodput than SGLang and vLLM, and maintains more stable QoS under workload drift, diverse request lengths and heterogeneous SLO targets.

1 Introduction

Large language models (LLMs) now power many cloud applications, including chatbots, Retrieval-Augmented Generation (RAG), and agentic workflows (Park et al., 2025; Lewis et al., 2020; Yao et al., 2023; Liu et al., 2025; Lu et al., 2026). To provide these capabilities, organizations increasingly adopt the Model-as-a-Service (MaaS) paradigm, where a centralized provider hosts models for multiple tenants sharing a common GPU fleet (Agrawal et al., 2024; Chen et al., 2024; Choi et al., 2022). In this setting, ensuring Quality of Service (QoS) is challenging because heterogeneous requests from diverse users fiercely compete for limited GPU resources (Lazuka et al., 2024). Consequently, an effective scheduler is crucial for maintaining QoS stability by dynamically adjusting key control knobs, such as request ordering and concurrency.

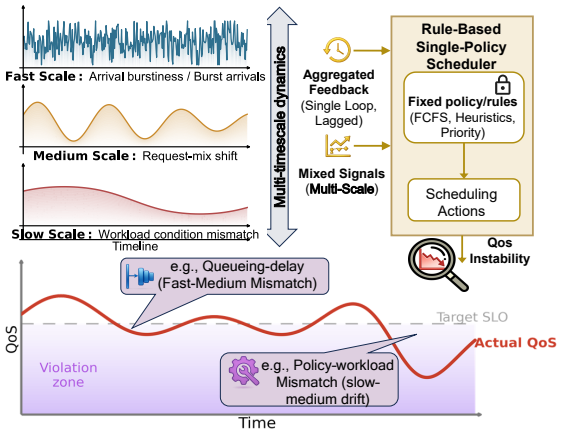


Figure 1: QoS instability induced by multi-scale workload dynamics under rule-based single-loop scheduling.

Most existing LLM serving systems adopt a single-policy, rule-based scheduler whose policy and parameters are configured before deployment and remain fixed at runtime (Kwon et al., 2023; Zheng et al., 2024; Gong et al., 2025; Kolluru, 2025; Liu et al., 2025). This paradigm implicitly assumes relatively stable workloads. However, real-world MaaS workloads are non-stationary: request arrivals (Requests Per Second, RPS) can be bursty, request lengths vary widely, and requests have different service-level objectives (SLOs) (Zhang et al., 2025). Bursts increase queueing delay, while shifts toward longer requests increase the GPU service time per request; both make SLO violations more frequent and reduce throughput. Maintaining stable QoS therefore requires a scheduler that can satisfy per-request SLO targets (Time-To-First-Token,

*Corresponding author

Findings of the Association for Computational Linguistics: ACL 2026, pages 39051–39071

TTFT / Time-Per-Output-Token, TPOT) while sustaining system-level efficiency (throughput). However, a fixed single-policy scheduler is not robust under such non-stationary workloads: it cannot adjust its control knobs to current workload conditions, leading to QoS instability.

This QoS instability stems from a mismatch between non-stationary workload dynamics and a fixed single-policy scheduler paradigm, as illustrated in Fig. 1. The workload changes, but a single-policy scheduler keeps the same scheduling policy at runtime. In particular, MaaS workload dynamics manifest over multiple time scales. We summarize three time scales as follows:

(i) At the **fast time scale (ms–s)**, arrival bursts can grow the queue within milliseconds to seconds, sharply increasing waiting time. Mitigation requires fast adjustments in request ordering and short-horizon concurrency control, which a fixed policy cannot react to effectively.

(ii) At **medium time scale (s–min)**, the request composition changes over seconds to minutes (e.g., long requests increase). Because the policy is fixed at runtime, the scheduler keeps using the old assumptions and cannot lower concurrency when longer requests become common.

(iii) At **slow time scale (min and beyond)**, sustained load conditions can persist for minutes or longer (e.g., the average RPS stays high during a peak period). Since the scheduling policy is fixed at runtime, the scheduler cannot switch to a more suitable scheduling policy for this persistent workload condition.

To address this multi-timescale mismatch, we present **H-MAS**, a hierarchical multi-agent scheduler for multi-tenant MaaS. H-MAS organizes scheduling into a layered closed loop. A perception and prediction layer extracts structured request attributes and execution cost estimates from incoming prompts. A feedback layer aggregates runtime signals into short- and long-horizon views of system state. Conditioned on these states, a hierarchical control layer separates long-horizon scheduling policy selection from short-horizon execution control. Finally, an execution layer enforces the selected scheduling structure and control parameters at fine granularity during inference. By coordinating these layers, H-MAS enables workload-adaptive scheduling that stabilizes QoS under non-stationary, multi-timescale workloads (Wooldridge, 2009; Xi et al., 2023). Overall, we make the following contributions:

- We characterize multi-timescale dynamics in multi-tenant MaaS LLM workloads and relate them to QoS instability.
- We propose **H-MAS** that decouples slow scheduling adaptation from fast execution regulation, guided by prediction and runtime QoS feedback.
- We implement H-MAS and evaluate it across multiple LLMs and workloads, demonstrating practicality and effectiveness.

2 Background & Motivating Analysis

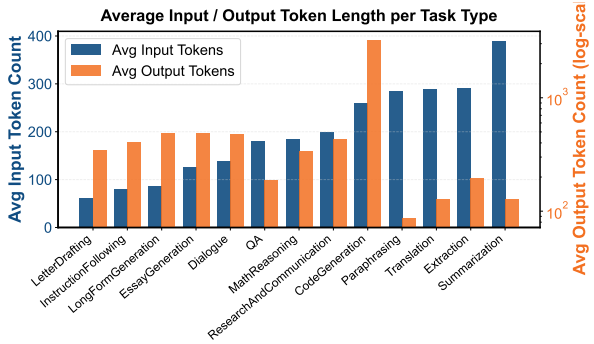
Our study is motivated by the inherent multi-scale dynamics of multi-tenant MaaS workloads.

Real-world LLM serving workloads are non-stationary and heterogeneous across time scales and request structures (Patel et al., 2024; Lazuka et al., 2024).

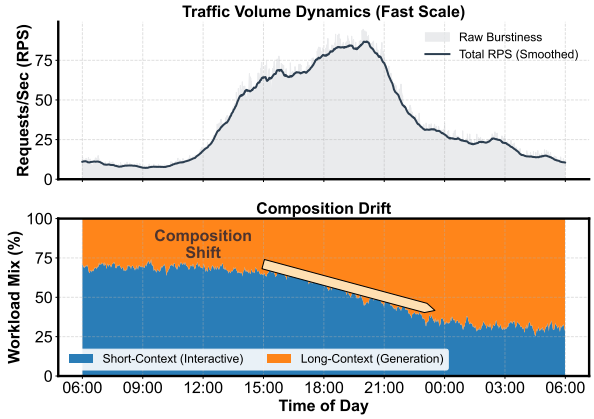
Fast scale: second-level bursts in arrival rate (Figure 2b, top). The Azure trace shows strong second-level jitter in requests-per-second (RPS) (Qiu et al., 2025). While the average rate is 45.15 RPS, the peak reaches 176 RPS, with a maximum 1-second step change of 140 RPS and a 5-second burst of up to 167 additional requests. These bursts quickly build up a queue backlog, causing sharp spikes in waiting time and short-term TTFT/SLO degradation.

Medium scale: drift in workload composition (Figure 2a and Figure 2b, bottom). At the minute scale, the Azure trace shows clear drift toward long requests. The long-request fraction varies by up to 0.1565 within a 5-minute window (min–max), with a maximum minute-level step change of 0.0874. Meanwhile, the P_{95} token-load multiplier reaches $1.21\times$ and the P_{95} input length rises to 5894 tokens. These shifts increase TPOT and reduce token throughput over minute-scale intervals when a scheduler keeps an unchanged execution setting.

Slow scale: structural drift beyond minutes. We analyze a production trace containing 1,048,576 requests (Stojkovic et al., 2025). Over hours to days, the share of short-context requests (0–512 tokens) ranges from 20.3% to 26.2%, while long-context requests (2048–4096 tokens) account for 22.3%–24.3% of the workload. Long contexts (>4096 tokens) also emerge intermittently, reaching a peak share of 9.2% on certain days. These slow shifts change the baseline token and memory pressure of the serving system, so long-window performance metrics (e.g., TPOT, throughput) can drift across periods, motivating long-horizon scheduling adaptation.



(a) Input/output token length statistics by task type.



(b) Traffic dynamics & composition drift

Figure 2: Workload characteristics across task-level input/output lengths and multi-scale traffic dynamics.

One key motivation for our design is the **structural limitations of single-policy scheduling**. Many existing LLM serving systems adopt a single-policy design in which the scheduling policy is fixed, and runtime signals (queue states, workload changes, and QoS statistics) are handled by the same decision logic at runtime (Kwon et al., 2023; Zheng et al., 2024; Gong et al., 2025; Kolluru, 2025). Even when thresholds or weights are retuned, the underlying policy logic (which signals are used, how ordering is computed, and how concurrency is set) remains unchanged (Oh et al., 2024).

This leads to two structural limitations. **(i) Structural rigidity.** The scheduling structure and the ordering/concurrency rule stay fixed, so when the workload shifts (e.g., longer requests become common), the scheduler still follows the same structure and rule. **(ii) Flattened multi-scale processing.** The same decision logic is used for all time scales, so it cannot handle both short bursts and longer-lasting changes equally well, leading to persistent scheduling policy mismatch across workload phases (Sun et al., 2024).

Another key motivation for our design is the need for **hierarchical multi-agent scheduling.** In MaaS, requests differ widely in length and latency objectives, while the workload evolves across multiple time scales. Effective scheduling therefore requires jointly leveraging *request-level* attributes (e.g., length, SLO targets) and *system-level* runtime signals (e.g., queueing, utilization, QoS feedback), and translating them into control actions. Multi-agent systems are suited to this setting because they enable role specialization and explicit information exchange: different agents can focus on heterogeneous signals and operate at different update cadences, while coordinating through shared state and feedback to produce coherent scheduling actions. We therefore adopt a multi-agent design (Wooldridge, 2009; Xi et al., 2023).

3 H-MAS: Adaptive Hierarchical Multi-Agent Scheduling

The term agent is used here in an operational sense: each decision entity is stateful, role-specialized, and updated on its own control horizon(Russell and Norvig, 2010; Weiss, 1999; Senehi et al., 1994). In H-MAS, this applies to perception/prediction, QoS feedback aggregation, long-horizon structural adaptation, and short-horizon execution regulation. H-MAS is a hierarchical multi-agent scheduler for MaaS LLM serving that decouples scheduling policy updates from fine-grained execution control in a closed loop. As shown in Fig. 3, H-MAS consists of four layers: perception & prediction, feedback, hierarchical control, and execution. The perception/prediction layer produces per-request features $\mathbf{x}_r$ and cost proxies. The feedback layer uses a QoS Agent to aggregate runtime logs into multi-horizon system states ($V_{\text{long}}, V_{\text{short}}$), which drive both Meta Controller selection ($\mathcal{A}_t$) and Statera Agent updates ($\mathcal{C}_t = (N_{\text{run}}, \mathcal{B}_{\text{token}}, \text{admission})$). Perception & Prediction runs asynchronously and is typically masked by backend queueing; a timing breakdown is in Appendix E.4.

3.1 Design Space and Modeling

We model H-MAS over decision epochs t: Perception/Prediction produces $\mathbf{x}_r$, the Meta Controller selects $\mathcal{A}_t \in \mathbf{\Omega}$ from V_{long}, and the Statera Agent updates $\mathcal{C}_t$ from V_{short}. For the complete notation, please refer to Appendix B.

3.1.1 Hierarchical Design Space

We summarize the state, structural, and execution-control spaces below.

State Space ($\mathcal{S}$): Constructed by the Perception, Prediction, and QoS agents, $\mathcal{S}$ transforms raw requests and logs into structured signals. The Perception agent extracts task types τ_r; the Prediction agent estimates service-time proxies $T_{tot}^{(r)}$; and the QoS agent aggregates system views V_{long} and collects calibration errors to update the bucketed error distribution D_b used by conformal calibration. These signals provide inputs for higher-layer decisions.

Structural Directive Space ($\mathcal{D}_{\text{algo}}$): The Meta Controller selects an active queue structure / ordering rule $\mathcal{A}_t \in \mathbf{\Omega}$ (e.g., FCFS, EDF, WFQ/Stride, Two-lane), and the execution layer instantiates $\mathcal{A}_t$ to order requests.

Execution Control Space ($\mathcal{D}_{ctrl}$): Managed by the Statera Agent, $\mathcal{D}_{ctrl}$ regulates execution through a small set of runtime controls. It enforces safe admission, suppresses jitter, and maintains utilization stability. In H-MAS, $\mathcal{C}_t$ is instantiated as ($N_{\text{run}}(t)$, $\mathcal{B}_{\text{token}}(t)$, $\hat{l}_{upper}$-based admission).

3.1.2 Problem Formulation

We model H-MAS as a stochastic control problem where $\mathcal{A}_t$ and $\mathcal{C}_t$ are updated over time under memory-safety constraints. We maximize weighted goodput, counting request r only if it meets both TTFT and TPOT SLOs. At epoch t, the system state is $s_t = (V_{\text{long}}(t), V_{\text{short}}(t))$, and the action is $a_t = (\mathcal{A}_t, \mathcal{C}_t)$, applied to requests annotated with $\mathbf{x}_r$. We let t index scheduling cycles; the Meta Controller updates $\mathcal{A}_t$ every ΔT_{meta} seconds, while the Statera Agent updates $\mathcal{C}_t$ every cycle.

$$\max \mathbb{E}\left[\sum_t \sum_{r \in \mathcal{R}_t^{\text{done}}} \text{Utility}(r) \cdot \mathbb{I}[\mathsf{OK}(r)]\right],$$
$$\mathsf{OK}(r): \; t_{ttft}^{(r)} \le SLO_{ttft}^{(r)} \;\wedge\; t_{tpot}^{(r)} \le SLO_{tpot}^{(r)}. \tag{1}$$

where $\mathcal{R}_t^{\text{done}}$ is the set of requests completed during epoch t, and $\text{Utility}(r)$ is class-dependent (from τ_r). $\mathbb{E}[\cdot]$ is taken over the randomness in request arrivals, request characteristics, and runtime variability.

Conservative Memory Safety. To bound OOM risk, we restrict admission using a calibrated upper-bound length estimate $\hat{l}_{upper}^{(r)}$ (Sec. 3.4): Let $Q_{run}(t)$ denote the set of admitted in-flight requests at epoch t. Admission satisfies:

$$\sum_{r \in Q_{\text{run}}(t)} m_{\text{KV}}\left(\hat{l}_{\text{upper}}^{(r)}\right) \le M_{\text{cap}}\,(1 - \delta_{\text{frag}}). \tag{2}$$

where M_{cap} is the available memory budget and δ_{frag} is a conservative fragmentation margin, $m_{\text{KV}}(\cdot)$ denotes the KV-cache memory demand induced by the calibrated upper-bound length estimate. We implement $\hat{l}_{upper}^{(r)}$ as a calibrated $(1-\epsilon)$-quantile bound via conformal calibration.

H-MAS solves the above objective via the Meta Controller updates $\mathcal{A}_t$ every ΔT_{meta} using V_{long}, while the Statera Agent updates $\mathcal{C}_t$ at a finer granularity using V_{short}.

3.2 Perception & Prediction Layer: Request Tagging and Cost Estimation

H-MAS avoids treating requests as black boxes by using a perception–prediction pipeline that converts raw prompts and metadata into structured request tags and lightweight cost proxies for scheduling.

Request tagging. The Perception Agent (Φ_{perc}), implemented with a fine-tuned small-scale LLM, extracts task type τ_r and per-request SLO constraints ($SLO_{ttft}^{(r)}, SLO_{tpot}^{(r)}$). It also predicts the output length $l_{\text{pred}}^{(r)}$ to support cost and memory planning. These outputs form a feature vector $\mathbf{x}_r$ attached to the request.

Cost estimation. The Prediction Agent (Φ_{pred}) maps $\mathbf{x}_r$ to latency-oriented cost proxies ($\hat{t}_{ttft}^{(r)}, \hat{t}_{tpot}^{(r)}$) and derives the total service duration $T_{\text{tot}}^{(r)} = \hat{t}_{ttft}^{(r)} + \hat{t}_{tpot}^{(r)} \cdot l_{\text{pred}}^{(r)}$. The tagged request and its cost proxies are then consumed by the execution layer for ordering and admission decisions.

Because scheduling quality depends on the reliability of semantic tags and cost proxies, H-MAS uses a simple deterministic backoff path when the perception output is missing, malformed, or incomplete. If the Perception Agent fails to return a valid structured output (e.g., malformed JSON, missing task type, or an invalid SLO field), we mark the request as *unknown* rather than forcing an uncertain semantic assignment. For such requests, H-MAS does not apply aggressive semantic-aware specialization; instead, it falls back to a historical safety prior derived from recent workload statistics, such as the most frequent task bucket and its associated coarse cost profile. This preserves predictability and avoids harmful misrouting caused by low-confidence semantics.

3.3 Feedback Layer: QoS Agent and Multi-Horizon State Construction

H-MAS closes the loop with a QoS Agent (Φ_{qos}) that aggregates runtime metrics into multi-horizon

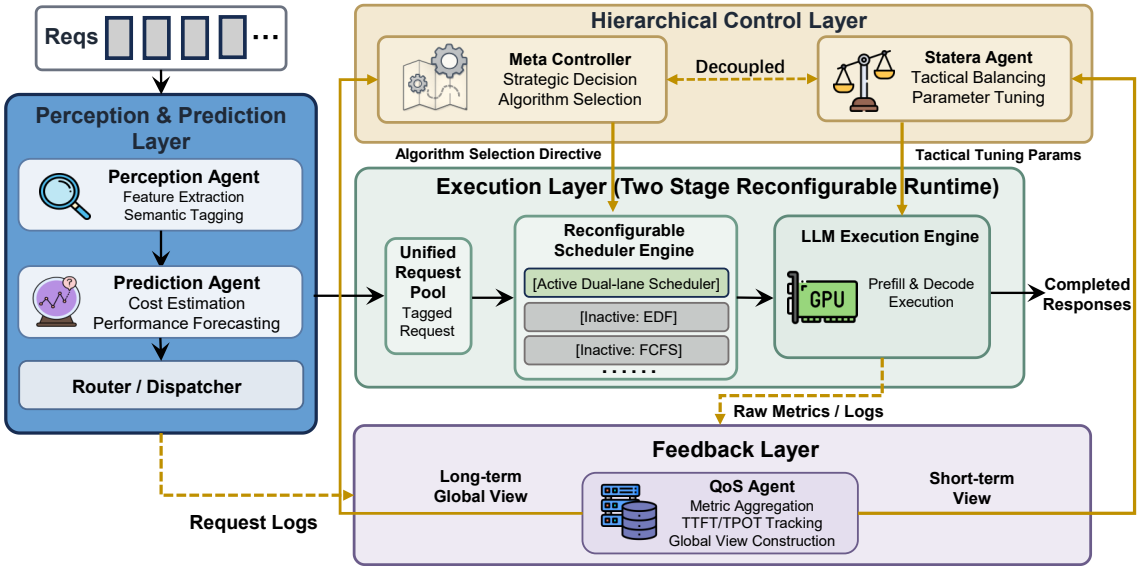


Figure 3: Overview of H-MAS with four layers: perception and prediction, feedback, hierarchical control, and execution. The controller outputs the active scheduling structure and execution parameters used during inference.

system states for hierarchical control.

Short-horizon state. Φ_{qos} tracks fast-changing signals (e.g., queue length, utilization, recent TTFT/TPOT) to form a short-horizon view V_{short} for fast execution regulation.

Long-horizon state. It also aggregates windowed workload statistics (e.g., arrival patterns, request-length distribution, tenant/task composition) and smoothed QoS outcomes to construct a long-horizon view V_{long} for slow structural adaptation.

3.4 Hierarchical Control Layer

This subsection describes the control layer: the Meta Controller updates high-level scheduling rules from V_{long}, while the Statera Agent regulates execution from V_{short}.

3.4.1 Meta Controller: Structural Adaptation

The Meta Controller is a slow loop that updates the scheduling structure at coarse time scales to track persistent workload shifts. It uses an LLM-based decision function $\Pi_{meta}(\cdot)$ to interpret the long-horizon global view V_{long} and select an active structure $\mathcal{A}_t \in \mathbf{\Omega}$. To avoid reacting to short-term noise, the decision is updated every ΔT_{meta} seconds:

$$\mathcal{A}_t = \Pi_{meta}(V_{\text{long}}), \quad \Pi_{meta} : \mathcal{V} \to \mathbf{\Omega}, \qquad (3)$$

where $\mathcal{V}$ is the feature space of aggregated workload and QoS signals.

To prevent inaccurate semantic inference from causing unstable structural decisions, the Meta Controller follows a deterministic degraded-mode path. If the controller output is invalid, unparsable, or inconsistent with the required JSON schema, the decision is rejected and the affected requests are treated as *unknown*. In this case, H-MAS bypasses semantic-sensitive specialization and instead selects a conservative policy using historical workload statistics already maintained in V_{long}, rather than relying on uncertain semantic tags. If guard conditions indicate severe instability or overload, the controller further falls back to the safe baseline through the emergency bypass path.

The chosen $\mathcal{A}_t$ reconfigures the queue structure to match the dominant failure pattern in V_{long}. It enables isolation (e.g., Two-lane) when long requests induce sustained HOL blocking (high length variance and TTFT violations co-moving with long-request backlog), switches to deadline ordering (e.g., EDF by D_r) when violations are driven by deadline misses, uses fairness-oriented sharing (e.g., WFQ/Stride) under persistent unfairness, and falls back to FCFS under anomalies ($\xi_{bypass} = 1$) for availability. Additional implementation details are provided in Appendix C.

3.4.2 Statera Agent: Stability Regulation

The Statera Agent governs the execution control space $\mathcal{D}_{ctrl}$ to mitigate OOM risks and suppress system jitter.

Probabilistic Safety via Conformal Calibration.

To maximize memory utilization without exhausting M_{cap}, the agent implements a bucketed

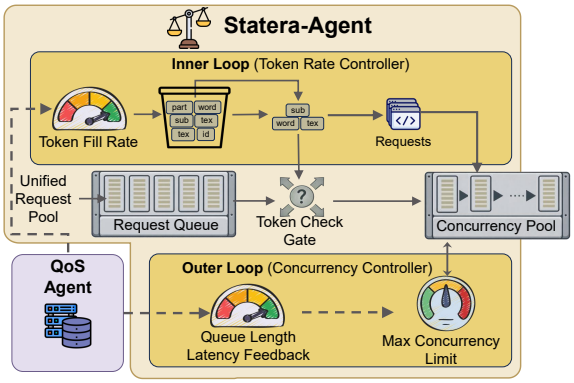


Figure 4: Architecture of the Statera Agent with two-layer regulation for execution stability.

conformal calibration mechanism Ψ_{safe}. By tracking real-time deviation e_r across prompt-length buckets, it updates the error distribution D_b to calculate a $(1-\epsilon)$ quantile upper bound:

$$\hat{l}^{(r)}_{upper} = l^{(r)}_{pred} + q_{1-\epsilon}(D_b). \quad (4)$$

This bound serves as a deterministic admission fence, letting $Q_{run}(t)$ denote the admitted in-flight requests, it ensures $\sum_{r \in Q_{\text{run}}(t)} \hat{l}^{(r)}_{upper}$ stays within the memory budget while accounting for model-specific length uncertainty.

Stability Control via Two-Layer Actuation. As shown in Fig. 4, the agent employs a two-layer feedback controller to maintain utilization within the target band $[u_{low}, u_{high}]$ (derived via extremum seeking from tg_{best}).

- **Inner Loop (P-Controller):** Regulates the token budget $\mathcal{B}_{token}$ at each scheduling cycle to throttle generation rates and damp instantaneous execution jitter; concretely, $\mathcal{B}_{token}(t)$ caps the number of decode tokens issued per scheduling cycle across all running sequences.
- **Outer Loop (I-Controller):** Adjusts the concurrency baseline N_{run} when fast-loop regulation saturates, correcting persistent deviations in SLO violation rates and utilization.

Together, the two layers adjust $\mathcal{B}_{token}$ and N_{run} to keep utilization within $[u_{low}, u_{high}]$.

3.5 Execution Layer: Polymorphic Scheduling Engine

The execution layer decouples logical request ordering from physical resource enforcement through a two-tier architecture.

External Polymorphic Scheduler: H-MAS maintains an extensible algorithm pool $\mathbf{\Omega}$ (e.g., EDF, WFQ, MLFQ). The Meta Controller selects the active queue structure $\mathcal{A}_t \in \mathbf{\Omega}$, and the execution engine instantiates $\mathcal{A}_t$ to enforce request ordering. For mixed workloads, the engine employs a Two-lane mechanism to achieve predictive isolation. This trade-off is most visible when workload heterogeneity is limited or when the per-lane concurrency becomes too small to sustain efficient decoding. It partitions requests into latency-sensitive Fast Lanes and throughput-oriented Slow Lanes via the proxy $T^{(r)}_{tot}$. By partitioning requests into separate lanes, the scheduler may reduce the effective decode batch size within each lane, which can slightly lower batching efficiency and worsen TPOT in some regimes. Within each lane, requests are prioritized by an urgency score:

$$S_r = D_r - V_{sys} - T^{(r)}_{tot}, \quad (5)$$

where S_r represents the remaining slack, and a lower score triggers higher scheduling priority, V_{sys} is the current system delay estimate. We derive $D_r = a_r + SLO^{(r)}_{ttft} + SLO^{(r)}_{tpot} \cdot l^{(r)}_{\text{pred}}$, a_r denotes the arrival time of request r.

Internal Regulation Interface: Acting as the actuation boundary for stability control, this interface enforces safety constraints at each scheduling cycle.

- **Admission Control:** It admits requests $Q_{run}(t)$ only if the calibrated memory demand satisfies Eq. 2, bounding OOM risk using the calibrated upper bound and a conservative fragmentation margin δ_{frag}.
- **Flow Regulation:** The interface modulates token issuance $\mathcal{B}_{token}$ to damp short-term execution jitter and caps parallel sequences N_{run} to stabilize utilization.

These mechanisms ensure that high-level ordering decisions are executed within explicit safety and hardware efficiency boundaries.

4 Evaluation

Setup. We evaluate H-MAS across multiple LLM scales and production-style workloads. Experiments cover real-world arrival patterns, semantic shifts, and mixed latency–throughput objectives using public traces from Azure (Qiu et al., 2025), ShareGPT (Team, 2024), and Coding (Hendrycks et al., 2021). Experiments are conducted on a cluster of 8 NVIDIA A100 (80GB) GPUs with NVLink. We evaluate Qwen3-8B/14B/30B (Yang et al., 2025) and Llama-3.3-70B (Grattafiori et al., 2024), employing Tensor Parallelism (TP) (Shoeybi et al., 2020) for models exceeding 8B parameters.

Workloads. To emulate production dynamics,

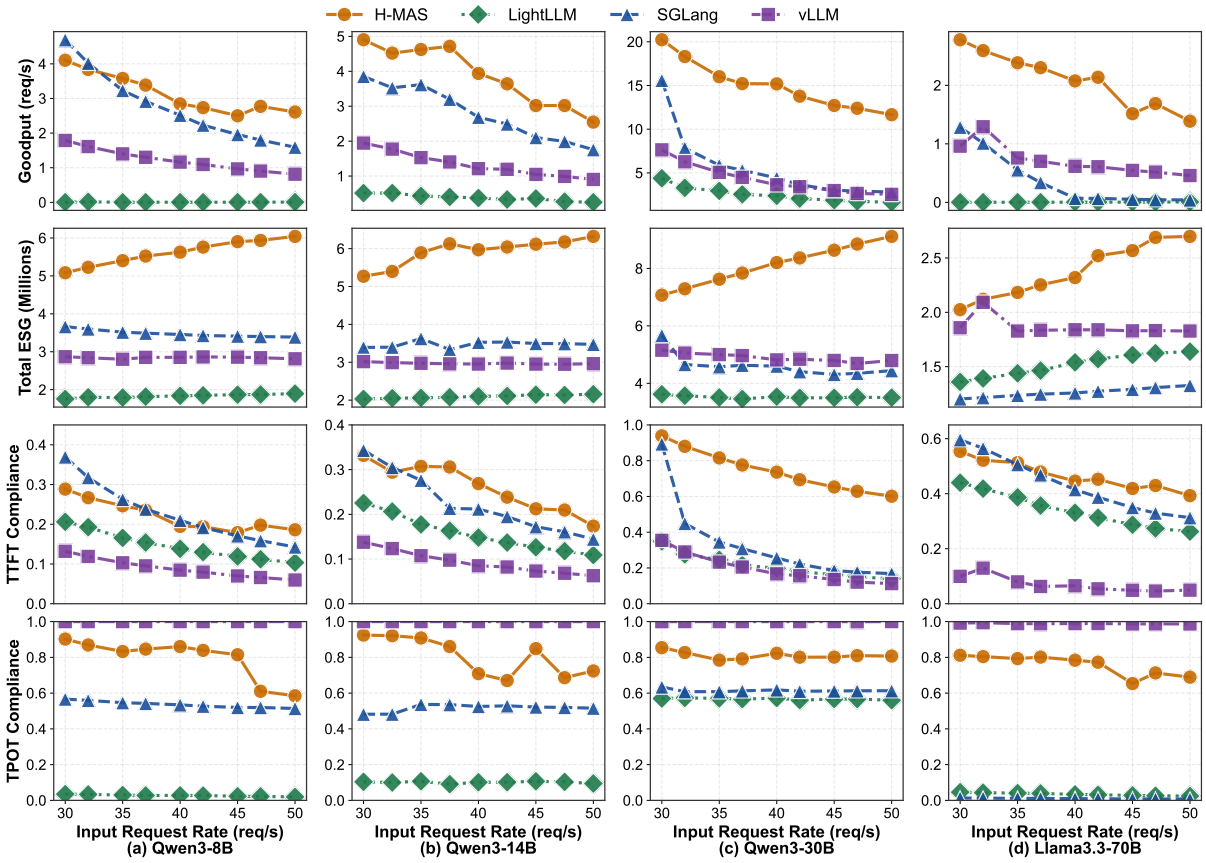


Figure 5: End-to-end scaling results under increasing input rate (RPS). Rows report (top to bottom) SLO-goodput, ESG, TTFT compliance, and TPOT compliance. H-MAS delivers higher SLO-goodput and ESG with more stable TTFT/TPOT compliance as load increases.

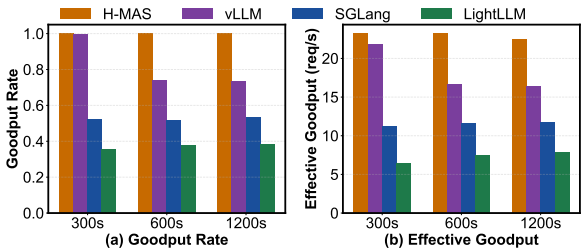
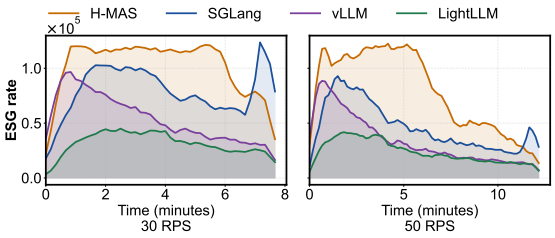
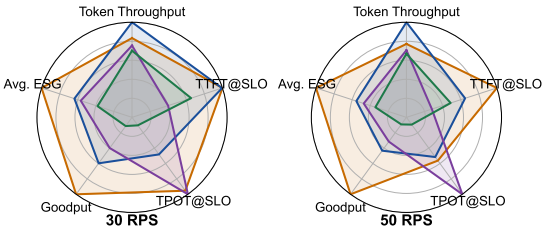


Figure 6: Trace-driven evaluation on the same workload trace with varying run durations (300 s / 600 s / 1200 s).



(a) ESG rate over time under different input rates.



(b) Multi-metric trade-offs at different loads.

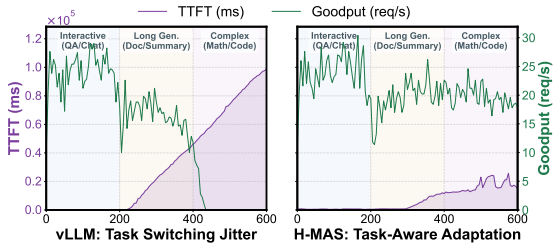
Figure 7: Multi-metric performance comparison under different input rates.

we construct mixed workloads by sampling requests from the Coding and ShareGPT datasets. Each request receives an individualized SLO based on semantic type and target output length (Appendix E.5).

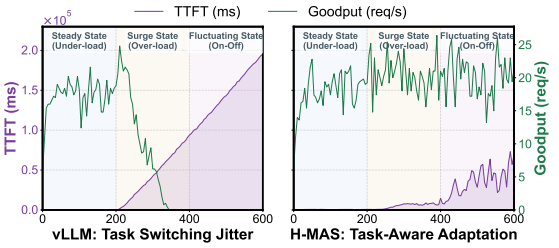
Evaluation Dimensions. We evaluate system behavior along two orthogonal dimensions:

- *Traffic Dynamics:* (i) *Dynamic Replay* that replays the Azure trace arrivals to preserve burstiness; (ii) *Fixed Scaling* that sweeps the offered load; (iii) *Scale-Dominated Slices* that isolate fast-, medium-, and slow-timescale workload dynamics under fixed scheduling; (iv) a 600s *Time-Varying Stress Test* with steady/surge/fluctuating phases to induce structural drift.
- *Semantic Adaptation:* a separate 600s *semantic shift* workload that sequentially switches among three request regimes—latency-sensitive short chats, throughput-intensive long generations, and mixed/high-variance requests—to evaluate adaptation to changing objectives.

Metrics. We evaluate Throughput (req/s), Good-



(a) Task-mix shifts from interactive to long-generation and complex requests, showing the evolution of online TTFT and goodput across phases.



(b) Arrival-rate shifts across steady, surge, and fluctuating RPS regimes, illustrating latency and throughput dynamics over time.

Figure 8: Time-series behavior under staged workload drift.

put (throughput of requests meeting SLOs), and SLO attainment. To capture utility beyond binary compliance, we adopt Expected Service Gain (ESG) (Zhang et al., 2025), calculated using the end-to-end completion latency TTLT = TTFT + $L_o \cdot$ TPOT:

$$ESG = \sum_{r \in \mathcal{R}} \mathcal{V}_{base}^{(r)} \cdot \min\left(1, \left[\frac{SLO_{ttlt}^{(r)}}{TTLT_r}\right]^{\alpha}\right), \quad (6)$$

where $\mathcal{V}_{base}^{(r)} = w_i L_i^{(r)} + w_o L_o^{(r)}$ denotes the token-weighted base value. In addition, for the scale-dominated analysis in Sec. 4.2, we report a utilization-gap mismatch score that measures the degree of runtime oversubscription under fixed scheduling.

Baselines. We compare H-MAS against three state-of-the-art and actively maintained serving systems that represent the current industry standards: (1) **vLLM** (Kwon et al., 2023), the industry-standard engine using PagedAttention and FCFS scheduling; (2) **SGLang** (Zheng et al., 2024), which optimizes shared prefixes via RadixAttention; and (3) **LightLLM** (Gong et al., 2025), which employs a high-performance asynchronous pipeline.

4.1 End-to-End Performance

We evaluate the end-to-end efficacy of H-MAS along two dimensions: (i) **Load Scalability** under increasing concurrency (30–50 RPS). (ii) **Long-term Stability** under sustained production-style execution.

SLO Compliance under Load. Fig. 5 shows that as load increases, baseline schedulers build up queues and Head-of-Line (HOL) blocking, which collapses TTFT compliance and degrades utility. H-MAS mitigates this by isolating latency-sensitive requests from long-context decoding, regulating utilization, and applying semantic-aware admission to prioritize requests likely to meet their SLOs. Two-lane isolation introduces a structural trade-off: it reduces cross-type interference and improves TTFT compliance under heterogeneous workloads, but may slightly worsen TPOT by shrinking the effective decode batch within each lane. In our experiments, the latency and goodput gains outweigh this batching penalty under heavy mixed workloads. At 50 RPS, H-MAS improves ESG by 1.82x over SGLang and 2.13x over vLLM.

Long-term Stability under Production Traces. We replay the same Azure trace for 300 s, 600 s, and 1200 s (Fig. 6). Here, Goodput Rate denotes the fraction of requests that satisfy both TTFT and TPOT SLOs, whereas Effective Goodput denotes the throughput of such requests. H-MAS remains stable across run durations, maintaining the highest Effective Goodput, while vLLM degrades as the run extends: its Goodput Rate drops to around 0.75, and its Effective Goodput decreases by roughly 15%–25%. SGLang and LightLLM remain consistently lower throughout.

Quantitative Insight and Trade-offs. Fig. 7a plots ESG over time on the Azure trace at 30 and 50 RPS. While the baselines degrade as the replay progresses, H-MAS sustains a higher ESG, especially under the 50 RPS peak load. Fig. 7b summarizes the trade-offs among throughput, goodput, TTFT/TPOT compliance, and mean ESG. H-MAS improves latency compliance without sacrificing throughput, yielding the highest mean ESG. ESG decomposition results are provided in Appendix D.1.

4.2 Adaptability to Multi-Scale Workload Dynamics

We conduct a 600s time-varying stress test with staged workload shifts. Fig. 8a illustrates the semantic migration across three phases (Interactive → Long Generation → Complex), while Fig. 8b reports the phase-wise trajectories under the corresponding RPS schedule. vLLM exhibits pro-

nounced task-switching jitter at phase transitions, as long and complex requests dominate, TTFT escalates sharply, goodput deteriorates, and recovery becomes difficult due to cross-type interference. In contrast, H-MAS sustains higher and steadier goodput across phase boundaries with a more gradual TTFT increase, keeping TTFT bounded even in complex phases. Overall, these results show that H-MAS adapts online to semantic shifts and preserves service utility under structural drift rather than overfitting to a single stationary regime.

Scale-dominated workload slices and mismatch analysis. To isolate how different temporal scales affect fixed scheduling, we construct three scale-dominated workload slices from a shared template, keeping the model, hardware, and SLO targets unchanged. The Fast slice injects short bursts without changing the long-run mean load. The Medium slice keeps arrivals relatively stable but introduces gradual drift in token demand and request composition. The Slow slice imposes persistent regime changes through lasting shifts in arrival intensity and task mix.

We quantify scheduling mismatch using a utilization-gap score:

$$M_\rho(t) = \max\left(0, \frac{\lambda_{\text{tok}}(t)}{\mu_{\text{tok}}(t) + \epsilon} - 1\right), \quad (7)$$

where $\lambda_{\text{tok}}(t)$ is the arriving token demand rate and $\mu_{\text{tok}}(t)$ is the runtime-measured token service rate over a sliding window Δ. This score captures runtime oversubscription when token demand exceeds effective service capacity, while naturally reflecting batching efficiency and prefill decode interference through $\mu_{\text{tok}}(t)$.

Table 1 reports the mismatch and execution performance under fixed scheduling. We observe distinct degradation patterns across time scales: the Fast slice mainly enlarges the tail mismatch, the Medium slice increases sustained mismatch through composition drift, and the Slow slice shifts the baseline mismatch upward under persistent regime changes. Across all three slices, H-MAS consistently reduces mismatch and improves TTFT compliance and normalized goodput, supporting the need to separate long-horizon policy adaptation from short-horizon execution control.

4.3 Ablation Study

We quantify the contributions of individual H-MAS components on the 600s trace (Table 2) to validate our design.

Table 1: Impact of scale-dominated workload slices under fixed scheduling. Mismatch is reported as mean / p95 utilization gap.

Scenario	Timescale	Mismatch (mean / p95)↓		TTFT Compliance↑		Goodput (norm.)↑	
		vLLM	H-MAS	vLLM	H-MAS	vLLM	H-MAS
Burst	Fast	1.560 / 3.588	1.203 / 2.767	19.24%	81.76%	0.2557	0.6831
Drift	Medium	1.835 / 2.110	1.415 / 1.627	28.96%	88.03%	0.2896	0.7199
Shift	Slow	2.202 / 5.606	1.798 / 4.094	11.25%	60.12%	0.1125	0.4357

Table 2: Ablation at 600s trace (Qwen3-30B). Metrics are normalized to H-MAS (1.00×).

Variant	ESG↑	TTFT_{95} ↓	TPOT_{95} ↓	Goodput↑
H-MAS	1.00×	1.00×	1.00×	1.00×
Disable Two-lane	0.88×	2.92×	1.08×	0.91×
Freeze Statera (static ($\hat{l}$, B_{token}, N_{run}))	0.23×	6.57×	0.92×	0.63×
Disable Meta (FCFS)	0.78×	3.16×	1.07×	0.87×
Rule-based Meta	0.89×	2.73×	1.04×	0.89×
Rule-based Perception & Prediction (rules+stats)	0.28×	3.95×	0.88×	0.74×

Perception & Prediction Quality. The Perception and Prediction Agents provide request tags and cost proxies for scheduling. Replacing them with rule/statistics-based tagging and cost estimation reduces ESG to **0.28**×, indicating that downstream scheduling is highly sensitive to the quality of request-level signals (see Appendix D.3 for the tagging and prediction accuracy).

Execution Stability. The *Statera Agent* is the stability anchor; freezing its dynamic regulation leads to the most catastrophic failure, with ESG dropping to **0.23**× and $TTFT_{95}$ spiking by **6.57**×. Similarly, disabling *Two-lane Isolation* increases $TTFT_{95}$ by **2.92**×, validating its role in mitigating HOL blocking during heavy workloads.

Meta-Coordination. Reverting the *Meta Controller* to FCFS reduces ESG to **0.78**×, while a lightweight rule-based meta controller improves it only to **0.89**×. This suggests that execution agents ensure baseline stability, but robust adaptation to non-stationary traffic still requires expressive long-horizon policy coordination.

5 Conclusion

This paper shows that QoS instability in MaaS LLM serving stems from a mismatch between multi-scale workload dynamics and fixed single-policy schedulers. We propose H-MAS, a hierarchical scheduler that decouples long-horizon adaptation from short-horizon execution control via runtime feedback. Across model sizes and workloads, H-MAS improves SLO attainment and QoS stability under workload variation. These results point to multi-scale hierarchical control as an effective design principle for robust LLM serving. Future work will extend H-MAS to multi-model serving and additional objectives such as cost and energy.

Limitations

Our approach has several limitations.

Reliance on prediction and observability. H-MAS depends on lightweight request-level attribute/cost prediction and timely runtime signals (e.g., queueing and latency feedback). When predictions are inaccurate or observability is limited, scheduling decisions may degrade.

Generality beyond our serving stack and traces. We evaluate on a specific set of engines/models and Azure-trace replays. Production MaaS can exhibit different caching behaviors, kernel-level bottlenecks, and agentic multi-step workloads; additional validation on broader models, longer horizons, and diverse traffic patterns is needed.

Overhead and deployment complexity. While latency masking can hide some overhead under contention, the benefit–overhead trade-off may differ at low load or in resource-constrained deployments.

Acknowledgments

We thank the anonymous reviewers and the area chair for their constructive feedback. This work was supported by the Shandong Provincial Natural Science Foundation under Grant ZR2023LZH012 and by the Laboratory Open Project of the Key Laboratory of Computing Power Network and Information Security, Ministry of Education, Qilu University of Technology (Shandong Academy of Sciences), under Grant 2023PY001.

References

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming Throughput-Latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 117–134, Santa Clara, CA. USENIX Association.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. 2024. Punica: Multi-tenant lora serving. In *Proceedings of Machine Learning and Systems*, volume 6, pages 1–13.

Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. 2022. Serving heterogeneous machine learning models on Multi-GPU servers with Spatio-Temporal sharing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 199–216, Carlsbad, CA. USENIX Association.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: fast and memory-efficient exact attention with io-awareness. NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Yichao Fu, Siqi Zhu, Runlong Su, Aurick Qiao, Ion Stoica, and Hao Zhang. 2024. Efficient llm scheduling by learning to rank. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

Ruihao Gong, Shihao Bai, Siyu Wu, Yunqian Fan, Zaijun Wang, Xiuhong Li, Hailong Yang, and Xianglong Liu. 2025. Past-future scheduler for llm serving under sla guarantees. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '25, page 798–813, New York, NY, USA. Association for Computing Machinery.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring coding challenge competence with APPS. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Ke Hong, Xiuhong Li, Lufang Chen, Qiuli Mao, Guohao Dai, Xuefei Ning, Shengen Yan, Yun Liang, and Yu Wang. 2025. SOLA: Optimizing SLO attainment for large language model serving with state-aware scheduling. In *Eighth Conference on Machine Learning and Systems*.

Saicharan Kolluru. 2025. Comparative analysis of large language model inference serving systems: A performance study of vllm and huggingface tgi. *Preprint*, arXiv:2511.17593.

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 *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 611–626, New York, NY, USA. Association for Computing Machinery.

Malgorzata Lazuka, Andreea Anghel, and Thomas Parnell. 2024. Llm-pilot: Characterize and optimize performance of your llm inference services. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '24. IEEE Press.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

Yuhan Liu, Cong Xu, Lu Liu, Yihua Wang, Feiyu Chen, Qi Jia, Yaqian Zhao, Zhichun Wang, and Xiang Li. 2025. DeMAC: Enhancing multi-agent coordination with dynamic DAG and manager-player feedback. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 14072–14098, Suzhou, China. Association for Computational Linguistics.

Tong Lu, Zhichun Wang, Yaoyu Zhou, Yiming Guan, Zhiyong Bai, and Junsheng Du. 2026. Scimkg: A multimodal knowledge graph for science education with text, image, video and audio. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 15466–15474.

Hyungjun Oh, Kihong Kim, Jaemin Kim, Sungkyun Kim, Junyeol Lee, Du-seong Chang, and Jiwon Seo. 2024. Exegpt: Constraint-aware resource scheduling for llm inference. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, page 369–384, New York, NY, USA. Association for Computing Machinery.

Sihyeong Park, Sungryeol Jeon, Chaelyn Lee, Seokhun Jeon, Byung-Soo Kim, and Jemin Lee. 2025. A survey on inference engines for large language models: Perspectives on optimization and efficiency. *Preprint*, arXiv:2505.01658.

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

Archit Patke, Dhemath Reddy, Saurabh Jha, Haoran Qiu, Christian Pinto, Chandra Narayanaswami, Zbigniew Kalbarczyk, and Ravishankar Iyer. 2024. Queue management for slo-oriented large language model serving. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*, SoCC '24, page 18–35, New York, NY, USA. Association for Computing Machinery.

Haoran Qiu, Anish Biswas, Zihan Zhao, Jayashree Mohan, Alind Khare, Esha Choukse, Íñigo Goiri, Zeyu Zhang, Haiying Shen, Chetan Bansal, Ramachandran Ramjee, and Rodrigo Fonseca. 2025. Modserve: Modality- and stage-aware resource disaggregation for scalable multimodal model serving. In *Proceedings of the 2025 ACM Symposium on Cloud Computing (SoCC 2025)*, New York, NY, USA. Association for Computing Machinery.

Stuart Russell and Peter Norvig. 2010. Artificial intelligence: a modern approach. 3rd. *Upper Saddle River, EUA: Prentice-Hall*.

MK Senehi, MK Senehi, Thomas R Kramer, John Michaloski, Richard Quintero, Steven R Ray, William G Rippey, and Sarah Wallace. 1994. *Reference Architecture for Machine Control Systems Integration: Interim Report*. US Department of Commerce, National Institute of Standards and Technology.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-lm: Training multi-billion parameter language models using model parallelism. *Preprint*, arXiv:1909.08053.

Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. 2025. Dynamollm: Designing llm inference clusters for performance and energy efficiency. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1348–1362.

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 *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*, OSDI'24, USA. USENIX Association.

Yinghao Tang, Tingfeng Lan, Xiuqi Huang, Hui Lu, and Wei Chen. 2025. Scorpio: Serving the right requests at the right time for heterogeneous slos in llm inference. *Preprint*, arXiv:2505.23022.

Vicuna Team. 2024. Sharegpt dataset.

Gerhard Weiss. 1999. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press.

Michael Wooldridge. 2009. *An Introduction to MultiAgent Systems*, 2nd edition. Wiley Publishing.

Bingyang Wu, Yinmin Zhong, Zili Zhang, Shengyu Liu, Fangyue Liu, Yuanhang Sun, Gang Huang, Xuanzhe Liu, and Xin Jin. 2024. Fast distributed inference serving for large language models. *Preprint*, arXiv:2305.05920.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, and 10 others. 2023. The rise and potential of large language model based agents: A survey. *Preprint*, arXiv:2309.07864.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41

others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, Carlsbad, CA. USENIX Association.

Wei Zhang, Zhiyu Wu, Yi Mu, Ning Rui, Banruo Liu, Nikhil Sarda, Myungjin Lee, and Fan Lai. 2025. Jitserve: Slo-aware llm serving with imprecise request information. *Preprint*, arXiv:2504.20068.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. Sglang: efficient execution of structured language model programs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

A Additional Related Work

Inference Runtime & Memory Efficiency. Significant strides have been made in optimizing the underlying execution engine to maximize hardware utilization. Orca (Yu et al., 2022) pioneered continuous batching (iteration-level scheduling), allowing new requests to join ongoing batches dynamically to minimize pipeline bubbles. Building on this, vLLM (Kwon et al., 2023) introduced PagedAttention to eliminate memory fragmentation, managing the KV-cache as non-contiguous pages. Further advancements like FlashAttention (Dao et al., 2022) and SGLang's RadixAttention (Zheng et al., 2024) have accelerated kernel-level performance and prefix sharing. Despite their execution efficiency, these systems mostly rely on rudimentary FCFS policies that treat requests homogeneously. H-MAS is orthogonal to these runtimes; it leverages them as the execution backend while introducing an intelligent control layer to manage the multi-scale dynamics and semantic diversity they overlook.

SLO-Aware Scheduling Heuristics. To address the "one-size-fits-all" limitation of FCFS, recent works have introduced specialized heuristics tailored for heterogeneous SLOs. **SOLA** (Hong et al., 2025) proposes a *state-aware* framework that dynamically adjusts iteration-level scheduling to balance the inherent trade-off between TTFT and TPOT. **JITServe** (Zhang et al., 2025) introduces the concept of "service gain," allocating *just enough* GPU bandwidth to meet deadlines while deferring non-urgent generation to save capacity. Similarly, **SCORPIO** (Tang et al., 2025) exploits SLO heterogeneity via a "TPOT Guard" mechanism, allowing requests with loose deadlines to skip decoding iterations to prioritize urgent tasks. However, these heuristics operate largely as *static policies* tailored to specific workload assumptions. They lack the situational awareness to handle "Medium Scale" structural drifts. Without a strategic governor, hard-coded rules (like iteration skipping) become rigid and sub-optimal when global traffic patterns shift unexpectedly.

Predictive & Learning-based Scheduling. Beyond reactive heuristics, modern engines increasingly integrate predictive signals to approximate optimal scheduling. **FastServe** (Wu et al., 2024) and **Ranking** (Fu et al., 2024) utilize output length predictors (via regression or ranking models) to implement preemptive scheduling (e.g., SJF), aiming to minimize job completion time. QLM (Patke et al., 2024) introduces a Request Waiting Time (RWT) estimator based on the Central Limit Theorem, dynamically reordering queues to prioritize requests at risk of SLO violation. LightLLM (Gong et al., 2025) leverage "Past-Future" profiling or quantization predictors to optimize memory planning and batch sizing proactively. Yet, these approaches function primarily as *single-layer micro-schedulers* that *tightly couple* prediction with execution. This leaves them vulnerable to prediction errors. In contrast, H-MAS employs a *Hierarchical Multi-Agent* architecture to explicitly decouple prediction from control. This separation allows the system to absorb forecast variance through tactical adjustments, ensuring robustness where tightly-coupled schedulers often fail.

B Notation & Definitions

B.1 Notation

Table 3 summarizes the symbols used throughout the paper. We group variables into **settings** (fixed hardware and SLO constraints), **states** (runtime observations at multiple timescales), and **strategies** (control outputs). Unless otherwise specified, r indexes a request and t indexes a scheduling step.

B.2 Metric Definitions

TTFT and TPOT. For each request r, let $t^{(r)}_{ttft}$ denote the measured time-to-first-token (from request arrival to first decoded token), and let $t^{(r)}_{tpot}$ denote the measured per-token decoding time (ms/token) during generation. We use $\hat{t}^{(r)}_{ttft}$ and $\hat{t}^{(r)}_{tpot}$ for their predicted proxies returned by Φ_{pred}.

SLO satisfaction and goodput. A request r is considered successful if it satisfies both SLO constraints:

$$\mathbb{C}_r \triangleq \left\{ t^{(r)}_{ttft} \leq SLO^{(r)}_{ttft} \ \wedge \ t^{(r)}_{tpot} \leq SLO^{(r)}_{tpot} \right\}. \tag{8}$$

We define Goodput Rate as the fraction of requests that satisfy both TTFT and TPOT SLOs:

$$\text{GoodputRate} \triangleq \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \mathbb{I}(\mathbb{C}_r). \tag{9}$$

Expected Service Gain (ESG). To capture utility beyond binary SLO compliance, we adopt the Expected Service Gain (ESG) metric following prior LLM serving work (Zhang et al., 2025). ESG models non-linear utility degradation as latency increases and weights requests by their service

importance. We use the same formulation with the following parameter settings: w_i=1 and w_o=2 for input and output token weights, reflecting the higher decoding cost, and α=1.0 as the latency penalty exponent. All evaluation metrics and ESG computations in this paper follow the definitions in (Zhang et al., 2025).

Utility and objective. $\mathrm{Utility}(r)$ denotes the per-request utility weight determined by the service class (e.g., tenant weight) inferred from τ_r. The control objective maximizes expected weighted goodput under safety and stability constraints.

B.3 Conformal Calibration for Memory Safety

H-MAS enforces a chance constraint on OOM by maintaining a bucketed error buffer D_b for length prediction errors. For request r in bucket b, we record $e_r = l_{real}^{(r)} - l_{pred}^{(r)}$ into D_b and compute the $(1-\epsilon)$ quantile $q_{1-\epsilon}(D_b)$. The admission upper bound is:

$$\hat{l}_{upper}^{(r)} = l_{pred}^{(r)} + q_{1-\epsilon}(D_b), \tag{10}$$

and admission must satisfy:

$$\sum_{r \in Q_{\mathrm{run}}(t)} m_{\mathrm{KV}}\left(\hat{l}_{\mathrm{upper}}^{(r)}\right) \le M_{\mathrm{cap}}\left(1-\delta_{\mathrm{frag}}\right). \tag{11}$$

B.4 Timescales and Update Intervals

H-MAS operates on multiple timescales. The QoS Agent updates V_{short} at each scheduling cycle and refreshes V_{long} over a longer aggregation interval to filter short-term noise. The Meta Controller updates the active topology $\mathcal{A}_t = \Pi_{meta}(V_{\mathrm{long}})$ every ΔT_{meta} to track slow workload drift. The Statera Agent applies fast/medium-timescale actuation by adjusting $(\hat{l}_{upper}, \mathcal{B}_{token}, N_{run})$ to enforce safe admission and suppress jitter.

C Meta Controller Details: Structural Adaptation and Policy Options

This appendix complements §3.4.1 by clarifying how the Meta Controller applies online structural changes in practice and what realizations of the decision function $\Pi_{meta}(\cdot)$ are supported. The Meta Controller is implemented as an LLM-based decision module instantiated with Qwen3-30B. To avoid interference with the serving GPUs, we deploy the Qwen3-30B controller on a separate server. The Meta Controller runs at a coarse time scale to track slow workload drift, issuing a macro structural decision $\mathcal{A}_t \in \mathbf{\Omega}$ while relying on bypass/rollback guards to avoid oscillation.

Table 3: Symbols and definitions used in H-MAS.

Category	Symbol	Description
Setting	M_{cap}	GPU memory capacity
	δ_{frag}	Memory fragmentation margin
	ϵ	OOM risk tolerance (confidence level)
	$\mathbf{\Omega}$	Pool of scheduling algorithms
State (System)	V_{long}	Long-term global system view
	V_{short}	Short-horizon system view
	$\bar{u}_t$	Window-averaged GPU utilization
	tg_{best}	Historical best throughput
	D_b	Bucketed prediction error distribution
	V_{sys}	System virtual time
	ξ_{bypass}	Emergency bypass indicator
State (Request r)	τ_r	Semantic task type
	$SLO_{ttft}^{(r)}$	TTFT constraint
	$SLO_{tpot}^{(r)}$	TPOT constraint
	$l_{pred}^{(r)}$	Predicted output length
	$\hat{t}_{ttft}^{(r)}$	Predicted time to first token
	$\hat{t}_{tpot}^{(r)}$	Predicted per-token generation time
	$T_{tot}^{(r)}$	Service-time proxy for scheduling
	D_r	Absolute deadline
Strategy	$\mathcal{A}_t$	Active scheduling algorithm
	$\hat{l}_{upper}^{(r)}$	Conformal length upper bound
	$\mathcal{B}_{token}$	Per-cycle token budget
	N_{run}	Maximum parallel sequences
Agent / Function	$\Phi_{perc}(\cdot)$	Semantic feature extractor
	$\Phi_{pred}(\cdot)$	Latency and cost predictor
	$\Phi_{qos}(\cdot)$	QoS aggregation function
	$\Pi_{meta}(\cdot)$	Meta Controller decision function

C.1 Interface Aligned with the Main Text Notation

Inputs: long-horizon state and guard signals. At each meta update (every ΔT_{meta} seconds), the Meta Controller reads the Long-horizon global view V_{long} produced by the QoS Agent $\Phi_{qos}(\cdot)$. In our implementation, V_{long} aggregates workload composition and performance trends, including (i) arrival intensity (e.g., RPS), (ii) request mix and length variance (proxying the severity of HOL blocking), and (iii) long-horizon TTFT/TPOT violation rates and tail statistics. For safety guards, the controller also consults a short-horizon diagnostic

signal from V_{short} (e.g., short-window TTFT violation rate or sudden tail spikes) to trigger ξ_{bypass} or rollback checks.

Outputs: macro structure and micro hints. The controller outputs: (i) the active scheduling topology $\mathcal{A}_t \in \mathbf{\Omega}$, which reconfigures the external queue ordering structure (e.g., EDF vs. Two-lane vs. FCFS), and (ii) optional *topology-specific hints* $\mathbf{h}_t$ that tune a small set of parameters without changing $\mathcal{A}_t$. We keep $\mathbf{h}_t$ low-dimensional (e.g., Two-lane gate period and weighting knobs) so that micro tuning remains stable and interpretable.

Online topology application (zero-downtime). When $\mathcal{A}_t$ changes, the execution layer applies the decision by rebuilding the scheduler's ordering state *in place*. Concretely, it clears the previous ordering structure and reconstructs the data structure corresponding to the new topology, while the execution engine continues to pump requests. EDF rebuilds a deadline-ordered heap keyed by D_r; SJF rebuilds a duration-ordered heap keyed by $T_{tot}^{(r)}$; Two-lane reinitializes lane gates and repopulates the Fast/Slow lane heaps using cognition outputs (e.g., $T_{tot}^{(r)}$, the inferred service class). This design supports online structural adaptation without restarting the serving pipeline.

C.2 Stability Guards and Choices for $\Pi_{meta}(\cdot)$

H-MAS uses a deterministic backoff rule when semantic outputs are unreliable. If the perception result is missing, malformed, or unparsable, the request is assigned to an *unknown* state. For *unknown* requests, the scheduler uses historical coarse statistics instead of semantic-aware routing. If the predicted length is inconsistent with runtime observations, the raw prediction is covered by a conservative rolling estimate used by admission control.

Update cadence and anti-thrashing. To avoid reacting to transient noise, $\Pi_{meta}(\cdot)$ is evaluated only once per ΔT_{meta}. In addition, we enforce two safety guards: emergency bypass ξ_{bypass} and rollback, which bound the downside of imperfect regime inference.

Emergency bypass. If the system enters extreme overload or severe instability, the controller activates $\xi_{bypass} = 1$ and reverts to a safe baseline: $\xi_{bypass} = 1 \Rightarrow \mathcal{A}_t \equiv \text{FCFS}, \quad \mathcal{D}_{ctrl} \equiv \text{Unbounded}$. This prioritizes availability and prevents cascading failure; normal adaptation resumes once guard conditions are cleared.

Rollback guard window. After a topology switch, the controller monitors a short guard window using V_{short} (e.g., TTFT tail and violation rate). Let $(\mathrm{TTFT}_{99}^{pre}, \nu^{pre})$ be the baseline measured immediately before switching, and $(\mathrm{TTFT}_{99}, \nu)$ be the current measurements within the guard window. Rollback triggers if

$$\mathrm{TTFT}_{99} > \rho \cdot \mathrm{TTFT}_{99}^{pre} \;\text{ or }\; \nu > \rho \cdot \nu^{pre}, \quad \rho > 1, \tag{12}$$

and the controller restores the last stable topology. This mechanism suppresses oscillation and limits the cost of transient misclassification.

Polymorphic policy pool $\mathbf{\Omega}$ (what can be switched). In our implementation, $\mathbf{\Omega}$ corresponds to the set of queue topologies supported by the external scheduler. To make the design interpretable, we group policies by the *dominant failure mode* they target: (i) deadline-driven policies (EDF, LLF) that prioritize by D_r or laxity to reduce deadline misses; (ii) duration-driven policies (SJF, SRTF) that prioritize by predicted service time $T_{tot}^{(r)}$ (or remaining time) to mitigate HOL blocking under high length variance; (iii) isolation-based policy (Two-lane) that partitions requests using $T_{tot}^{(r)}$ into Fast/Slow lanes for latency–throughput decoupling; (iv) fairness-oriented policies (WFQ/Stride/Lottery/CFS/RR/MLFQ/Priority) that enforce tenant/class fairness or bounded sharing under persistent unfairness; (v) the availability baseline (FCFS) used for stable fallback and emergency bypass. Unless stated otherwise, we mainly evaluate a representative subset of $\mathbf{\Omega}$ (FCFS, EDF, SJF, Two-lane, WFQ), while the remaining policies demonstrate extensibility of the polymorphic engine.

Choices for implementing $\Pi_{meta}(\cdot)$. The decision function $\Pi_{meta}(\cdot) : \mathcal{V} \to \mathbf{\Omega}$ can be instantiated in multiple ways under the same interface. We support the following options:

- **LLM-based selector (ours).** A LLM interprets V_{long} and emits a strict JSON decision containing $\mathcal{A}_t$ plus optional hints $\mathbf{h}_t$ and bypass signal ξ_{bypass}. Outputs are validated against a whitelist $\mathbf{\Omega}$; parsing failures fall back to local rules.
- **Rule-based policy map.** A deterministic mapping from aggregated failure patterns in V_{long} to $\mathcal{A}_t$ (e.g., deadline-dominated misses $\to$ EDF; sustained HOL symptoms $\to$ Two-lane; persistent unfairness $\to$ WFQ/Stride; anomalies/overload

$\rightarrow$ FCFS).

- **Supervised classifier.** A lightweight model trained offline to predict the best $\mathcal{A}_t$ from features of V_{long} (arrival intensity, length mix, violation trends, utilization statistics), enabling fast inference and reproducibility.

- **Contextual bandit.** Treat policy selection as online bandit learning with reward defined by ESG or weighted goodput; rollback/bypass act as hard safety guards during exploration.

- **Change-point + lookup table.** Detect regime changes in V_{long} (e.g., distribution shift in length mix) and switch using a precomputed policy table; this is simple and robust for production deployment.

D Additional Results

D.1 ESG Decomposition: Additional Analysis

Fig. 9 decomposes Expected Service Gain (ESG) into *TTFT*, *TPOT*, and *Throughput* components under two complementary summaries. The *Peak Window* aggregates ESG over the most congested interval of the trace, reflecting robustness under bursty contention and worst-case queueing pressure. The *Run-wise Average* reports the mean ESG over the full run, capturing sustained service efficiency under typical operating conditions.

The decomposition indicates that some throughput-oriented systems can appear strong in the *Peak Window* yet fail to sustain their utility over the entire run. For instance, vLLM achieves a relatively high peak-window ESG, but its total ESG drops substantially under the run-wise average, suggesting that peak utility is transient and sensitive to regime changes and evolving queue dynamics. In contrast, H-MAS remains high in both views, implying more stable value delivery throughout execution rather than benefiting only from short periods of favorable conditions.

Component-wise, vLLM's ESG is largely driven by *TPOT* and *Throughput*, with a limited *TTFT* contribution. By contrast, H-MAS preserves substantial *TTFT* and *TPOT* components in addition to throughput, supporting the claim that its advantage comes from consistently satisfying latency-sensitive objectives (TTFT/TPOT SLO attainment) instead of relying on raw throughput alone.

D.2 Multi-scale Robustness under Traffic and Semantic Drift

We evaluate robustness under two non-stationarity axes: traffic drift (arrival-rate changes) and semantic drift (task-mix changes). All drift tests use a 600 s horizon.

For traffic drift, we consider (i) dynamic replay that preserves the burstiness of the Azure trace by replaying its arrival timestamps, (ii) fixed scaling that sweeps the offered load around a representative operating point (we use the Azure trace mean RPS), and (iii) a time-varying stress test that induces explicit structural drift by changing the arrival rate over three phases.

Time-varying stress test. We set the base rate to $\lambda_0 = 40$ RPS and define the stress-test arrival rate as:

$$\lambda(t) = \begin{cases} 0.6\lambda_0, & t \in [0, 200), \\ 1.4\lambda_0, & t \in [200, 400), \\ 1.8\lambda_0, & t \in [400, 600),\ (t \bmod 20) < 10, \\ 0.2\lambda_0, & t \in [400, 600),\ (t \bmod 20) \geq 10. \end{cases} \tag{13}$$

and generate arrivals with an inhomogeneous Poisson process, i.e., within each interval $\Delta t \sim \text{Exp}(\lambda(t))$. This yields a steady phase, a surge phase, and an on-off fluctuating phase.

For semantic drift, we fix the traffic process using dynamic replay of the Azure trace to preserve burstiness, and only vary the task composition over time. We then construct three regimes and switch them sequentially every 200 s: a latency-sensitive regime dominated by short interactive tasks, a throughput-intensive regime dominated by long generations, and a mixed high-variance regime. In implementation, `sample_json_phased` parses `task_type` and SLO fields from the dataset records, filters invalid prompt/output pairs, and routes requests into three buckets according to predefined task-type sets.

Table 4 summarizes multi-scale robustness under (A) RPS drift and (B) semantic task drift. $\Delta\text{GP}_{1\to3}$ measures the goodput drop from Phase I to Phase III (smaller is better), while $\mathcal{R}_{3/2}$ reports the recovery ratio from Phase II to Phase III (larger is better). CV_{TTFT} and CV_{ESG} quantify run-level jitter via coefficient of variation (σ/μ), and $\text{ESG}_{\text{P/A}}$ (peak-to-average ratio) captures whether service utility is sustained or dominated by transient peaks. Across both drift types, H-MAS exhibits substantially smaller goodput degradation and stronger recovery, while also achieving lower variability and

Table 4: Multi-scale robustness under (A) RPS Drift and (B) Semantic Task Drift. ↓: lower is better; ↑: higher is better. CV denotes Coefficient of Variation (σ/μ).

System	$\Delta GP_{1\to3}(\%)\downarrow$	$\mathcal{R}_{3/2}\uparrow$	$CV_{TTFT}\downarrow$	$ESG_{P/A}\downarrow$	$CV_{ESG}\downarrow$	Avg ESG↑
Panel A: Robustness under RPS Drift (arrival-rate fluctuations)						
vLLM	100.0	0.00	1.849	4.67	1.304	819.43
H-MAS	**29.4**	**0.89**	**1.230**	**3.73**	**0.913**	**1158.47**
Panel B: Robustness under Task Drift (semantic task-mix shifts)						
vLLM	94.0	0.07	2.033	3.05	1.304	878.61
H-MAS	**7.1**	**0.93**	**1.468**	**2.26**	**0.913**	**1261.15**

$\Delta GP_{1\to3}$: Goodput drop from Phase I to III.
$\mathcal{R}_{3/2}$: Recovery Ratio (Phase III/II).
CV: coefficient of variation (σ/μ) over the full run; smaller indicates lower jitter.
Avg ESG Rate: run-wise average of ESG rate computed over all 10 s windows during the evaluation period.
ESG Peak/Avg: ratio between the maximum ESG rate observed in any 10 s window and the run-wise average ESG rate.

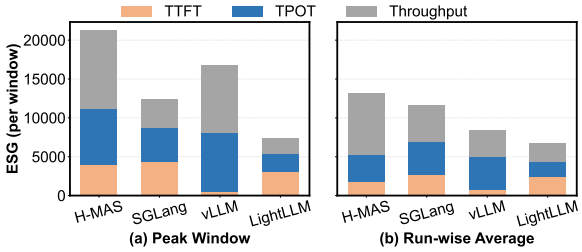


Figure 9: ESG decomposition across scheduling systems. (a) Peak-window ESG aggregated over the most congested time window of the trace, reflecting system robustness under bursty and worst-case contention. (b) Run-wise average ESG computed over the entire trace, capturing long-term service efficiency under typical workload conditions.

Table 5: **Accuracy of Cognitive Agents.** Evaluation on the LLM-Pilot validation set. The Perception Agent is measured by classification accuracy or error margins (MAE/MAPE), while the Prediction Agent is evaluated by R^2 regression scores.

Component	Target Field / Metric	Metric Type	Score
Perception	`Task type`	Accuracy	**94.2%**
	`Output tokens`	MAPE	12.4%
	`ttft` (Target)	MAE	18.5 ms
	`tpot` (Target)	MAE	3.2 ms
Prediction	Avg. Token Decode Time	R^2 Score	0.9576
	Token Throughput	R^2 Score	0.9418
	Mean Latency/Token	R^2 Score	0.9319
	Prefill Time	R^2 Score	0.7236

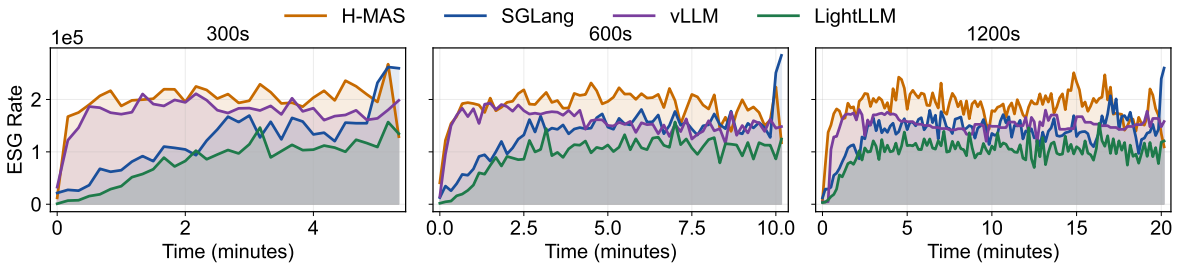
peakiness (smaller CV and $ESG_{P/A}$) and higher average ESG. These results indicate that the hierarchical design—coarse-grained structural adaptation via $\mathcal{A}_t$ combined with fast execution regulation via $\mathcal{C}_t$—delivers more stable utility under both arrival-rate fluctuations and semantic objective shifts.

D.3 Accuracy of Perception and Prediction Agents

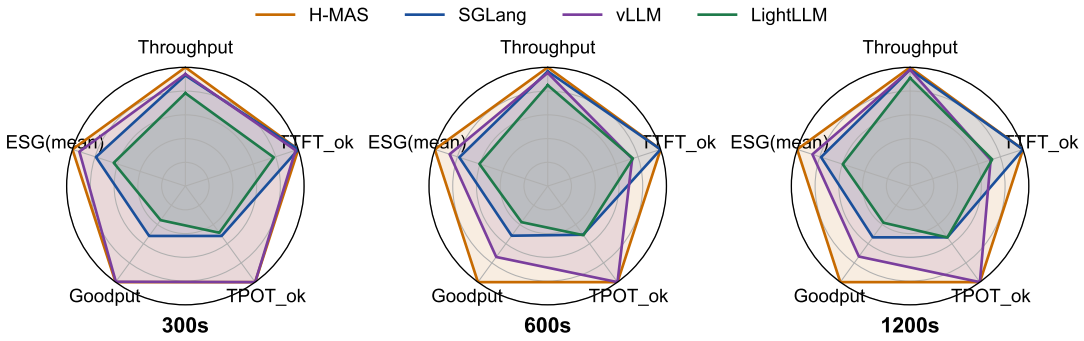
H-MAS relies on the Perception & Prediction layer to convert raw prompts and metadata into request tags and cost proxies used by downstream scheduling. Table 5 reports the accuracy of the Perception and Prediction agents on the ShareGPT and LLM-Pilot validation set (Lazuka et al., 2024). The Perception Agent is fine-tuned from Qwen3-1.7B on our constructed dataset (Appendix E.5). It Agent achieves high accuracy on semantic routing fields such as `Task type` (**94.2%**) and maintains bounded numeric error when extracting or estimating request attributes (e.g., `Output tokens` with 12.4% MAPE, and small MAE on latency-related targets such as `ttft` and `tpot`). Building on the perception output, the Prediction Agent estimates execution costs using a CatBoost regressor trained on the LLM-Pilot dataset (Lazuka et al., 2024). It attains strong regression fidelity for the key quantities that drive ordering and control, including Avg. Token Decode Time (R^2=0.9576), Token Throughput (R^2=0.9418), and Mean Latency/Token (R^2=0.9319). While Prefill Time is inherently noisier and thus exhibits a lower R^2 (0.7236), this level of accuracy is primarily used for coarse-grained cost stratification and lane/topology decisions, where relative ranking and bucket-level separation matter more than exact point estimates.

D.4 Window-length sensitivity of ESG-rate dynamics

To complement the main results, we further evaluate the *ESG-rate* dynamics under different evaluation windows (300s/600s/1200s) on the Azure trace. Fig. 10 reports (a) the ESG-rate trajectories



(a) ESG rate over time for 300s/600s/1200s windows across H-MAS, SGLang, vLLM, and LightLLM (curves truncated to the earliest completion time per panel).



(b) Radar summaries of normalized multi-metric performance, including Throughput, TTFT_{ok}, TPOT_{ok}, Goodput, and ESG(mean), under 300s/600s/1200s windows.

Figure 10: ESG-rate dynamics and multi-metric trade-offs of four serving systems on the Qwen3-30B model under different window settings.

over time and (b) radar summaries of normalized multi-metric performance, including Throughput, TTFT_{ok}, TPOT_{ok}, Goodput, and ESG(mean). All systems exhibit temporal fluctuations in ESG rate; however, H-MAS consistently operates around a higher ESG-rate level . The radar views further indicate that H-MAS sustains the most balanced operating point: its gains persist across both latency compliance axes (TTFT/TPOT) and throughput-facing axes (Throughput/Goodput), rather than being driven by a single metric. Overall, varying the evaluation window does not change the qualitative conclusion that H-MAS delivers more stable and well-balanced utility over sustained execution.

E Implementation and Evaluation Details

E.1 End-to-end Control Loop of H-MAS

Algorithm 1 summarizes the end-to-end closed-loop execution of H-MAS. At a high level, the system overlaps cognition with inevitable queueing time: upon request arrival, the Perception and Prediction agents produce request-level features $\mathbf{x}_r$ and cost proxies (e.g., $l_{pred}^{(r)}$, $\hat{t}_{ttft}^{(r)}$, $\hat{t}_{tpot}^{(r)}$, and $T_{tot}^{(r)}$), which are attached to the request and stored in a unified request pool. During execution, the QoS agent continuously aggregates runtime logs into a short-horizon view V_{short} and a long-horizon view V_{long}, and updates the bucketed calibration buffer D_b from observed completion errors. On a slow timescale, the Meta Controller applies $\Pi_{meta}(V_{\text{long}})$ to select the active topology $\mathcal{A}_t \in \mathbf{\Omega}$ (with an emergency bypass ξ_{bypass}). On a fast/medium timescale, the Statera Agent computes risk-bounded admission fences $\hat{l}_{upper}^{(r)}$ and applies two-layer actuation to output execution controls $\mathcal{C}_t = (\hat{l}_{upper}, \mathcal{B}_{token}, N_{run})$. The execution layer then instantiates the selected topology and enforces $\mathcal{C}_t$ in each pump cycle to admit and dispatch a batch safely while stabilizing utilization.

E.2 Statera Agent: Safety Fence and Dual-loop Regulation

Algorithm 2 details the Statera Agent, which governs the execution control space $\mathcal{D}_{ctrl}$ by outputting $\mathcal{C}_t = (\hat{l}_{upper}, \mathcal{B}_{token}, N_{run})$. Its role is twofold: (i) enforce risk-bounded admission under output-length uncertainty, and (ii) suppress short-horizon jitter while keeping GPU utilization within an efficient operating regime.

The safety component implements online bucketed conformal calibration. Upon request completion, the agent observes the length prediction

Algorithm 1: H-MAS main control loop (end-to-end closed-loop scheduling).

Input : Request stream $\mathcal{R}$; algorithm pool $\mathbf{\Omega}$; memory settings $(M_{cap}, \delta_{frag}, \epsilon)$; control bounds $(B_{\min}, B_{\max}, N_{\min}, N_{\max})$; update periods $(\Delta T_{meta}, \Delta T_{long})$.

Output : Per-cycle scheduling topology $\mathcal{A}_t$ and control actions $\mathcal{C}_t = (\hat{l}_{upper}, \mathcal{B}_{token}, N_{run})$.

Initialize: $\mathcal{A}_0 \leftarrow$ SJF; $D_b \leftarrow \emptyset$; $V_{\text{short}}, V_{\text{long}} \leftarrow \emptyset$; $\mathcal{B}_{token} \leftarrow B_{\max}$; $N_{run} \leftarrow N_{\max}$.

foreach *pump cycle* $t = 1, 2, \ldots$ **do**

  `// (1) Cognition: Perception & Prediction (overlapped with queueing when possible)`

  foreach *new request* r *arriving since last cycle* **do**

    $\mathbf{x}_r \leftarrow$ `Perceive`(r) `// ` $\mathbf{x}_r = [\tau_r, SLO^{(r)}_{ttft}, SLO^{(r)}_{tpot}, p_r, l^{(r)}_{pred}]$

    $(\hat{t}^{(r)}_{ttft}, \hat{t}^{(r)}_{tpot}) \leftarrow$ `Predict`$(\mathbf{x}_r)$

    $T^{(r)}_{tot} \leftarrow \hat{t}^{(r)}_{ttft} + \hat{t}^{(r)}_{tpot} \cdot l^{(r)}_{pred}$

    push r with features $(\mathbf{x}_r, T^{(r)}_{tot})$ into unified request pool

  `// (2) Feedback: aggregate runtime logs to multi-horizon views`

  $(V_{\text{short}}, V_{\text{long}}, D_b, \bar{u}_t, \nu_t) \leftarrow$ `Agg`(logs, D_b)

  `// (3) Emergency bypass (availability first)`

  if `Bypass` (V_{short}, V_{long}) **then**

    $\xi_{bypass} \leftarrow 1$; $\mathcal{A}_t \leftarrow$ FCFS; $\mathcal{C}_t \leftarrow$ Unbounded

    `BuildTopology` $(\mathcal{A}_t)$; `Dispatch` $(\mathcal{A}_t, \mathcal{C}_t)$

    continue

  $\xi_{bypass} \leftarrow 0$

  `// (4) Macro control: Meta Controller updates topology at coarse timescale`

  if *time since last update* $\geq \Delta T_{meta}$ **then**

    $\mathcal{A}_t \leftarrow$ `MetaCtrl`(V_{long}) `// ` $\mathcal{A}_t \in \mathbf{\Omega}$

    `BuildTopology` $(\mathcal{A}_t)$ `// zero-downtime reconfiguration`

  `// (5) Micro control: Statera Agent computes safe admission & stability actuation`

  $(\hat{l}_{upper}, \mathcal{B}_{token}, N_{run}) \leftarrow$ `Statera`$(V_{\text{short}}, D_b, \bar{u}_t, \nu_t)$

  $\mathcal{C}_t \leftarrow (\hat{l}_{upper}, \mathcal{B}_{token}, N_{run})$

  `// (6) Execution: ordering + lane assignment + regulated dispatch`

  `Dispatch` $(\mathcal{A}_t, \mathcal{C}_t)$

error $e_r = l_{real} - l^{(r)}_{pred}$ and inserts it into a bucketed buffer $D_{b(r)}$, where $b(r)$ is determined by a lightweight bucketing function (we use prompt-length buckets for simplicity and stability). For any request considered for admission, the agent computes a $(1-\epsilon)$ quantile $q_{1-\epsilon}(D_{b(r)})$ and forms a deterministic upper bound $\hat{l}^{(r)}_{upper} = l^{(r)}_{pred} + q_{1-\epsilon}(D_{b(r)})$. This upper bound is exported to the execution layer as an admission fence: the runtime

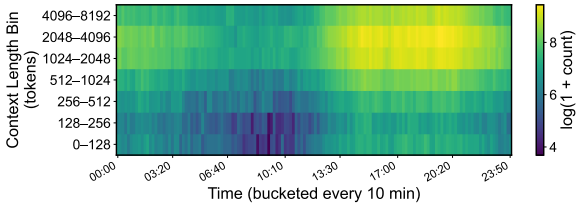


Figure 11: Heatmap of context-length distribution over a 24-hour period in the Azure trace.

admits a set of concurrent requests $Q_{run}(t)$ only if $\sum_{r \in Q_{run}(t)} \hat{l}^{(r)}_{upper} \leq M_{cap}(1 - \delta_{frag})$ (Eq. 2), thereby bounding the probability of OOM events by the target risk tolerance ϵ while reserving a fragmentation margin δ_{frag}.

The stability component uses a dual-loop band-tracking controller driven by the short-horizon view V_{short}. We track window-averaged GPU utilization $\bar{u}_t$ (and optionally short-window violation indicators included in V_{short}) against a target band $[u_{low}, u_{high}]$. Let $u^\star_t$ be the projection of $\bar{u}_t$ onto this band and define the tracking error $e_t = \bar{u}_t - u^\star_t$. The inner loop applies proportional actuation to the per-cycle token budget: $\mathcal{B}_{token} \leftarrow \text{Proj}_{[B_{\min}, B_{\max}]}\big(\mathcal{B}_{token} - \eta_B e_t\big)$, which reacts quickly to transient over-utilization and damps short-term jitter. The outer loop adjusts the concurrency cap using an averaged error $\bar{e}_t = \frac{1}{W}\sum_{i=0}^{W-1} e_{t-i}$: $N_{run} \leftarrow \text{Proj}_{[N_{\min}, N_{\max}]}\big(N_{run} - \eta_N \bar{e}_t\big)$, so concurrency changes only under persistent deviation. Together, these two loops stabilize execution while respecting hard safety limits imposed by the admission fence.

E.3 Azure trace characterization

Fig. 11 characterizes the trace by visualizing the context-length distribution over a full day. We bucket requests by time (10-minute intervals) and group them into logarithmic context-length bins. The heatmap shows clear time-of-day drift: both the dominant length region and the mass of long-context requests vary substantially across hours, indicating that the workload is not stationary even within a single day.

E.4 Latency masking

Figure 12 illustrates why the cognition overhead of H-MAS does not necessarily translate into additional end-to-end delay under load. In the baseline scenario (Fig. 12(a)), a newly arrived request is queued until an execution slot becomes available, incurring unavoidable queueing latency.

Algorithm 2: Statera Agent: conformal admission fence + dual-loop stability control.

Input : Short-horizon view V_{short} containing $(\bar{u}_t, \nu_t)$ and recent completions; bucketed error buffers $\{D_b\}$; safety params $(\epsilon, M_{cap}, \delta_{frag})$; bounds $(B_{\min}, B_{\max}, N_{\min}, N_{\max})$; controller steps (η_B, η_N); band params (u_{low}, u_{high}).

Output : Per-request upper bound $\hat{l}^{(r)}_{upper}$, token budget $\mathcal{B}_{token}$, and concurrency cap N_{run}.

`// (A) Online conformal calibration: update bucketed error buffers`

foreach *completed request r in this window* **do**

  $e_r \leftarrow l^{(r)}_{real} - l^{(r)}_{pred}$

  $b \leftarrow \texttt{Bucket}(\text{prompt length of } r)$

  $D_b \leftarrow \texttt{Update}(D_b, e_r)$

`// (B) Compute per-request safe upper bound (Admission Fence)`

foreach *pending/running request r considered for admission* **do**

  $b \leftarrow \texttt{Bucket}(\text{prompt length of } r)$

  $q \leftarrow \texttt{Quantile}(D_b, 1-\epsilon)$

  $\hat{l}^{(r)}_{upper} \leftarrow l^{(r)}_{pred} + q$

`// Admission is enforced by execution via:` $\sum_{r \in Q_{run}(t)} \hat{l}^{(r)}_{upper} \le M_{cap}(1-\delta_{frag})$.

`// (C) Dual-loop stability control (band tracking)`

$u^\star_t \leftarrow \text{clip}(\bar{u}_t, [u_{low}, u_{high}])$ `// nearest point in target band`

$e_t \leftarrow \bar{u}_t - u^\star_t$

`// Inner loop: token-rate actuation (P-like)`

$\mathcal{B}_{token} \leftarrow \texttt{Proj}_{[B_{\min}, B_{\max}]}(\mathcal{B}_{token} - \eta_B\, e_t)$

`// Outer loop: concurrency actuation (I-like on window-averaged error)`

$\bar{e}_t \leftarrow \frac{1}{W}\sum_{i=0}^{W-1} e_{t-i}$

$N_{run} \leftarrow \texttt{Proj}_{[N_{\min}, N_{\max}]}(N_{run} - \eta_N\, \bar{e}_t)$

return $(\hat{l}_{upper}, \mathcal{B}_{token}, N_{run})$

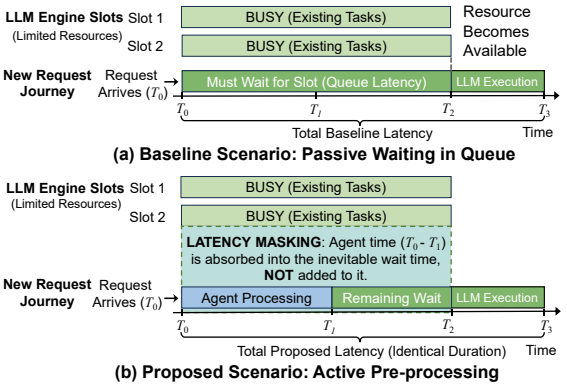


Figure 12: Latency masking through concurrent agent pre-processing under resource contention.

H-MAS exploits this waiting window by running Perception/Prediction and safety preparation asynchronously while the request is pending (Fig. 12(b)). As long as the agent processing time falls within the original queueing interval, it is absorbed by the idle wait rather than added on the critical path. In our implementation, this overlap is enabled by decoupling the cognition execution from the GPU inference slots and invoking the agents only when the run queue is non-empty.

We also instrument the Perception/Prediction path and the lightweight runtime control logic. Since Perception/Prediction runs in parallel with backend queueing, it affects end-to-end latency only when its output is not ready by the time a request becomes eligible for dispatch; otherwise, its cost is fully hidden by the waiting time. In a 100-request microbenchmark, the Perception+Prediction path takes 1.64 ms per request on average (0.164 s in total), with p80 $\le$ 2 ms and p98 $\le$ 4 ms. This corresponds to a single-worker processing capacity of roughly 250 RPS at p98 and over 600 RPS on average. Under typical serving loads (e.g., 30–50 RPS), backend queueing and GPU execution dominate the critical path, and the front-end typically completes well before requests reach the head of the dispatch queue, so its overhead is effectively masked. To avoid the front-end becoming a bottleneck under extreme bursts, we implement a non-blocking fallback: if the agent output is unavailable at dispatch time, the scheduler falls back to default estimates, ensuring that the control plane never delays admission. Overall, the overhead is negligible in practice and is mainly due to structured input preparation rather than the scheduling logic itself.

E.5 Workload construction and per-request SLO annotation

We build a production-style benchmark by sampling prompts from Coding and ShareGPT and annotating each request with semantic attributes and heterogeneous SLO targets. For each request r, we infer its semantic task type τ_r using a hybrid "script+LLM" labeler. Lightweight rules handle explicit patterns (e.g., code fences and translation cues), while an LLM resolves ambiguous cases and maps requests into a fixed taxonomy. Each request is annotated with (i) a task type τ_r and (ii) an importance score $\text{imp}_r \in [0.5, 1.0]$.

We specify a task-to-SLO baseline table $\text{SLO\_BASE}(\tau)$ that assigns each task type τ a pair of reference targets $(\text{TTFT}^{\text{base}}_\tau, \text{TPOT}^{\text{base}}_\tau)$. Table 6 lists the baseline templates used in our

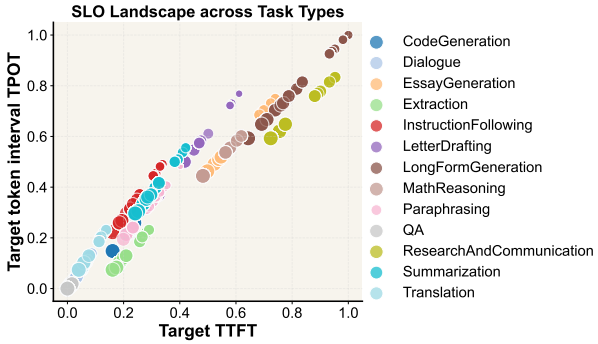


Figure 13: Task-level SLO configuration in the TTFT–TPOT space.

Table 6: Baseline per-task SLO templates used for annotating the mixed workload. TTFT is measured in ms; TPOT is measured in ms/token.

Task Type	Base TTFT (ms)	Base TPOT (ms/token)
QA	4000	70
Dialogue	4000	70
Translation	4500	80
Extraction	6000	80
Paraphrasing	6000	90
CodeGeneration	6000	90
InstructionFollowing	6000	100
Summarization	7000	110
LetterDrafting	8000	120
MathReasoning	10000	130
EssayGeneration	10000	130
LongFormGeneration	12000	150
ResearchAndCommunication	13000	150

workload annotation. This table is a reproducible template for emulating heterogeneous production constraints rather than an oracle of correct latency targets.

We set $\mathrm{SLO\_BASE}(\tau)$ using a two-step procedure. First, we impose a human-centric tolerance hierarchy across task types: interactive tasks (e.g., QA, Dialogue, Translation) use stricter baselines, while long-form and research-style generation is assigned looser targets. Second, we calibrate the numeric values at a representative operating point defined by the Azure trace mean arrival rate (mean RPS). This choice avoids the trivial regime where the system is under-utilized and nearly all requests satisfy SLOs, and also avoids the overload regime where most requests violate SLOs.

To introduce within-type diversity, we scale the baseline targets using imp_r. We apply a clipped linear factor $f(\mathrm{imp}_r) \in [1-\alpha,\, 1+\alpha]$ with $\alpha = 0.2$, where higher importance yields a smaller factor and therefore stricter latency targets. The final per-request targets are:

$$\begin{aligned} SLO_{ttft}^{(r)} &= \left\lfloor \mathrm{TTFT}_{\tau_r}^{\mathrm{base}} \cdot f(\mathrm{imp}_r) \right\rfloor, \\ SLO_{tpot}^{(r)} &= \left\lfloor \mathrm{TPOT}_{\tau_r}^{\mathrm{base}} \cdot f(\mathrm{imp}_r) \right\rfloor. \end{aligned} \tag{14}$$

We store the annotated fields (e.g., `task_type`, `importance`, `ttft`, `tpot`) together with the prompt into a Parquet dataset for replay.

Fig. 13 visualizes a representative portion of the requests in the SLO landscape in the TTFT–TPOT space, where each point corresponds to one request and is colored by τ_r. The plot highlights a structured spectrum. Short interactive requests concentrate in the low-TTFT and low-TPOT region, while long-form or reasoning-heavy tasks shift toward more relaxed targets.