HEXAGENT: WORKFLOW- AND HETEROGENEITY-AWARE SCHEDULING FOR AGENTIC LLM SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Serving agentic LLM applications requires scheduling dependent model calls to meet service-level objectives (SLOs) for complete user workflows. This is difficult when calls and branches emerge during execution, external tools delay progress, and prefill–decode (P-D) disaggregation places each call on separate, heterogeneous workers. Priorities based only on individual calls overlook workflow dependencies, while workflow priorities alone do not account for placement-dependent queueing, transfer, and KV-capacity costs. We present HEXAGENT, a workflow- and heterogeneity-aware scheduler that couples workflow progress with joint P-D placement. HEXAGENT maintains a critical-path horizon over revealed calls and completed tool phases, then normalizes projected call completion from workflow arrival by this horizon. It uses the resulting priorities to plan queue order and worker pairs, updating projected queues and capacity after each assignment. Local and global KV-prefix reuse refine per-worker prefill costs, and asynchronous plan application allows workers to continue serving during scheduling. In trace-driven simulation with two LLMs, four agentic workloads, and three homogeneous and heterogeneous cluster settings, HEXAGENT reduces the SLO scaling factor required for 95% and 99% workflow attainment by up to 49.7% and 63.0%, respectively, relative to the best evaluated baseline at each target. Gains also persist on homogeneous workers, and additional experiments characterize sensitivity to service-time estimation error and aggregate planning cost.

1 INTRODUCTION

Agentic LLM applications turn an end-to-end user request into a workflow of dependent LLM calls. An agent may invoke tools, revise intermediate outputs, or branch into parallel LLM calls before returning an answer (Yao et al., 2023b; Madaan et al., 2023; Besta et al., 2024). The serving objective is therefore end-to-end workflow completion under a service-level objective (SLO), rather than the latency of an isolated LLM call. In this paper, we study *the online scheduling problem of completing competing agentic LLM workflows under end-to-end latency service-level objectives (SLOs) on heterogeneous prefill-decode (P-D) disaggregated clusters.*

Such scheduling is essential because local serving efficiency does not determine when a user receives an answer. Delaying one conversational turn postpones its successors, while a join cannot proceed until every required branch completes. Consequently, calls with similar token counts can have different effects on workflow progress. P-D disaggregation adds another dependency within each call: completing prefill helps only when KV transfer and downstream decode can also proceed. Scheduling must therefore connect progress across calls with execution across both serving stages.

Scheduling these workflows on heterogeneous prefill-decode (P-D) disaggregated clusters poses two coupled challenges. First, priorities must reflect end-to-end workflow progress even when future calls are unknown. Calls and branches are revealed during execution, so the scheduler cannot rely on a complete dependency graph. Accelerating one call may not advance a workflow blocked elsewhere, and tool invocations add workflow time without GPU service. Second, priorities must be evaluated together with heterogeneous P-D placement. A call's completion depends on worker service times, downstream queues, transfer costs, and available KV reuse. The fastest prefill worker may still feed a congested decode queue; workflow urgency alone cannot identify the best worker pair.

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

Existing serving approaches leave a gap between workflow-level objectives and P-D execution decisions. LLM-call-level systems optimize batching and KV management (Kwon et al., 2023; Zheng et al., 2024), but local efficiency does not identify which call to prioritize for workflow completion. P-D systems specialize phase-specific resources (Zhong et al., 2024; Patel et al., 2024), yet phase efficiency alone does not determine which dynamic workflow should advance. Program-aware systems coordinate calls using application structure (Lin et al., 2024; Luo et al., 2025; Li et al., 2025), but workflow priority alone does not resolve placement under different worker speeds, queues, and KV locality. The missing link is to derive priorities from revealed workflow progress and evaluate them together with P-D placements.

We present HEXAGENT, a workflow- and heterogeneity-aware scheduler for agentic LLM serving. For workflow prioritization, HEXAGENT updates a critical-path horizon as calls are revealed and tool durations become known, then ranks waiting calls by projected completion from workflow arrival relative to this horizon. For joint P-D planning, it evaluates candidate workers using KV-aware prefill costs, transfer latency, and downstream decode availability, updating these estimates after each assignment. Both decisions use these updated completion estimates. Planning is asynchronous and does not interrupt calls already in service. Concretely, we make the following contributions:

- **Prioritization under partial workflow revelation.** We formulate a workflow-level SLO-attainment objective and develop a revealed-horizon priority heuristic that relates elapsed workflow time and projected call completion to known dependency-path duration. Completed tool phases inform progress without becoming GPU work or requiring future-duration predictions.
- **Joint workflow and P-D scheduling.** We design a planner that uses the same completion projections to inform call priorities and evaluate heterogeneous worker pairs, with KV-aware prefill costs and successive updates to projected queues and capacity. Asynchronous execution and state-validated plan application integrate these decisions without interrupting calls already in service.
- **Evaluation across workflow and hardware settings.** In trace-driven simulation covering two LLMs, four workflow traces, and three cluster settings, HEXAGENT reduces the required SLO scaling factor by up to 49.7% at 95% attainment and 63.0% at 99% attainment relative to the best evaluated baseline at each target (Section 6). Further experiments characterize estimation sensitivity and planning cost.

2 BACKGROUND AND RELATED WORK

This section places HEXAGENT relative to LLM serving, agent-aware scheduling, and classical workflow scheduling. A fuller comparison appears in Appendix A.

LLM serving and P-D disaggregation. ORCA (Yu et al., 2022), vLLM (Kwon et al., 2023), SGLang (Zheng et al., 2024), and Sarathi-Serve (Agrawal et al., 2024) optimize LLM-call batching, KV management, and local scheduling. DistServe (Zhong et al., 2024), Splitwise (Patel et al., 2024), and related systems (Jin et al., 2024) separate phases with different bottlenecks. These systems supply the execution substrate; HEXAGENT decides which LLM call should advance and which prefill and decode workers should serve it.

Program-aware and agentic serving. Parrot (Lin et al., 2024), Hermes (Liu et al., 2026), Autellix (Luo et al., 2025), and Continuum (Li et al., 2025) elevate scheduling from isolated calls to programs or workflows. SmoothAgent (Pan et al., 2026) uses slack-aware admission for best-effort lookahead transformations, while SMetric (Wang et al., 2026) exploits KV reuse across related LLM calls. HEXAGENT instead optimizes completion of competing dynamic workflows while jointly selecting heterogeneous P-D placements; its tool and cache extensions modify horizon and service estimates rather than adding a second resource scheduler.

Classical scheduling boundary. Least-laxity-first (LLF), shortest-remaining-processing-time, and DAG scheduling are established ideas (Leung, 1989; Stankovic et al., 1998; Schrage, 1968). Dynamic execution engines such as CIEL (Murray et al., 2011), Ray (Moritz et al., 2018), and Timely Dataflow (Murray et al., 2013) reveal work dynamically, while Borg (Verma et al., 2015) and Sparrow (Ousterhout et al., 2013) highlight the cost of maintaining cluster-wide scheduling state. Our focus is how dynamically revealed workflow progress guides call priorities and heterogeneous P-D placements, with tool delays and KV reuse reflected in the estimates used for those decisions.

3 MOTIVATION

The two challenges imply two scheduling requirements: assess workflow progress using incomplete information, and account for placement when deciding how to advance it. We illustrate these requirements with multi-turn conversations from ShareGPT (Chiang et al., 2023), function-calling interactions from BFCL v3 (Patil et al., 2025), branching retrieval and search from LATS (Zhou et al., 2024), and terminal tasks from Terminal-Bench (Merrill et al., 2026).

Workflow prioritization with incomplete information. Later calls emerge during execution, yet scheduling must advance the whole workflow. Delaying a conversational or function-calling turn postpones its successors. In a branching workflow, a parent can release several calls, while a join waits for all required branches. Calls with similar service times can therefore have different effects on workflow progress. The scheduler must compare them using revealed dependencies, rather than local latency alone or an assumed complete graph.

A workflow may also wait for an external tool with no runnable LLM call. This delay contributes to workflow completion but is neither GPU queueing nor GPU service. BFCL captures function-call dependencies; Terminal-Bench additionally provides tool durations, allowing external blocking to be distinguished from P-D serving delay.

Insight 1. Prioritize LLM calls using revealed workflow progress rather than isolated call latency. Track known dependencies and observed tool durations, without assuming future calls are known or treating tool execution as GPU service.

Workflow priority and P-D placement are coupled. Prompt-heavy and generation-heavy calls stress different P-D resources, and heterogeneous GPUs serve them at different speeds. A fast prefill choice can incur a costly transfer or feed a congested decode queue. KV reuse adds a choice between local prefix reuse, global-prefix fetch, and recomputation; waiting for a cache-local worker is not always the best choice. Each assignment also changes the queues seen by later calls, so priority and placement cannot be evaluated separately.

Insight 2. Combine workflow priority with predicted completion across both P-D stages. Account for heterogeneous worker speeds, KV reuse, transfer, queues, and capacity, and update these estimates after each assignment.

4 SYSTEM OVERVIEW

This section introduces HEXAGENT's workflow representation, serving architecture, and execution lifecycle. We describe how dependencies identify runnable LLM calls, how workflow and worker state support priority and placement decisions, and how execution events update those decisions.

4.1 DYNAMIC WORKFLOW AND TOOL ABSTRACTION

A workflow comprises dependent LLM calls and external tool invocations. HEXAGENT tracks known calls and schedules those with completed dependencies.

Dynamically revealed graph. We represent workflow w by a directed acyclic graph $G_w = (V_w, E_w)$, where V_w contains LLM calls and E_w contains their dependencies. The scheduler sees only the revealed subgraph $G_w(t) = (V_w(t), E_w(t))$ at time t. Source calls are revealed at arrival; subsequent calls are revealed as their parents and any intervening tools complete. Each call records its workflow identifier, input length, estimated output length, dependencies, and KV-cache metadata. Completed calls remain in the graph as execution history, while future calls are not assumed to be known.

Runnable frontier. The runnable frontier contains revealed LLM calls that have not started prefill and whose dependencies have completed. Figure 1 marks these runnable calls. A call leaves the frontier when prefill starts; it is later scheduled again when waiting for decode.

Tool completion and KV reuse. Tools execute outside the P-D workers. A `TOOL_COMPLETE` event records the observed duration and releases eligible dependent calls; no future tool duration is predicted. For each new LLM call, cache metadata identifies the reusable prefix, its local prefill worker, and global-copy availability. The scheduler uses this information to estimate placement costs. A cache context identifies reusable KV; the cache manager controls residency.

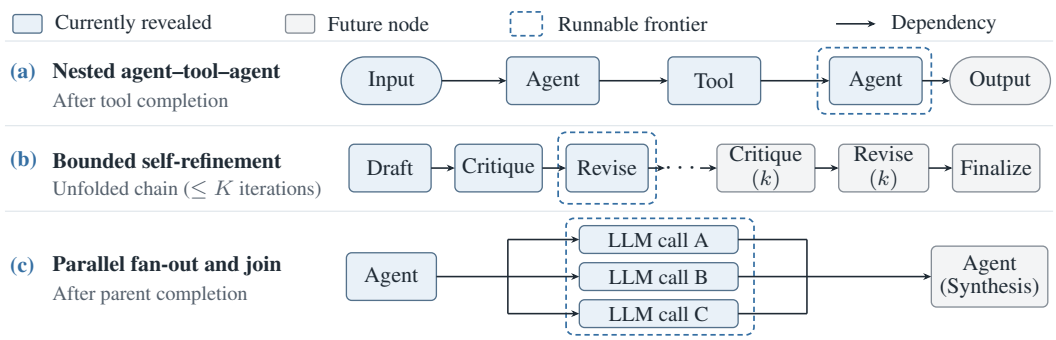

Figure 1: Dynamic workflow revelation. Dashed outlines mark runnable LLM calls; their blue predecessors have completed. Gray nodes illustrate possible future execution, not scheduler knowledge. Tools and input/output provide context; $k \leq K$ indexes a later refinement round.

4.2 Serving Architecture and Scheduler Placement

Figure 2 shows the four major components and their information flow, from workflow submission to LLM-call execution and completion feedback.

Components and state. The *Workflow Front-End* manages dependencies and tool-completion events, stores workflow progress, and submits runnable LLM calls. The *Global Scheduler* processes this information through four modules: the *State Collector* reads workflow progress and worker state; the *Estimator* predicts service times and KV demand; the *Joint Planner* chooses P-D placement and queue order; and the *Plan Dispatcher* applies decisions that remain valid. The *Disaggregated P-D Serving Cluster* executes calls on prefill and decode workers connected by the KV-transfer network, and reports queues, running calls, and available KV capacity. *External Services* provide model weights and tools; tool completion is reported to the front-end.

Runtime estimates. For LLM call c, $L_{\text{in}}(c)$ is its input-token count and $\widehat{L}_{\text{out}}(c)$ its estimated output-token count. The Estimator combines these lengths with model and hardware parameters. A roofline-style model takes the larger of compute time and memory-access time to estimate prefill and decode service (Williams et al., 2009; Cao et al., 2025). KV size, link bandwidth, and transfer startup cost determine KV-transfer latency; input plus estimated output length determines decode KV demand. A deployment can obtain $\widehat{L}_{\text{out}}(c)$ from a proxy predictor (Qiu et al., 2024); our trace-driven evaluation separately perturbs service-time estimates in Section 6.4.

4.3 End-to-End Workflow Lifecycle

The lifecycle determines when LLM calls enter P-D queues, when tools release dependent work, and when the scheduler revises its assignments.

Execution and tools. An LLM call waits for prefill, executes prefill, transfers KV to its decode worker, and waits for sufficient KV capacity before decoding. Decode completion releases eligible child calls, subject to any required tool invocations. The event-driven simulator models this lifecycle and supplies queue and capacity state for scheduling; Appendix B lists the state transitions.

Asynchronous planning. Workflow arrivals and newly runnable LLM calls trigger prefill planning; KV-transfer completion triggers decode planning. The planner computes assignments from a snapshot, a copy of current workflow and worker state. Workers continue executing, and a lightweight fallback policy can assign waiting calls before the plan returns. The dispatcher then applies actions that remain feasible. Appendix C.1 describes plan precedence and the application of still-feasible older actions.

5 Scheduling Framework

HexAGenT implements the two insights in Section 3: prioritization from revealed workflow progress and joint P-D planning. After defining the objective (Section 5.1), we derive placement costs and call-completion estimates (Sections 5.2 and 5.3). Section 5.4 combines these estimates with the revealed horizon to rank and place calls. Throughout, c is an LLM call in workflow w; $p \in \mathcal{P}$ and $d \in \mathcal{D}$ denote prefill and decode workers. Times are in seconds; see Appendix D for notation.

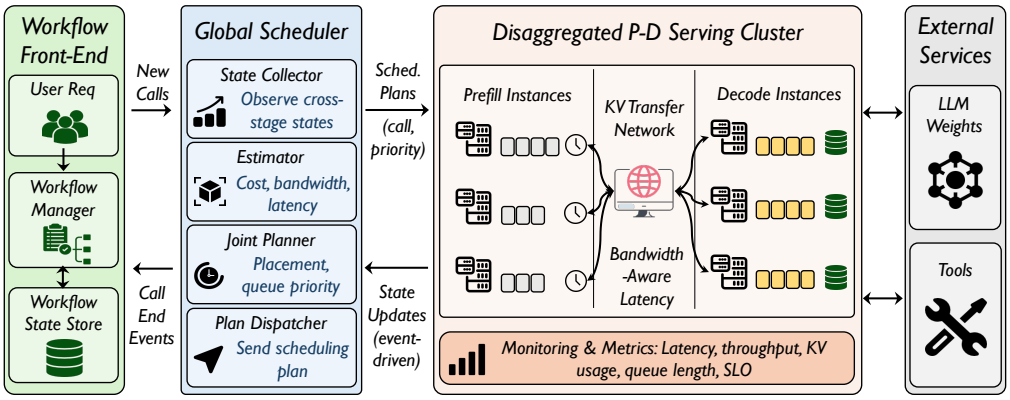

Figure 2: HEXAGENT architecture. Workflow state and P-D worker state inform scheduling decisions; execution and tool-completion events update the workflow front-end.

5.1 PROBLEM FORMULATION

Workflow completion objective. Let $\mathcal{W}$ be a nonempty set of workflows and π a scheduling policy. Workflow w arrives at a_w and completes after duration C_w^π, measured from arrival. Its end-to-end latency SLO is $S_w > 0$, and for a scaling factor $\alpha > 0$, the scaled SLO is αS_w. Given attainment target $\tau \in (0, 1]$, the factor required by policy π is

$$\alpha_\tau^\pi = \inf \left\{ \alpha > 0 : \frac{1}{|\mathcal{W}|} \sum_{w \in \mathcal{W}} \mathbb{1}[C_w^\pi \le \alpha S_w] \ge \tau \right\}. \tag{1}$$

Here $|\mathcal{W}|$ counts workflows and $\mathbb{1}[\cdot]$ is the indicator function. We seek a policy that minimizes α_τ^π, meeting the same attainment target under tighter scaled SLOs.

Available information and decisions. At time t, the scheduler is given revealed workflow graphs $G_w(t)$, completed tool durations, and current worker and cache state. It chooses queue order and P-D placements for waiting LLM calls, subject to dependencies, worker compatibility, and decode KV capacity. Unrevealed calls and future tool durations are unavailable.

5.2 KV-AWARE PREFILL ESTIMATION

Joint P-D planning must account for how KV locality changes prefill work. We estimate each worker's cost, then use it to project completion in Section 5.3.

Per-worker prefill costs. Let h_c be the reusable-prefix length of call c, with $0 \le h_c \le L_{\text{in}}(c)$, and let $p_{\text{cache}}(c, t)$ identify its cache-local prefill worker when one exists. Write $T_{\text{P}}(n, p)$ for estimated prefill service on worker p for n input tokens. The suffix-length estimate uses $n_c = \max\{1, L_{\text{in}}(c) - h_c\}$, retaining a one-token minimum. The three execution costs are

$$\begin{aligned} T_{\text{full}}(c, p) &= T_{\text{P}}(L_{\text{in}}(c), p), \\ T_{\text{local}}(c, p) &= T_{\text{P}}(n_c, p), \\ T_{\text{global}}(c, p) &= \tau_G + h_c / B_G + T_{\text{P}}(n_c, p). \end{aligned} \tag{2}$$

Here τ_G is global-fetch startup time and B_G is effective transfer rate in KV tokens/s. Local reuse is available only at $p = p_{\text{cache}}(c, t)$; global reuse requires a compatible cached prefix. Unavailable costs are $+\infty$. The effective prefill cost is

$$\widetilde{T}_{\text{P}}(c, p, t) = \min\{T_{\text{full}}(c, p), T_{\text{local}}(c, p), T_{\text{global}}(c, p)\}. \tag{3}$$

The time argument reflects cache availability. Following SMetric (Wang et al., 2026), the minimum permits recomputation when fetching costs more, without binding follow-up calls to their cache-local worker. The cache manager controls residency and eviction.

5.3 PROJECTING CALL COMPLETION

For a call waiting at stage $s \in \{\text{P}, \text{D}\}$, $\Delta_s(c, t)$ estimates the time from t until decode completes, including queueing. The stages are prefill (P) and decode (D). These projections provide the completion-time input to the priority score in Section 5.4.

Calls waiting for prefill. A candidate placement (p, d) specifies both workers. Let $\mathcal{M}_c(t) \subseteq \mathcal{P} \times \mathcal{D}$ contain the feasible worker pairs for call c. Its minimum projected latency is

$$\Delta_{\mathrm{P}}(c,t) = \min_{(p,d)\in\mathcal{M}_c(t)} \big\{ Q_{\mathrm{P}}(c,p,t) + \widetilde{T}_{\mathrm{P}}(c,p,t) + T_{\mathrm{PD}}(c,p,d) + \widehat{Q}_{\mathrm{D}}(c,p,d,t) + T_{\mathrm{D}}(c,d) \big\}. \tag{4}$$

The five terms follow the call's execution: Q_{P} is its wait for prefill; $\widetilde{T}_{\mathrm{P}}$ is the KV-aware prefill cost from Equation 3; T_{PD} is KV-transfer latency; $\widehat{Q}_{\mathrm{D}}$ is the wait from KV arrival until decode starts; and T_{D} is decode service time. The decode wait depends on when (p, d) delivers the KV, not just on the decode queue at time t.

Calls waiting for decode. Only decode queueing and service remain:

$$\Delta_{\mathrm{D}}(c,t) = \min_{d\in\mathcal{D}_c(t)} \{ Q_{\mathrm{D}}(c,d,t) + T_{\mathrm{D}}(c,d) \}. \tag{5}$$

Here $\mathcal{D}_c(t)$ contains feasible decode destinations, and Q_{D} is the wait from t until decode starts. A transferred call retains its destination. Both projections enforce worker compatibility and decode KV demand $m(c) = L_{\mathrm{in}}(c) + \widehat{L}_{\mathrm{out}}(c)$, in KV tokens, throughout the predicted decode interval. Calls without a feasible placement remain waiting.

5.4 Workflow Prioritization and Joint Planning

Insight 1 motivates a priority score based on elapsed workflow time and the revealed horizon. The projected completion term comes from Section 5.3, further coupling this score to the P-D placement costs required by Insight 2.

Revealed workflow horizon. Let $\mathcal{A}_w(t)$ be the set of directed paths in $G_w(t)$. For each revealed call c, $\ell(c)$ is its estimated standalone prefill, KV-transfer, and decode duration, using the fastest estimated service in each worker pool and excluding queueing. Let $u(c,t)$ be the observed duration of the tool phase following c if that phase has completed, and zero otherwise. The horizon is

$$H_w(t) = \max_{A\in\mathcal{A}_w(t)} \sum_{c\in A} \big(\ell(c) + u(c,t)\big). \tag{6}$$

It measures the duration of the longest revealed dependency path, including history and current calls but no unrevealed descendants or predicted time for active tools. Tool completion updates $H_w(t)$ before newly runnable calls are ranked.

Projected normalized completion. Using $\Delta_s(c,t)$ from Equations 4 and 5, we compute

$$R_s(c,t) = \frac{(t - a_w) + \Delta_s(c,t)}{H_w(t)}. \tag{7}$$

The numerator is the projected completion of call c, measured from workflow arrival; the denominator is its workflow's revealed horizon. Larger scores indicate more elapsed and projected time relative to revealed work. This is a priority heuristic, not a prediction of final workflow completion.

Priority-guided placement and state updates. At each stage, the ratio-based path sorts its waiting calls by decreasing $R_s(c,t)$, with deterministic tie-breaking, and assigns them one by one to their best feasible placements. Before evaluating the next call, each tentative assignment updates projected prefill availability, KV-transfer completion, and decode-capacity usage. Thus later calls see running work and earlier assignments, rather than all choosing the same apparently idle worker. Small queues allow re-ranking after each assignment; large queues retain the initial order, but still update projected state and placement costs for every call.

Bounded refinement. Pairwise exchanges refine the order and recompute placement. Plans are compared first by the largest projected workflow ratio, then its mean, then total projected call latency. Appendix C gives the scheduling loop, and Appendix C.1 describes how feasible returned actions are applied without interrupting calls already in service.

6 Evaluation

We evaluate tail SLO scaling factors on heterogeneous and homogeneous clusters, robustness to service-time estimation error, and the cost of asynchronous planning.

6.1 Experimental Setup

All schedulers use the same SGLang v0.5.15 (Zheng et al., 2024) serving adapter, P-D runtime state, service-time inputs, decode-capacity constraints, and asynchronous plan-ready mechanism; only scheduling policy differs.

Traces and workflow-level arrival rates. The configured arrival rate counts complete workflows, not individual LLM calls. ShareGPT (Chiang et al., 2023) uses 100 conversational workflows at 24.0 workflows/s; BFCL v3 (Patil et al., 2025) uses 100 multi-turn function-calling workflows at 0.3 workflows/s; LATS (Zhou et al., 2024) on HotpotQA (Yang et al., 2018) uses 100 branching workflows at 1.0 workflow/s; and Terminal-Bench (Merrill et al., 2026) uses 100 LLM–terminal workflows at 5.0 workflows/s. BFCL retains LLM input/output profiles and dependency structure but not downstream API wall time, whereas Terminal-Bench contributes observed terminal-action delay. Arrival rates differ because workflow call counts, token lengths, branching, and tool waits produce different loads on P-D workers.

Models, cluster settings, and P/D splits. We evaluate Llama-3.1-70B and Qwen3-235B-A22B on three fixed hardware compositions: Homo has six A100 workers, Hetero-1 has five A100 plus five H100 workers, and Hetero-2 has two each of A100, H100, and H200. Table 1 assigns the fixed budget to prefill and decode separately for each trace; A, H, and X denote A100, H100, and H200. The split varies because prompt work, output work, concurrency, and KV pressure stress the two phases differently, consistent with workload-dependent phase allocation in P-D systems (Zhong et al., 2024; Patel et al., 2024; Li et al., 2026; Liao et al., 2025).

We implement three comparison policies over the same SGLang serving interface (Zheng et al., 2024). They adapt classical least-laxity-first (LLF) and first-come, first-served (FCFS) ordering to LLM calls or workflows; they are not built-in SGLang policies. Laxity is the time until a target completion deadline minus estimated remaining service.

- **SGLang-LLF**: LLM-call-level LLF (Leung, 1989), using each waiting call's deadline and estimated stage service. Worker dispatch uses power-of-two choices (Azar et al., 1994).
- **Workflow-LLF**: LLF applied at workflow level; waiting LLM calls inherit workflow laxity using its revealed graph and completed service. Dispatch is round-robin.
- **Workflow-FCFS**: FCFS applied at workflow level (Arpaci-Dusseau & Arpaci-Dusseau, 2023); waiting LLM calls are ordered by workflow arrival, with revelation order breaking ties. Worker dispatch is round-robin.

All methods use the same arrivals, service-time inputs, P-D split, capacity constraints, and recorded tool delays; planner execution time is applied asynchronously.

Scaled SLO. For evaluation, each workflow's SLO is its standalone critical-path duration on H100, including prefill, KV transfer, decode, and recorded tool time but excluding queueing. This reference is fixed across policies and cluster settings. Req95 and Req99 are the SLO scaling factors α needed to achieve 95% and 99% workflow attainment; lower is better.

Scheduler overhead. We measure total planner execution time and normalize it by completed workflows. This accounting covers all invocations in a run; because planning is asynchronous, it affects plan freshness rather than pausing active service.

6.2 End-to-End Evaluation with Heterogeneity

Tail improvement across mixed-generation clusters. Figure 3 shows SLO-attainment curves for Hetero-1 and Hetero-2. Req95 reductions range from 2.5% to 41.9% on Hetero-1 and from 5.4% to 49.7% on Hetero-2; Req99 reductions are 2.0%–62.9% and 1.2%–63.0%, respectively.

Representative settings. On Qwen/BFCL in Hetero-1, HexaGen lowers Req95 from 18.92 to 11.00 and Req99 from 29.99 to 11.12, corresponding to 41.9% and 62.9%. On Qwen/ShareGPT in Hetero-2, Req95 falls from 15.32 to 7.71 and Req99 from 17.70 to 8.00 (49.7% and 54.8%). For Qwen/Terminal-Bench on Hetero-1, Req95 and Req99 improve by 19.2% and 22.4%.

Trace-level behavior. Across both heterogeneous settings and models, ShareGPT reductions span 5.7%–49.7% at Req95 and 12.7%–54.8% at Req99. Delays on one turn propagate through the conversational chain. BFCL spans 3.3%–41.9% and 5.9%–63.0%: repeated prompt-side placement

Table 1: Fixed cluster compositions and trace-specific P/D splits. Hardware totals stay unchanged within each setting.

Setting	Composition	ShareGPT	BFCL v3	LATS	Terminal-Bench
Homo	6A	1P+5D	4P+2D	2P+4D	1P+5D
Hetero-1	5A+5H	P:1A; D:4A+5H	P:4A+5H; D:1A	P:2A+1H; D:3A+4H	P:1A; D:4A+5H
Hetero-2	2A+2H+2X	P:2A+1H+2X; D:1H	P:1A+2H+2X; D:1A	P:1H+1X; D:2A+1H+1X	P:1A; D:1A+2H+2X

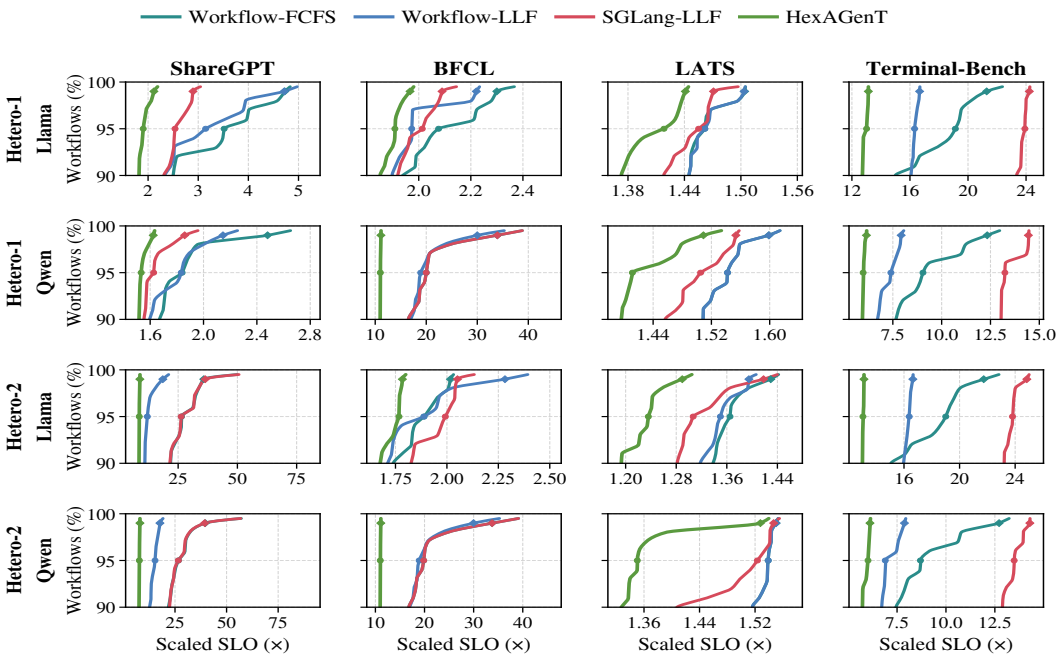

Figure 3: Workflow SLO-attainment on Hetero-1 and Hetero-2. Columns denote traces and rows model/setting pairs. The horizontal axis is the SLO scaling factor; lower is better.

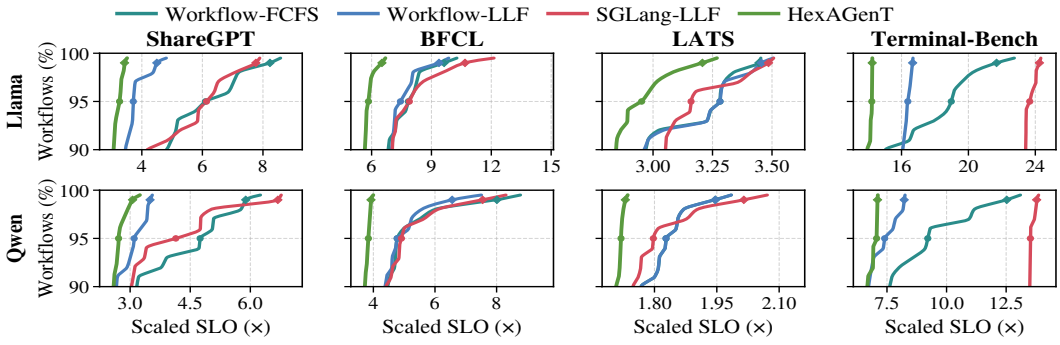

Figure 4: Workflow SLO-attainment on Homo. Columns show the four traces and rows the two models. Plotting conventions match Figure 3.

decisions affect dependent function-calling stages. Terminal-Bench shows 12.8%–20.4% Req95 and 21.1%–22.4% Req99 reduction, with observed tool delays included in workflow progress. The comparison does not isolate tool accounting alone. LATS has the smallest margin (2.5%–11.4% at Req95 and 1.2%–7.6% at Req99), consistent with its small frontier giving LLF-based baselines useful local priorities.

Interpretation. The results are consistent with workflow priority and placement complementing each other. A fast prefill worker may feed a congested decode queue, while an urgent workflow may receive a poor placement. Projecting both stages lets HEXAGENT compare these downstream costs when ordering calls, rather than treating priority and placement independently.

6.3 Evaluation on the Homogeneous Cluster

On the six-A100 Homo cluster (Figure 4), HexAGenT also improves workflow completion. Req95 reductions range from 4.3% to 21.6%, and Req99 reductions from 6.9% to 40.2%. Qwen/BFCL is representative: HexAGenT lowers Req95 from 4.77 to 3.84 and Req99 from 6.56 to 3.92. Terminal-Bench improves Req99 by 14.5% on Llama and 13.6% on Qwen; both models also improve on ShareGPT and LATS.

Interpretation. Identical GPUs remove hardware-speed differences, but calls still contend in P-D queues and depend on earlier calls in their workflow. The gains therefore do not require choosing a faster GPU generation: workflow ordering and queue-aware placement remain useful together. LATS's smaller margin is consistent with LLF already providing useful local priorities.

6.4 Robustness to Estimation Error

We inject deterministic multiplicative error of 10%, 20%, and 30% into scheduler-visible prefill/decode estimates while leaving actual service unchanged. Table 2 reports the controlled Hetero-1 sweep. Llama Req99 degradation remains 0.8%–1.5%; Qwen is more sensitive but remains within 5.4%–9.5%. Req95 changes range from −1.9% to 0.9%. These results indicate tolerance to the tested errors. The small negative changes are near-ties, not evidence that less accurate service-time estimates systematically improve the reported tail metrics.

Table 2: Robustness to estimation error on Hetero-1: changes (%) from the 0%-error case.

	Llama		Qwen	
Error	**Req95**	**Req99**	**Req95**	**Req99**
10%	0.9%	1.5%	−1.9%	9.5%
20%	−0.5%	0.8%	−1.1%	7.0%
30%	0.7%	1.5%	−1.4%	5.4%

6.5 Asynchronous Scheduler Overhead

Across the 24 end-to-end cases, total measured planner execution time ranges from 0.33 s to 15.86 s, or 3.26–158.59 ms per completed workflow. Table 3 reports ranges across traces. Terminal-Bench has the highest overhead in each model/setting pair, consistent with repeated LLM–tool cycles releasing additional calls for scheduling. These totals include all planning invocations; they measure aggregate planning work per completed workflow, not the latency of one invocation or a blocking delay added to each call. Workers continue serving while plans are computed, but asynchronous execution does not make planner cost irrelevant. A longer computation can return decisions based on an older snapshot. The dispatcher therefore validates each action against current state, skips calls that have already started, and applies feasible actions according to plan precedence (Appendix C.1). The reported costs characterize the tested workloads and cluster sizes; they do not establish how planning scales to substantially larger queues or clusters.

Table 3: Asynchronous planner overhead (ms/completed workflow). Each cell is the min–max range across the four traces.

Setting	**Llama**	**Qwen**
Homo	23.6–158.6	5.0–135.0
Hetero-1	3.3–155.2	5.4–142.5
Hetero-2	3.3–139.4	4.4–119.7

7 Conclusion

In this paper, we introduce HexAGenT, which addresses online agentic workflow scheduling by connecting revealed dependency progress with heterogeneous P-D execution costs. Its central design is to use placement-sensitive call-completion estimates both to prioritize workflows relative to their revealed horizon and to select prefill and decode workers. This links the incomplete application structure available at runtime to concrete resource decisions. Completed tool phases update the horizon without occupying GPU workers, KV-prefix reuse refines per-worker prefill costs, and asynchronous plan validation allows the service to continue while scheduling decisions are computed. Trace-driven simulation across two LLMs, four workflow traces, and three cluster settings shows reductions of up to 49.7% and 63.0% in the SLO scaling factors required for 95% and 99% workflow attainment, respectively, relative to the best evaluated baseline at each target. Improvements on homogeneous workers show that the benefit extends beyond selecting faster GPU generations, while the estimation-error and overhead studies characterize sensitivity and planning cost in the tested settings. These findings support jointly scheduling workflow progress and P-D resources; component-level ablations and live-serving validation remain the next steps for establishing the separate benefits of each mechanism and their behavior under deployment variability.

AI USE STATEMENT

In this work, we used generative AI tools to aid and polish writing, including language refinement, grammar checking, typo correction, and improving clarity and readability. We did not use generative AI tools to generate research ideas, design the methodology, or conduct experiments. All AI-assisted text was reviewed and revised by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency trade-off in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/agrawal.

Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, 2023. URL https://pages.cs.wisc.edu/~remzi/OSTEP/. Version 1.10, Chapter 7.

Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, pages 593–602, New York, NY, USA, 1994. Association for Computing Machinery. doi: 10.1145/195058.195412.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16), 2024.

Shiyi Cao, Shu Liu, Tyler Griggs, Peter Schafhalter, Xiaoxuan Liu, Ying Sheng, Joseph E. Gonzalez, Matei Zaharia, and Ion Stoica. MoE-Lightning: High-throughput MoE inference on memory-constrained GPUs. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 715–730. ACM, 2025. doi: 10.1145/3669940.3707267. URL https://doi.org/10.1145/3669940.3707267.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing GPT-4 with 90%* ChatGPT quality. LMSYS Blog, March 2023. URL https://lmsys.org/blog/2023-03-30-vicuna/.

Robert I. Davis, Ken W. Tindell, and Alan Burns. Scheduling slack time in fixed priority preemptive systems. In *Proceedings of the 14th IEEE Real-Time Systems Symposium*, pages 222–231, Washington, DC, USA, 1993. IEEE Computer Society. doi: 10.1109/REAL.1993.393496.

Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. ServerlessLLM: Low-latency serverless inference for large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 135–153, Santa Clara, CA, July 2024a. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/fu.

Yichao Fu, Siqi Zhu, Runlong Su, Aurick Qiao, Ion Stoica, and Hao Zhang. Efficient LLM scheduling by learning to rank. In *Advances in Neural Information Processing Systems*, volume 37, 2024b. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/6c8985579293e0209bdaa4f21bb1d237-Abstract-Conference.html.

Youhe Jiang, Ran Yan, Xiaozhe Yao, Yang Zhou, Beidi Chen, and Binhang Yuan. Hexgen: Generative inference of large language model over heterogeneous environment. *arXiv preprint arXiv:2311.11514*, 2023.

Youhe Jiang, Fangcheng Fu, Xiaozhe Yao, Taiyi Wang, Bin Cui, Ana Klimovic, and Eiko Yoneki. Thunderserve: High-performance and cost-efficient llm serving in cloud environments. *Proceedings of Machine Learning and Systems*, 7, 2025a.

Youhe Jiang, Ran Yan, and Binhang Yuan. Hexgen-2: Disaggregated generative inference of llms in heterogeneous environment. *arXiv preprint arXiv:2502.07903*, 2025b.

Yibo Jin, Tao Wang, Huimin Lin, Mingyang Song, Peiyang Li, Yipeng Ma, Yicheng Shan, Zhengfan Yuan, Cailong Li, Yajing Sun, Tiandeng Wu, Xing Chu, Ruizhi Huan, Li Ma, Xiao You, Wenting Zhou, Yunpeng Ye, Wen Liu, Xiangkun Xu, Yongsheng Zhang, Tiantian Dong, Jiawei Zhu, Zhe Wang, Xijian Ju, Jianxun Song, Haoliang Cheng, Xiaojing Li, Jiandong Ding, Hefei Guo, and Zhengyong Zhang. P/D-Serve: Serving disaggregated large language model at scale, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, pages 611–626, New York, NY, USA, 2023. Association for Computing Machinery. doi: 10.1145/3600006.3613165.

Joseph Y.-T. Leung. A new algorithm for scheduling periodic, real-time tasks. *Algorithmica*, 4:209–219, 1989. doi: 10.1007/BF01553887. URL https://doi.org/10.1007/BF01553887.

Hanchen Li, Qiuyang Mang, Runyuan He, Qizheng Zhang, Huanzhi Mao, Xiaokun Chen, Alvin Cheung, Joseph E. Gonzalez, and Ion Stoica. Continuum: Efficient and robust multi-turn LLM agent scheduling with KV cache time-to-live, 2025.

Luchang Li, Dongfang Li, Bozhao Gong, and Yu Zhang. SLO-aware compute resource allocation for prefill-decode disaggregated LLM inference, 2026.

Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 663–679, Boston, MA, July 2023. USENIX Association. ISBN 978-1-939133-34-2. URL https://www.usenix.org/conference/osdi23/presentation/li-zhouhan.

Junhan Liao, Minxian Xu, Wanyi Zheng, Yan Wang, Kejiang Ye, Rajkumar Buyya, and Chengzhong Xu. A dynamic PD-disaggregation architecture for maximizing goodput in LLM inference serving, 2025.

Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. Parrot: Efficient serving of LLM-based applications with semantic variable. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 929–945, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/lin-chaofan.

Yifei Liu, Zuo Gan, Zhenghao Gan, Weiye Wang, Chen Chen, Yizhou Shan, Xusheng Chen, Zhenhua Han, Yifei Zhu, Shixuan Sun, and Minyi Guo. Hermes: Efficient serving of LLM applications with probabilistic demand modeling. *ACM Transactions on Architecture and Code Optimization*, 2026. doi: 10.1145/3803390.

Michael Luo, Xiaoxiang Shi, Colin Cai, Tianjun Zhang, Justin Wong, Yichuan Wang, Chi Wang, Yanping Huang, Zhifeng Chen, Joseph E. Gonzalez, and Ion Stoica. Autellix: An efficient serving engine for LLM agents as general programs, 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36. Curran Associates, Inc., 2023.

Ziming Mao, Tian Xia, Zhanghao Wu, Wei-Lin Chiang, Tyler Griggs, Romil Bhardwaj, Zongheng Yang, Scott Shenker, and Ion Stoica. Skyserve: Serving ai models across regions and clouds with spot instances. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 159–175, 2025.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces, 2026.

Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging ai applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 561–577. USENIX Association, 2018.

Derek G. Murray, Malte Schwarzkopf, Christopher Smowton, Steven Smith, Anil Madhavapeddy, and Steven Hand. CIEL: A universal execution engine for distributed data-flow computing. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*. USENIX Association, 2011.

Derek G. Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martin Abadi. Naiad: A timely dataflow system. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles*, pages 439–455, 2013.

Kay Ousterhout, Patrick Wendell, Matei Zaharia, and Ion Stoica. Sparrow: Distributed, low latency scheduling. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles*, pages 69–84, 2013.

Zaifeng Pan, Qianxu Wang, Zhengding Hu, Chang Chen, Yue Guan, Yanbo Zhou, Steven Swanson, and Yufei Ding. Smoothagent: Efficient long-horizon llm-based agent serving with lookahead context engineering, 2026.

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

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://icml.cc/virtual/2025/poster/46593.

Archit Patke, Dhemath Reddy, Saurabh Jha, Haoran Qiu, Christian Pinto, Chandra Narayanaswami, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. Queue management for SLO-oriented large language model serving. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*, pages 18–35, New York, NY, USA, 2024. Association for Computing Machinery. doi: 10.1145/3698038.3698523.

Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew T. Kalbarczyk, Tamer Başar, and Ravishankar K. Iyer. Efficient interactive LLM serving with proxy model-based sequence length prediction. In *Proceedings of the 5th International Workshop on Cloud Intelligence / AIOps at ASPLOS 2024 (AIOps 2024)*, pages 1–7, San Diego, CA, USA, 2024. Association for Computing Machinery. URL https://haoran-qiu.com/publication/asplos-2024/.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36. Curran Associates, Inc., 2023.

Linus Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, 1968.

Rana Shahout, Eran Malach, Chunwei Liu, Weifan Jiang, Minlan Yu, and Michael Mitzenmacher. Don't stop me now: Embedding based scheduling for LLMs, 2024.

John A. Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio C. Buttazzo. *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*. Springer, Boston, MA, USA, 1998.

Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. Llumnix: Dynamic scheduling for large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 173–191, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/sun-biao.

Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems*, pages 1–17, 2015.

Jiahao Wang, Kaizhan Lin, Kaixi Zhang, Jinbo Han, Xingda Wei, Sijie Shen, Chenguang Fang, Wenyuan Yu, Rong Chen, and Haibo Chen. SMetric: Rethink LLM scheduling for serving agents with balanced session-centric scheduling, 2026.

Samuel Williams, Andrew Waterman, and David A. Patterson. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009. doi: 10.1145/1498765.1498785. URL https://doi.org/10.1145/1498765.1498785.

Bingyang Wu, Yinmin Zhong, Zili Zhang, Gang Huang, Xuanzhe Liu, and Xin Jin. Fast distributed inference serving for large language models, 2023.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380. Association for Computational Linguistics, 2018. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36. Curran Associates, Inc., 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=WE_vluYUL-X.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for transformer-based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-28-1. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Yilong Zhao, Shuo Yang, Kan Zhu, Lianmin Zheng, Baris Kasikci, Yang Zhou, Jiarong Xing, and Ion Stoica. Blendserve: Optimizing offline inference for auto-regressive large models with resource-aware batching, 2024.

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. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, Red Hook, NY, USA, 2024. Curran Associates, Inc. URL https://papers.nips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 193–210, Santa Clara, CA, July 2024. USENIX Association.

ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 62138–62160. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhou24r.html.

A Detailed Related-Work

LLM-call-level scheduling. FastServe (Wu et al., 2023), QLM (Patke et al., 2024), SSJF (Qiu et al., 2024), learned-to-rank scheduling (Fu et al., 2024b), and embedding-based scheduling (Shahout et al., 2024) improve call ordering, preemption, or size prediction. BlendServe (Zhao et al., 2024) additionally uses resource-aware batching and prefix sharing for offline inference. These policies address individual model invocations, whereas HexAGenT considers the completion of dependent LLM calls within dynamic workflows on heterogeneous P-D workers.

Heterogeneous serving and migration. HexGen (Jiang et al., 2023), HexGen-2 (Jiang et al., 2025b), and ThunderServe (Jiang et al., 2025a) optimize serving over heterogeneous resources and networks. SkyServe (Mao et al., 2025) addresses availability and cost across regions and clouds using spot instances. Llumnix (Sun et al., 2024) and ServerlessLLM (Fu et al., 2024a) support elasticity or migration.

Dynamic workflows and cluster state. CIEL (Murray et al., 2011), Ray (Moritz et al., 2018), and Timely Dataflow (Murray et al., 2013) support execution of dynamically revealed graphs. Borg (Verma et al., 2015) and Sparrow (Ousterhout et al., 2013) study cluster scheduling and the cost of maintaining scheduling state. HexAGenT's focus is workflow-level prioritization combined with heterogeneous P-D placement and asynchronous plan application, rather than dynamic revelation or centralized planning alone.

Agent workflow patterns. ReAct (Yao et al., 2023b) and Toolformer (Schick et al., 2023) connect model generation with tool use. Self-Refine (Madaan et al., 2023), Tree of Thoughts (Yao et al., 2023a), and Graph of Thoughts (Besta et al., 2024) explore iterative or branching reasoning. These application patterns motivate the dependency structures in Figure 1; they do not specify how to prioritize competing workflows on heterogeneous P-D workers.

Scheduling objectives. Classical slack-stealing analysis (Davis et al., 1993) studies how to serve additional work without violating hard timing constraints. In model serving, AlpaServe (Li et al., 2023) and DistServe (Zhong et al., 2024) evaluate service under latency constraints. Our formulation applies an attainment target to complete workflows rather than independent model calls.

B System Architecture and Detailed Lifecycle

Simulator and estimator. The standalone event-driven simulator models the lifecycle of LLM calls, worker queues, KV transfer, decode KV capacity, and tool-completion events. It advances the runtime state used by the scheduler. The hardware-aware resource estimator supplies prefill and decode service times, transfer latency, and KV-token demand; the planner uses these estimates to project the consequences of tentative assignments. The simulator, estimator, and planner therefore have distinct roles: advancing events, estimating execution costs, and choosing a schedule. Figure 2 shows how the corresponding modules exchange state.

Call lifecycle. An LLM call follows `WAITING_PREFILL` → `PREFILLING` → `TRANSFER` → `WAITING_DECODE` → `DECODING` → `COMPLETED`. A following tool phase may keep the workflow blocked after the LLM call completes. A multi-parent child is revealed only after all parent calls and required tool phases have completed. A tool phase consumes no P-D worker capacity.

Cache metadata. The implementation uses a cache-context identifier to locate reusable KV; this identifier is not interchangeable with the workflow identifier. Multiple LLM calls can share a context, and a workflow can contain calls with different contexts. After a call completes, the cache metadata

records its local prefill worker and global-copy availability. The scheduler evaluates the prefix available to the next call without controlling eviction or offloading. Its suffix-length prefill model is an approximation, not a claim that attention to cached tokens is free.

C Scheduling Procedure and Plan Application

Planning loop. Algorithm 1 uses runtime state $\mathcal{S}(t)$ at time t. Stage s is either prefill or decode, Q_s is its waiting-call set, $\widehat{\mathcal{S}}$ is the planner's copied state, and Π_s is the resulting list of assignment and queue-order actions. The implementation permits one outstanding computation per stage; unassigned calls can receive fallback assignments while it is pending.

Algorithm 1 Workflow-aware P-D planning in HexAGenT

Require: execution event e, time t, runtime state $\mathcal{S}(t)$
Update worker queues, KV capacity, and call/cache state using e
if e completes a tool phase **then**
    Record its observed duration and reveal eligible dependent LLM calls
end if
Update affected revealed workflow horizons $H_w(t)$
for all stages s receiving new waiting LLM calls **do**
    Give unassigned calls feasible fallback assignments
    if a planning computation for s is already pending **then**
        continue
    end if
    Copy $\widehat{\mathcal{S}} \leftarrow \mathcal{S}(t)$; $Q_s \leftarrow$ waiting calls; $\Pi_s \leftarrow \emptyset$
    Estimate completion costs and order Q_s using the stage's priority rule
    while $Q_s \neq \emptyset$ **do**
        Take the next call c and evaluate placements in $\widehat{\mathcal{S}}$
        if a feasible placement exists **then**
            Assign c to its best placement and add the action to Π_s
            Update projected worker availability, transfers, and KV capacity in $\widehat{\mathcal{S}}$
        end if
        Remove c from Q_s; calls not assigned remain waiting in the runtime
        if the queue uses recomputed priorities **then**
            Recompute priorities and reorder the remaining calls in Q_s
        end if
    end while
    Refine the order within a bounded search budget
    Return Π_s when its planning computation completes
end for

Ordering and bounded refinement. HexAGenT ranks larger projected normalized completion scores first. Pairwise refinement exchanges two positions and re-evaluates placement; it is disabled for fan-out cases in the supplied implementation. The objective first minimizes the maximum projected ratio across represented workflows, then the mean ratio, then total projected call latency. These are practical scheduling heuristics, not guarantees of optimality or fairness.

C.1 P-D Placement and Asynchronous Plan Application

Joint prefill–decode placement. A prefill decision records both the prefill worker and the decode destination, using the predicted completion of both stages rather than prefill speed alone. The destination must accommodate the call's KV demand; after KV transfer, the call may be reordered in that decode queue. A stage already in service is never reassigned.

Applying returned plans. The dispatcher validates returned actions against current call state. An action can apply while its LLM call is still waiting and the assignment remains feasible; actions for calls already dispatched or in service are skipped. Starting a newer computation does not invalidate an older result. Older plans can still schedule waiting calls. Once a newer plan returns, its feasible actions replace older assignments; an older result cannot overwrite that newer applied plan. Planning therefore proceeds without interrupting service or discarding useful actions solely because another computation has started.

Tool duration and the reference SLO. Recorded tool durations enter the scheduler-independent reference duration for every method. During execution, the scheduler includes a tool duration in its revealed workflow horizon only after completion. This prevents future tool information from entering call prioritization.

D NOTATION

Table 4 summarizes the scheduling and evaluation notation. Times are measured in seconds; token counts represent model-compatible KV tokens rather than bytes. A workflow completion duration and an absolute timestamp are distinct quantities.

Table 4: Notation used in the scheduling framework and its implementation.

Symbol	Type/unit	Meaning
Workflows, objective, and resources		
$w,\ c$	identifiers	Workflow and one of its LLM calls, respectively.
$\mathcal{W}$	set	Nonempty set of workflows in the attainment objective.
π	policy	A scheduling policy, not an individual returned plan.
$t,\ a_w$	seconds	Current timestamp and arrival timestamp of workflow w.
C_w^π	seconds	Workflow completion duration under π, measured from a_w.
S_w	seconds	Specified end-to-end workflow latency SLO; chosen as a standalone H100 duration for evaluation.
$\alpha,\ \alpha S_w$	factor; seconds	Dimensionless SLO scaling factor and the resulting scaled SLO.
τ	fraction	Target workflow attainment in $(0, 1]$.
α_τ^π	factor	Smallest scaling factor achieving attainment τ under policy π.
$\mathbb{1}[\cdot]$	indicator	One when its condition holds, and zero otherwise.
$\mathcal{P},\ \mathcal{D}$	sets	Available prefill and decode workers; p and d index these sets.
$s \in \{\mathrm{P}, \mathrm{D}\}$	stage	The prefill or decode waiting stage at which a call is scheduled.
Revealed workflow progress		
$G_w = (V_w, E_w)$	DAG	Workflow graph, with LLM calls as vertices and dependencies as edges.
$G_w(t)$	DAG	Currently revealed subgraph, including completed history; $V_w(t)$ and $E_w(t)$ are its vertices and edges.
$\mathcal{A}_w(t),\ A$	set; path	Directed paths in $G_w(t)$ and one such path.
$\ell(c)$	seconds	Estimated standalone prefill, transfer, and decode duration used to construct the revealed horizon, excluding queueing.
$u(c, t)$	seconds	Observed duration of the completed tool phase following c; zero before that phase completes.
$H_w(t)$	seconds	Longest revealed dependency-path duration. It is neither the user SLO nor a prediction of remaining workflow time.
$R_s(c, t)$	score	Projected call completion from workflow arrival, normalized by $H_w(t)$; a heuristic, not a violation probability.
KV reuse and service estimates		
$L_{\mathrm{in}}(c)$	tokens	Input-token count of LLM call c.
$\widehat{L}_{\mathrm{out}}(c)$	tokens	Estimated output-token count of call c.
h_c	tokens	Compatible reusable prefix, bounded by $L_{\mathrm{in}}(c)$.
n_c	tokens	Estimated suffix length $\max\{1, L_{\mathrm{in}}(c) - h_c\}$.
$p_{\mathrm{cache}}(c, t)$	worker	Prefill worker holding the reusable local prefix, when available.
$T_{\mathrm{P}}(n, p)$	seconds	Estimated prefill service for n tokens on worker p.
$T_{\mathrm{full}}(c, p)$	seconds	Full-prefill cost with no prefix reused.
$T_{\mathrm{local}}(c, p)$	seconds	Suffix-prefill cost when the compatible prefix is local.
$T_{\mathrm{global}}(c, p)$	seconds	Global-prefix fetch plus suffix-prefill cost.
τ_G	seconds	Global-prefix fetch startup time; distinct from attainment target τ.

Continued on the next page

Table 4 continued

Symbol	Type/unit	Meaning
B_G	KV tokens/s	Effective global-prefix transfer rate in the token-based model.
$\widetilde{T}_{\mathrm{P}}(c,p,t)$	seconds	Minimum feasible full, local-reuse, or global-reuse prefill cost.
$T_{\mathrm{PD}}(c,p,d)$	seconds	KV-transfer latency from prefill worker p to decode worker d.
$T_{\mathrm{D}}(c,d)$	seconds	Estimated decode service time on worker d.
$m(c)$	KV tokens	Decode KV demand, $L_{\mathrm{in}}(c) + \widehat{L}_{\mathrm{out}}(c)$.
Queueing and completion projections		
$Q_{\mathrm{P}}(c,p,t)$	seconds	Predicted wait from t until prefill starts on p.
$\widehat{Q}_{\mathrm{D}}(c,p,d,t)$	seconds	Predicted wait from KV arrival at d until decode starts for a call still awaiting prefill.
$Q_{\mathrm{D}}(c,d,t)$	seconds	Predicted wait from t until decode starts for a call already awaiting decode.
$\mathcal{M}_c(t)$	set	Feasible prefill/decode worker pairs for call c.
$\mathcal{D}_c(t)$	set	Feasible decode destinations; a transferred call retains its destination.
$\Delta_s(c,t)$	seconds	Best feasible projected time from t to decode completion, including queueing and the stages still required.
Implementation notation (Appendix C)		
e	event	Runtime event triggering a state update.
$\mathcal{S}(t),\ \widehat{\mathcal{S}}$	states	Runtime state and the planner's copied, tentatively updated state.
Q_s	set	Waiting LLM calls for stage s; not the time-valued queue estimates Q_{P} or Q_{D}.
Π_s	action list	Stage-s plan of worker assignments and queue-order actions; distinct from policy π.
Req95, Req99	factors	Nearest-rank 95th and 99th percentiles of per-workflow required SLO scaling factors in evaluation.