TAO: TOOL-AWARE JOINT SCHEDULING FOR LLM-BASED AGENTIC SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Agentic services alternate between model inference on clustered GPUs and external tools on distributed CPUs. In this scenario, existing systems (1) overlook tool-return readiness in service evaluation, wasting limited resources, and (2) use heuristic strategies to decide admission and placement on GPUs, resulting in suboptimal scheduling decisions. To bridge this gap, we present TAO, a tool-aware joint scheduling framework for LLM-based agentic serving. Its value interface combines near-term tool readiness, node-specific cache credit with time-aware decay, and token footprint; a dynamic program jointly selects and places waiting sessions under capacity constraints. Across the 12 reported model–workload–concurrency settings, TAO reduces latency by up to 26.8%, reduces mean time to first token (TTFT) by up to 36.3%, and increases decode throughput by up to 29.5% relative to the best evaluated state-of-the-art baseline for each metric. Our code is available at `https://anonymous.4open.science/r/TAO-330C`.

1 INTRODUCTION

Recent advances in language models have expanded their use beyond basic chatbots to complex agents, making CPU-based tool execution a substantial source of latency alongside GPU-based inference (Xiaomi; Huang et al., 2026). Specifically, an agent operates through an ***agent session*** that alternates between LLM inference and external tools, such as compilers and retrievers, to complete multi-step tasks (Anthropic, 2026a; OpenAI, 2026). When tools run outside the inference service, a session's demand for GPU computation becomes interruptible. We call this serving scenario **C**lustered-**G**PUs with **D**istributed-**C**PUs (CGDC), illustrated in Figure 1(Left): clustered GPUs, i.e., nodes, provide shared inference capacity, while external tool calls introduce delays between LLM requests[1]. Hence, efficient agentic serving must account for both inference demand and tool timing to sustain throughput and session responsiveness.

We find that existing serving systems generally follow an *evaluate-then-decide* scheduling pattern (Guy et al., 2025; SGLang Team, n.d.; vLLM Production Stack Team, n.d.). They evaluate node features at the session level and then decide where and when to serve each session. This separates the problem into two related challenges: (1) how to value a session's next model turn, and (2) how to select and place sessions under shared capacity constraints.

For *session evaluation*, existing systems in CGDC scenarios generally overlook tool-call latency when evaluating established workload features. Cache-aware systems reuse KV prefixes and balance inference load (Zheng et al., 2024; Srivatsa et al., 2025; Qin et al., 2025; NVIDIA, 2025); workflow-aware systems anticipate future calls (Wadlom et al., 2026; SmoothAgent Contributors, 2026); and latency-aware schedulers prioritize sessions using latency or remaining-work estimates (Choi et al., 2025; Chen et al., 2025). Tool timing provides another useful signal, yet timing is utilized only as a surrogate for KV reuse or token count: Li et al. (2025) predicts tool duration to choose KV-cache retention times, while Kang et al. (2026) discounts the token footprint of sessions executing tools as time elapses. Such features commonly enter fixed scoring rules or manually chosen thresholds.

For *decision making*, existing systems usually apply heuristic scheduling strategies to each session, leading to myopic and suboptimal placements. In llm-d and SGLang (Guy et al., 2025;

[1]One agent session consists of multiple LLM inference requests (i.e., steps) after tool calls.

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.

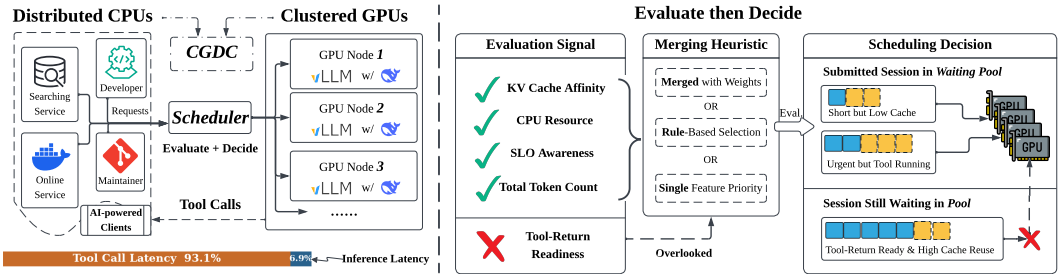

Figure 1: Left: In the CGDC serving scenario, external tool calls create intervals between LLM requests and opportunities to reallocate inference capacity. Right: Two scheduling challenges motivate TAO, including (1) overlooking tool-return readiness and using heuristic merging for evaluation, and (2) making myopic and suboptimal decisions about session admission and placement.

SGLang Team, n.d.), per-session scoring and routing rules assign inference opportunities *individually*. Yet sessions compete for the same node capacity. In particular, assigning one session can prevent a more valuable combination of other sessions from being served. Furthermore, the heuristic selection and placement rules can also lead to suboptimal choices (Peng et al., 2026; Yuan et al., 2026). As shown in Figure 1(Right), they fail to formulate scheduling jointly across multiple sessions.

To address these challenges, we propose TAO, a Tool-Aware jOint scheduling framework for LLM-based agentic serving. To incorporate *tool latency* into **session evaluation**, we propose tool-return readiness and explicitly integrate it with established features such as KV-cache reuse. Following Kang et al. (2026), we interrupt sessions during tool calls and release their GPU admission capacity. Interrupted sessions and new ones enter a *waiting pool*. Tool-return readiness estimates which sessions will need inference within the next scheduling window. For each session–node pair, we form a value triplet from readiness, time-decayed *KV-cache reuse*, and token footprint. Given these evaluation triplets, we design a *joint scheduling* algorithm in the **decision making** stage that considers all sessions in the waiting pool. Across deployments in different scenarios, joint optimization considers how the admission of each session affects the placements available to others. The optimizer maximizes the additive evaluation triplets using dynamic programming. Their coordinates evolve with cache age and tool progress, allowing distinct signals to guide decisions over time without a weighted sum. Separating the optimization procedure from session evaluation makes the framework extensible.

We summarize our contributions as follows:

- We identify the Clustered GPUs with Distributed CPUs (CGDC) serving scenario in the era of LLM-based agents, which shifts the key bottleneck of latency to tool calls.
- We propose TAO, a tool-aware joint scheduling framework for LLM-based agentic serving. It improves the evaluate-then-decide paradigm through session evaluation with explicit tool-return readiness and decision making with joint modeling and optimization.
- TAO can be generalized and extended to different workloads such as multi-agent orchestration and Looped Agents. Moreover, it is also compatible with non-agent chatbot scenarios.
- Across the 12 reported model–workload–concurrency settings, TAO reduces latency by up to 26.8%, reduces mean time to first token (TTFT) by up to 36.3%, and increases decode throughput by up to 29.5% relative to the state-of-the-art baseline for each metric.

2 Background and Problem Description

2.1 Agent Session and Distributed Resources

An agent session is a persistent interaction that alternates between model requests and external tool calls. During a model request, it uses a serving node; during tool execution, it awaits a result outside the inference service. A scheduler records each session's state and controls session admission, while the serving engine manages physical KV-cache residency and eviction.

A later model request can reuse the KV prefix produced by earlier turns and avoid prefill computation (Kwon et al., 2023). Because an agent session carries its instructions, history, and tool observations across turns, reuse depends on both the agent session history and the node selected for its next request. SGLang and Preble exploit related prefix structure in model programs and distributed scheduling (Zheng et al., 2024; Srivatsa et al., 2025).

Our CGDC separates clustered GPU inference from distributed tool execution. Client or remote tools introduce variable delays between model requests, while their CPU and RAM are managed *outside* the GPU admission budget (Anthropic, 2026a; OpenAI, 2026; Chang et al., 2026). Tool latency therefore determines when an agent session may need GPU resources for inference again.

2.2 SESSION INTERRUPTION AND SESSION RESTART

During tool execution, a session can release its GPU resources safely, although its cached prefix may subsequently be evicted. In contrast, preempting an active inference session requires recomputation. Repeated recomputation and reconstruction cause *KV-cache thrashing*, wasting capacity on state that must be rebuilt. This phenomenon motivates cache-aware engine and scheduling work (Kwon et al., 2023; Zheng et al., 2024; Srivatsa et al., 2025; NVIDIA, 2025), as well as policies that weigh retaining state against serving other work (Li et al., 2025; Abhyankar et al., 2024).

Therefore, at a tool boundary, the scheduler can apply *session interruption*. Specifically, a scheduler can interrupt the agent session's model admission and release its node resources while the external tool continues. When inference service is needed again, *session restart* restores admission on an eligible node and resumes the existing interaction. session interruption and session restart describe session state transitions; they do not move, preserve, or directly evict physical KV blocks.

Furthermore, the decisions made for these two operations can be based on the same session evaluation. Thus, we optimize toward a unified goal. Session interruption can be decided by ordering sessions according to their values. Interrupted sessions join new arrivals in a waiting pool. We focus on *unified session evaluation* and *restarting decisions* from this pool.

3 METHOD

3.1 FRAMEWORK OVERVIEW

We present TAO, which combines agentic session evaluation with decision making, i.e. dynamic programming for joint selection and placement. The detailed formulation appears in Section 3.2.

Figure 2 illustrates the pipeline of our method. First, session evaluation maps agent session and node observations to assignment values (Section 3.2), including KV cache with time-aware decay (Section 3.3) and tool-return readiness (Section 3.4). Second, an exact dynamic program jointly selects waiting agent sessions and assigns them to nodes under residual capacity constraints (Sections 3.2 and 3.5). Separating session evaluation from decision making lets the evaluation interface evolve while preserving the scheduling procedure.

3.2 FORMULATION AND AGENTIC SESSION EVALUATION

At each restart scheduling decision, let the waiting pool P contain the agent sessions awaiting inference admission, including interrupted sessions and new arrivals. Let B be the nodes in the frozen snapshot. Each agent session i has admission cost w_i, and each node b has residual capacity C_b. Let $E \subseteq P \times B$ denote eligible assignments, and let $x_{i,b} \in \{0, 1\}$ indicate whether session i is admitted to node b. The feasible set is

$$\mathcal{X} = \left\{ x : \sum_{b \in B} x_{i,b} \leq 1, \quad \sum_{i \in P} w_i x_{i,b} \leq C_b, \quad x_{i,b} = 0 \text{ if } (i,b) \notin E \right\}. \tag{1}$$

TAO assigns each agent session–node pair the lexicographic value

$$\boldsymbol{v}_{i,b} = (k_{i,b}, r_i, t_i), \tag{2}$$

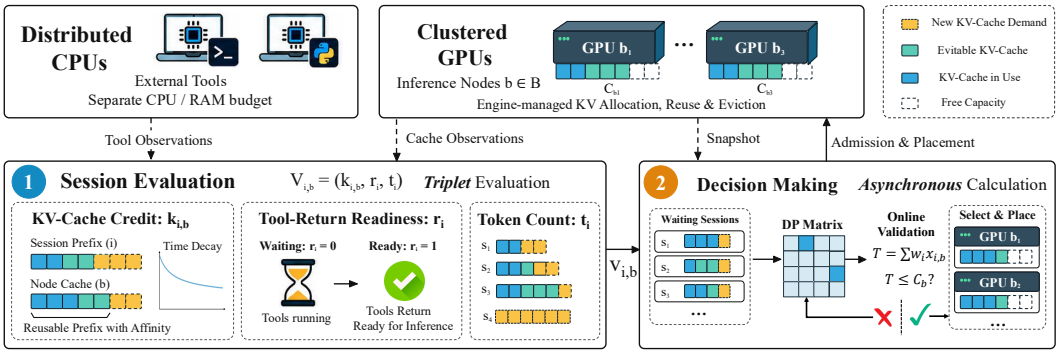

Figure 2: The TAO evaluate-then-decide framework in CGDC. External tools on distributed CPUs provide historical tool-call durations, while the serving engines on clustered GPUs provide cache observations. Then, the session evaluation component constructs triplets (decayed KV-cache credit, binary readiness, token count) for lexicographic comparison. The decision making component jointly selects and places waiting agent sessions; the executor revalidates the plan against live state before applying it. The resulting assignments schedule sessions to eligible nodes.

where $k_{i,b}$ is decayed cache credit, r_i is the binary readiness signal, and t_i is token count. We jointly select and place sessions to maximize their aggregate value:

$$x^\star \in \arg\operatorname*{lexmax}_{x\in\mathcal{X}} \sum_{i\in P}\sum_{b\in B} x_{i,b}\boldsymbol{v}_{i,b}. \tag{3}$$

The evaluation triplet's order remains fixed while its values evolve over time. Our design rationale is detailed as follows:

- *KV-Cache credit* takes precedence, but its value decays quickly after interruption, because prefix reuse directly avoids repeated prefill, motivating cache-aware scheduling (NVIDIA, 2025). The age-based discount in Section 3.3 favors recent cache evidence, which is often most useful when a tool returns quickly.
- As unrefreshed credit decays and the first component is equalized, *tool-return readiness* can distinguish agent sessions nearing a return from those still far from their next model turn (Section 3.4).
- *Token footprint* then favors larger accumulated contexts among plans tied on cache credit and readiness. The optimizer accounts for the greater capacity cost of such contexts when choosing the complete set of assignments.

3.3 Time-aware cache value for Clustered GPUs

TAO estimates reusable KV state from observations associated with each target node. It uses an observed cached-token count when available, or estimates the count from observed token volume and cache rate. A node without matching evidence receives zero credit. Figure 3 reports observed cache-token retention in DeepSeek-V4 Flash traces from two nodes with 4×H100 per node; the decline in binned medians motivates discounting older observations. Points show binned medians with bootstrap 95% intervals; the curve is a descriptive exponential fit over the populated bins through 1,800 s. Older observations receive less credit:

$$k_{i,b}(t) = k^0_{i,b}\, 2^{-\tau_{i,b}/h},$$
$$\tau_{i,b} = \max(0, t - t^{\text{obs}}_{i,b}). \tag{4}$$

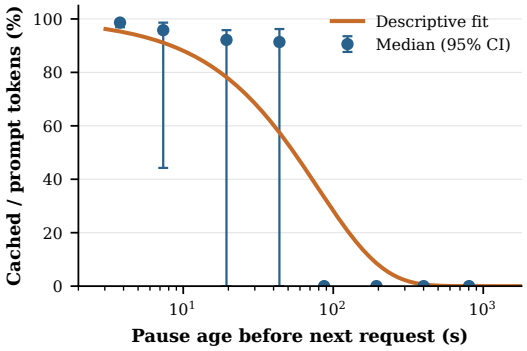

Figure 3: Observed cache-token retention in DeepSeek-V4 Flash traces from two nodes with 4×H100 per node.

Here $k^0_{i,b}$ is the observed cache credit and h is its decay half-life. We use $h = 10$ seconds and round the resulting credit for valuation. The discount makes the admission value responsive to observation age; the serving engine continues to control physical allocation, reuse, and eviction. The half-life is a scheduler parameter, distinct from the fitted trace time constant. Dynamo similarly adjusts overlap value under excess prefill load, and ThunderAgent discounts token demand over time (NVIDIA, 2025; Kang et al., 2026).

3.4 Tool-return readiness in a distributed CPU

In our CGDC deployment, external executors manage tool CPU and RAM, while TAO controls GPU admission. Tool-return readiness connects these resource domains by identifying sessions likely to request model service within a short horizon. Tool timing reflects external execution conditions; the GPU admission budget accounts for the returning session's inference footprint. Tool interfaces can expose progress or return deadlines: Claude Code's Monitor forwards background-work events, while OpenCode reports termination after a Bash timeout (Anthropic, 2026b; OpenCode Contributors, 2026).

The predictor combines these hints, parsed sleep commands, and historical tool durations, using direct timing evidence first and falling back to the tool-specific history, global history, or a five-second default (Li et al., 2025). For a call with elapsed time $e_i(t)$, the historical estimate $\widehat{D}_f$ gives

$$\widehat{d}_i(t) = \max\{0, \widehat{D}_f - e_i(t)\}. \tag{5}$$

The implementation adds a finite-sample margin to the historical mean; Appendix A specifies this estimator in detail.

Binary readiness. Let $o_i(t)$ indicate observed tool completion, $a_i(t)$ indicate an established tool execution candidate with an eligible prediction source, and $g_i(t)$ indicate valid prediction timestamps refreshed within 30 seconds. Write $z_i(t)$ for the earlier available predicted completion or exposed return deadline, treating an unavailable deadline as $+\infty$. With horizon H and overdue-probe indicator $b_i(t)$, we define

$$r_i(t) = \begin{cases} 1, & o_i(t) = 1, \\ a_i(t)b_i(t), & o_i(t) = 0,\ z_i(t) \le t, \\ a_i(t)g_i(t)\mathbf{1}[z_i(t) \le t + H], & \text{otherwise.} \end{cases} \tag{6}$$

TAO uses $H = 1$ second; the scheduler's session interruption valuation uses its configured scheduling interval. Overdue predictions receive temporary probe opportunities instead of remaining ready indefinitely. Observed completion takes precedence over prediction checks. This signal prioritizes near-term returns among plans tied on higher-precedence values.

3.5 Joint scheduling with dynamic programming

The optimizer is a capacity-state dynamic program for the multiple-knapsack and generalized-assignment structure (Kellerer et al., 2004; Shmoys & Tardos, 1993). Let Δ be the capacity quantum, $\bar{w}_i = \lceil w_i/\Delta \rceil$, and $\bar{C}_b = \lfloor C_b/\Delta \rfloor$. For candidate j, let $D_j(\boldsymbol{s})$ be the best objective attainable from the first j candidates at the used capacity $\boldsymbol{s}$, and let $\boldsymbol{u}_{j,b}$ embed the value and count for assigning j to node b:

$$D_j(\boldsymbol{s}) = \operatorname{lexmax}\left\{ D_{j-1}(\boldsymbol{s}),\ \operatorname*{lex\,max}_{b \text{ eligible}} \left[D_{j-1}(\boldsymbol{s} - \bar{w}_j \boldsymbol{e}_b) + \boldsymbol{u}_{j,b} \right] \right\}. \tag{7}$$

Only nonnegative states within $\bar{\boldsymbol{C}}$ are allowed; $D_0(\mathbf{0}) = \mathbf{0}$ and other initial states are unreachable. Skipping or assigning each candidate exhausts the feasible choices, so induction gives exactness for the frozen, discretized problem; rounding can exclude plans feasible before discretization.

Node-specific cache observations are decayed and folded into assignment values during $O(nm)$ preprocessing; cache-equivalent nodes can be grouped into $g \le m$ aggregate capacity dimensions without moving physical KV blocks. The planner runs in an isolated asynchronous worker (detailed in Appendix B) and revalidates before application; recorded preparation plus completed-solve work is below 0.12% of runtime in the four-replica sweep (Appendix I).

Table 1: Main results for DeepSeek-V4 Flash on 2 nodes with 4×H100 per node and Qwen3-32B on 4 nodes with 2×H100 per node. DeepSeek uses Terminal-Bench to study more demanding tasks at moderate concurrency; Qwen3-32B uses SWE-verified at higher concurrency to exercise the capacity of its smaller model replicas. **Bold** and underline mark the best and second-best distinct values in each concurrency-level/metric column.

DeepSeek-V4 Flash, 2 nodes with 4×H100 per node

		16 workers			32 workers			64 workers		
Benchmark	Method	Latency ↓	TTFT ↓	Decode ↑	Latency ↓	TTFT ↓	Decode ↑	Latency ↓	TTFT ↓	Decode ↑
SWE-Lite	Dynamo	121.993	0.146	874.085	152.297	0.176	1393.267	269.801	0.351	1548.472
	vLLM Scheduler w/ RR	122.814	0.177	888.489	158.841	0.217	1145.560	290.686	0.508	1299.754
	vLLM Scheduler w/ CA	116.011	0.183	889.590	150.431	0.226	1256.064	253.228	0.410	1561.307
	ThunderAgent	123.564	0.379	796.215	215.727	2.708	961.791	407.180	7.447	969.018
	TAO (Ours)	**113.264**	**0.135**	**934.967**	**142.332**	**0.153**	**1477.463**	**185.410**	**0.224**	**2021.523**
	Rel. Imprv. (%)	+2.4	+7.4	+5.1	+5.4	+13.3	+6.0	+26.8	+36.3	+29.5
Terminal-Bench	Dynamo	536.600	1.371	494.866	579.650	1.252	651.151	688.531	1.315	760.736
	vLLM Scheduler w/ RR	342.464	0.306	820.754	375.182	0.499	1071.063	467.410	0.582	1597.497
	vLLM Scheduler w/ CA	304.802	0.319	723.889	394.677	0.331	1146.434	393.065	0.387	**1654.108**
	ThunderAgent	361.061	0.336	**876.138**	380.108	1.402	1053.042	391.580	1.010	1376.539
	TAO (Ours)	**298.224**	**0.239**	798.624	**333.587**	**0.264**	**1161.438**	**376.647**	**0.324**	1551.387
	Rel. Imprv. (%)	+2.2	+21.7	-8.8	+11.1	+20.3	+1.3	+3.8	+16.2	-6.2

Qwen3-32B, 4 nodes with 2×H100 per node

		96 workers			128 workers			160 workers		
Benchmark	Method	Latency ↓	TTFT ↓	Decode ↑	Latency ↓	TTFT ↓	Decode ↑	Latency ↓	TTFT ↓	Decode ↑
SWE-Lite	Dynamo	821.558	2.098	2074.598	857.418	7.924	1203.708	1203.712	17.208	1239.372
	vLLM Scheduler w/ RR	775.303	5.109	1400.577	975.382	11.179	1359.789	1088.036	16.645	1433.349
	vLLM Scheduler w/ CA	811.604	5.339	1614.671	881.068	11.707	1456.411	1179.340	18.304	1285.413
	ThunderAgent	716.272	2.051	**2104.896**	839.274	8.705	2036.280	**1028.478**	16.209	1712.363
	TAO (Ours)	**634.694**	**1.827**	1996.793	**829.361**	**7.907**	**2051.807**	1030.973	**14.554**	**1827.561**
	Rel. Imprv. (%)	+11.4	+10.9	-5.1	+1.2	+0.2	+0.8	-0.2	+10.2	+6.7
SWE-verified	Dynamo	736.823	1.985	1862.845	1041.911	10.441	1820.919	1286.167	22.469	1667.794
	vLLM Scheduler w/ RR	777.584	5.082	1931.333	933.347	12.586	1804.505	1244.525	24.221	1091.002
	vLLM Scheduler w/ CA	816.201	5.731	1522.641	1016.513	14.263	1670.856	1264.406	23.199	1619.768
	ThunderAgent	699.801	2.396	2184.933	**911.097**	10.500	2045.875	1202.215	21.608	1993.620
	TAO (Ours)	**642.017**	**1.392**	**2354.709**	933.678	**9.092**	**2128.599**	**1094.318**	**17.619**	**2087.191**
	Rel. Imprv. (%)	+8.3	+29.9	+7.8	-2.5	+12.9	+4.0	+9.0	+18.5	+4.7

4 EXPERIMENTS

4.1 EXPERIMENT SETUP

Models and workloads. We evaluate DeepSeek-V4 Flash, Qwen3-32B, and GLM-4.7 (DeepSeek-AI et al., 2026; Yang et al., 2025; Team et al., 2025) on the SWE-bench Lite workload (SWE-Lite) using the mini-SWE-agent harness (MiniSWE) (mini-SWE-agent Contributors, 2026; Jimenez et al., 2024). For DeepSeek, we also use Claude Code on Terminal-Bench 2.0 (Terminal-Bench Contributors, 2026) for challenging tasks. For Qwen, we test with SWE-verified to exercise the serving capacity of its smaller model replicas. We further evaluate GLM-4.7 on SWE-Lite to test single-node performance. Appendix E reports an InferenceX trace-replay evaluation.

Deployment. DeepSeek runs on 2 nodes with 4×H100 per node; Qwen runs on 4 nodes with 2×H100 per node. Each node hosts one tensor-parallel model replica. GLM-4.7 runs on one node with 8×H100. Appendix D reports a lower-concurrency DeepSeek check on the same single-node topology.

Baselines. vLLM Scheduler w/ RR and vLLM Scheduler w/ CA denote the vLLM schedulers with round-robin and cache-aware policies, respectively (vLLM Production Stack Team, n.d.), which align with the policy in SGLang (Zheng et al., 2024). These baselines cover load spreading and prefix-locality scheduling. Dynamo is a KV-aware scheduler (NVIDIA, 2025). ThunderAgent

Table 2: High-concurrency extensibility showcase for a larger model: GLM-4.7 on SWE-Lite through MiniSWE, one node with 8×H100 (TP8), and 300 tasks per run.

GLM-4.7, one node with 8×H100 (TP8)									
		96 workers			128 workers			160 workers		
Benchmark	Method	Latency ↓	TTFT ↓	Decode ↑	Latency ↓	TTFT ↓	Decode ↑	Latency ↓	TTFT ↓	Decode ↑
SWE-Lite	Dynamo	2870.313	21.677	290.532	3779.196	41.865	287.844	4786.720	59.548	262.215
	vLLM Scheduler w/ RR	2687.766	20.927	310.800	3633.795	39.502	294.582	4390.961	53.588	272.460
	vLLM Scheduler w/ CA	2606.523	19.553	318.687	3435.779	37.665	307.983	4314.458	56.326	297.280
	ThunderAgent	999.624	7.014	773.250	1360.382	14.243	731.052	1747.028	23.345	668.940
	TAO (Ours)	**887.663**	**6.777**	**869.628**	**1261.906**	**14.103**	**813.653**	**1476.190**	**20.066**	**805.939**
	Rel. Imprv. (%)	+11.2	+3.4	+12.5	+7.2	+1.0	+11.3	+15.5	+14.1	+20.5

uses token-count scheduling and best-fit-decreasing placement (Kang et al., 2026). TAO combines predictive cache-aware session evaluation with joint dynamic-programming placement.

Metrics. We report mean task end-to-end latency, mean time to first token (TTFT), and decode throughput in tokens per second (TPS). Task latency includes both model inference and tool execution. Component analyses also measure prefix-cache hit rate, tool-end-to-first-token delay, and pause wait, which includes the tool call latency of an interrupted session.

4.2 Main Results

Table 1 compares five baselines across different levels of concurrency, models, and topologies. Taken together, these results make TAO the strongest overall choice: it combines consistently low latency and TTFT with high decode throughput more effectively than any single baseline, consistent with our focus on responsiveness after tool calls and improved KV reuse.

DeepSeek-V4 Flash. We evaluate on challenging benchmarks with moderate concurrency. On SWE-Lite, TAO achieves the lowest latency and TTFT and the highest decode throughput at every displayed worker count. Relative to the best competing method in each column, latency falls by 2.4–26.8%, TTFT by 7.4–36.3%, and decode throughput rises by 5.1–29.5%. On Terminal-Bench, TAO also leads on latency and TTFT at all three concurrency levels. At 32 workers, it improves latency, TTFT, and decode throughput by 11.1%, 20.3%, and 1.3%, respectively. At 16 and 64 workers, decode throughput is 8.8% and 6.2% below the best competing method.

Qwen3-32B. We use higher concurrency to exercise the serving capacity of its smaller model replicas. On SWE-Lite, TAO achieves the lowest TTFT at all three worker counts and the highest decode throughput at 128 and 160 workers. At 96 workers, it reduces latency by 11.4% and TTFT by 10.9%, with 5.1% lower decode throughput than ThunderAgent. On SWE-verified, TAO leads on TTFT and decode throughput throughout, with reductions of 12.9–29.9% in TTFT and gains of 4.0–7.8% in decode throughput. It also reduces latency by 8.3% and 9.0% at 96 and 160 workers; at 128 workers, latency is 2.5% higher than ThunderAgent.

GLM-4.7. As a larger-model setting, GLM-4.7 provides a high-concurrency extensibility showcase on one node with 8×H100. Table 2 reports the separate worker-grouped result: at 160 workers, improvements reach 15.5% in latency, 14.05% in TTFT, and 20.48% in decode throughput. Appendix D reports the complementary low-concurrency DeepSeek fallback check on the same topology.

User and system tradeoff. The worker sweep exposes a latency–throughput tradeoff: the lowest latency and TTFT occur at the smallest worker count, while peak decode throughput occurs at 64 workers for DeepSeek SWE-Lite and Terminal-Bench, 128 workers for Qwen SWE-Lite, and 96 workers for SWE-verified. We use DeepSeek at 32 workers as a balanced operating point and audit deferred sessions and long-tail responsiveness for this TP4×2 deployment in Appendix H. More analysis follows.

4.3 Improvement Breakdown

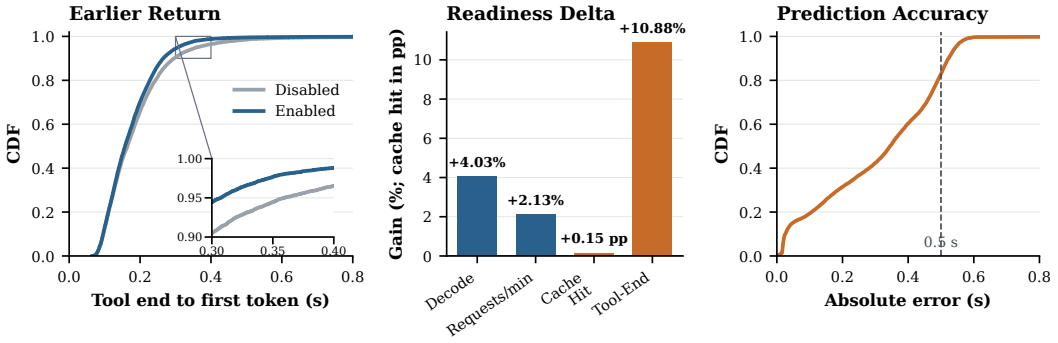

Figure 5: Readiness effectiveness in the matched 32-worker SWE-Lite traces. From left to right, the panels show the tool-end-to-first-token CDF, the aggregate readiness delta, and absolute error of the structured-history duration predictions. Tool-end measures the time from tool completion to the first output token. The left inset magnifies 0.3–0.4 s. The dashed line in the right panel marks 0.5 s, half of the 1 s scheduling period.

We examine how cache-aware valuation, tool readiness, and joint placement improve reuse, tool-return responsiveness, and admission decisions using the 32-worker DeepSeek deployment on 2 nodes with $4\times$H100 per node.

Cache-aware valuation. In a fixed-workload replay of 218 tasks with DP placement held fixed for the ThunderAgent control and cache-aware variants, cache-token ordering raises prefix-cache hit rate by 0.19 percentage points and reduces mean pause wait by 4.5% relative to ThunderAgent. Adding cache-credit decay raises hit rate by 0.31 percentage points and decode throughput by 1.68% relative to ThunderAgent. Figure 4 summarizes these changes; bars are percentage points for cache hit and percentages for the other metrics, with positive values denoting improvement.

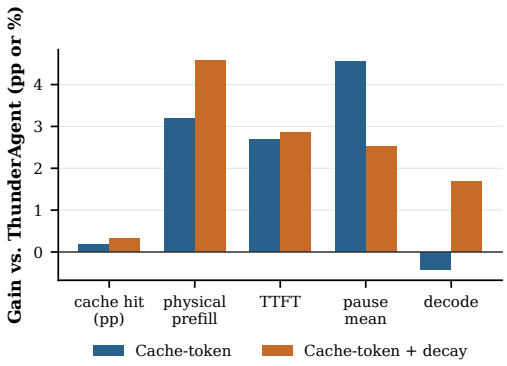

Figure 4: Cache-aware valuation in the 32-worker replay, relative to the ThunderAgent control with DP placement. Reductions in prefill, TTFT, and pause wait are shown as positive improvements.

Tool readiness. Enabling readiness in the paired SWE-Lite runs increases decode throughput by 4.03% and reduces tool-end-to-first-token delay by 10.88%. Mean pause wait falls from 1.067 to 0.972 s. Figure 5 shows the return-delay distributions and prediction error. Across 10,967 completed tools, the final duration prediction has a mean absolute error of 0.410 s, with 82.9% of errors within 0.5 s. The return-delay CDFs show that readiness shifts model responses earlier after tool completion. The middle panel summarizes the serving gains, and the right panel shows the prediction accuracy that informs readiness valuation. The return CDFs contain 10,481 and 10,673 observations, and the prediction panel contains 10,967 predictions.

Joint Scheduling. Joint optimization improves the lexicographic scheduling objective in six of the 23 replay snapshots with multiple candidates (26.1%), matching the short-first heuristic in the remaining 17. Across all 35 reconstructable snapshots, including single-candidate cases, DP improves six (17.1%) and matches the heuristic in 29 (82.9%). Detailed cache-credit and admission examples appear in Appendix G.

4.4 Component Ablations

Table 3 compares the complete TAO configuration with variants of its value function, pause ordering, and resumption planner. The complete configuration achieves the highest decode throughput and

Table 3: Component variants on SWE-Lite with DeepSeek-V4 Flash, 2 nodes with 4×H100 per node, and 32 workers. Each row uses the indicated component variant of TAO. Decode is in tok/s and hit is prefix-cache reuse (%). Mean and P90 timing values are reported separately in seconds. The best results are in **bold** and the second-best results are underlined.

Configuration	Decode ↑	Hit (%) ↑	TTFT ↓		Tool-end→first ↓		Pause ↓	
			Mean	P90	Mean	P90	Mean	P90
TAO (Ours)	**1417.97**	**96.31**	**0.154**	**0.241**	**0.203**	**0.299**	**0.972**	1.051
w/o Cache Value	1398.35	96.24	0.157	0.246	0.209	0.309	1.028	**1.047**
w/o Readiness	1363.06	96.16	0.161	0.253	0.227	0.358	1.067	1.079
w/o Cache Value & Readiness	1373.51	96.24	0.182	0.278	0.213	0.317	1.025	1.059
w/o Session Evaluation	1355.20	96.20	0.162	0.257	0.226	0.350	1.025	1.061
w/o Decision Making	1233.72	96.05	0.195	0.300	0.235	0.354	1.081	1.119

cache-hit rate, together with the lowest mean TTFT, post-tool delay, and pause wait. Removing cache credit lowers decode throughput from 1417.97 to 1398.35 tok/s; removing readiness lowers it to 1363.06 tok/s. Using the pause ordering from ThunderAgent yields 1355.20 tok/s, while replacing DP resumption with best-fit-decreasing (BFD) yields 1233.72 tok/s. Both valuation and decision making contribute to the observed performance.

5 Related Work

Related systems and our position. PagedAttention and RadixAttention improve engine-level KV management, while Preble, Mooncake, and Dynamo exploit distributed cache locality (Kwon et al., 2023; Zheng et al., 2024; Srivatsa et al., 2025; Qin et al., 2025; NVIDIA, 2025). SMetric combines session affinity with load balancing (Wang et al., 2026a), and production schedulers support extensible scoring (Guy et al., 2025). Autellix prioritizes programs using service history; ThunderAgent introduces the lifecycle and waiting pool on which TAO builds (Luo et al., 2025; Kang et al., 2026). InferCept and Continuum manage KV retention across tool calls, with Continuum using historical tool-duration distributions (Abhyankar et al., 2024; Li et al., 2025). TAO brings cache opportunity and tool readiness into a joint scheduler-level admission and placement objective, complementing engine-level KV management. Parrot and Helium exploit workflow dependencies, while KVFlow and CacheScout guide residency or prefetching using execution structure (Lin et al., 2024; Wadlom et al., 2026; Pan et al., 2025; Zhang et al., 2026). TOPAS jointly considers prefix residency and admission through bounded search and greedy packing (Ni et al., 2026). Conveyor, Sutradhara, and PASTE reduce exposed tool latency through partial execution, overlap, or speculation (Xu et al., 2024; Biswas et al., 2026; Sui et al., 2026). TAO converts tool timing into near-term model-service opportunities that inform joint admission and placement. AgentSysBench and MARS motivate accounting for tool delays and deployment boundaries (Chang et al., 2026; Wang et al., 2026b). TAO couples an agentic value interface with joint placement and asynchronous revalidation, using classical multiple-knapsack and generalized-assignment machinery (Kellerer et al., 2004; Shmoys & Tardos, 1993). This formulation optimizes competing session–node assignments under shared capacity; Section 3.5 and Appendix B.2 establish its snapshot optimality and execution guarantees. Appendix J provides the extended comparison.

6 Conclusion

Tool execution makes agentic inference demand intermittent, while accumulated context makes the choice of node consequential. TAO brings these two properties into a common scheduling objective through cache credit, tool-return readiness, and token footprint. Joint dynamic-programming placement maximizes this objective under snapshot capacity constraints, and asynchronous revalidation connects the plan to live execution. Across the evaluated DeepSeek-V4 Flash and Qwen3-32B settings, this design delivers improvements of up to 26.8% in latency, 36.3% in mean TTFT, and 29.5% in decode throughput. The resulting architecture gives agentic serving an extensible way to translate workload knowledge into coordinated admission and placement.

AI USE STATEMENT

We used generative AI tools to assist with manuscript editing and wording review. We did not use generative AI to run experiments, fabricate measurements, or decide which experimental artifacts are valid. We reviewed all AI-assisted text and figure suggestions against the source code, experiment records, cited literature, and the compiled manuscript. The authors take responsibility for the final content, claims, and artifacts in this submission.

REFERENCES

Reyna Abhyankar, Zijian He, Vikranth Srivatsa, Hao Zhang, and Yiying Zhang. InferCept: Efficient intercept support for augmented large language model inference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 81–95. PMLR, 2024. URL https://proceedings.mlr.press/v235/abhyankar24a.html.

Anthropic. How tool use works. https://platform.claude.com/docs/en/agents-and-tools/tool-use/how-tool-use-works, 2026a. Official documentation; accessed September 11, 2026.

Anthropic. Claude Code tools reference: Monitor tool. https://code.claude.com/docs/en/tools-reference, 2026b. Accessed September 8, 2026.

Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

Anish Biswas, Kanishk Goel, Srivarshinee S, Jayashree Mohan, Alind Khare, Anjaly Parayil, Ramachandran Ramjee, and Chetan Bansal. Sutradhara: An intelligent orchestrator-engine co-design for tool-based agentic inference. *arXiv preprint arXiv:2601.12967*, 2026. URL https://arxiv.org/abs/2601.12967v3.

Chaokun Chang, Yukun Zhou, Kaihua Fu, Dakai An, Tianyu Feng, Hanfeng Lu, Sheng Yao, Pu Guo, Yinghao Yu, Yizhou Shan, Bo Li, Binhang Yuan, and Wei Wang. From llm inference to agentic workloads: Characterization and implications for serving systems. *arXiv preprint arXiv:2608.15127*, 2026. URL https://arxiv.org/abs/2608.15127v1.

Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2005. doi: 10.1137/S0097539700382820.

Jinyuan Chen, Jiuchen Shi, Quan Chen, and Minyi Guo. Kairos: Low-latency multi-agent serving with shared LLMs and excessive loads in the public cloud. *arXiv preprint arXiv:2508.06948*, 2025. URL https://arxiv.org/abs/2508.06948.

Seungbeom Choi, Jeonghoe Goo, Eunjoo Jeon, Mingyu Yang, and Minsung Jang. ELIS: Efficient LLM iterative scheduling system with response length predictor. *arXiv preprint arXiv:2505.09142*, 2025. URL https://arxiv.org/abs/2505.09142.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang,

Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, SH Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Bochao Feng, Jianjiang Li, Haojie Wang, Lin Qiao, Yinghui Li, Yukun Yan, and Jidong Zhai. Decoupling readiness from release for tail-aware scheduling of agentic LLM workflows. *arXiv preprint arXiv:2609.10964*, 2026. URL https://arxiv.org/abs/2609.10964v1. Version 1, September 10, 2026.

In Gim, Zhiyao Ma, Seung-seob Lee, and Lin Zhong. Pie: A programmable serving system for emerging LLM applications. In *Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles*, 2025. doi: 10.1145/3731569.3764814. URL https://arxiv.org/abs/2510.24051v1.

Dongxin Guo, Jikun Wu, and Siu Ming Yiu. SAGA: Workflow-atomic scheduling for AI agent inference on GPU clusters. *arXiv preprint arXiv:2605.00528*, 2026. URL https://arxiv.org/abs/2605.00528v2. Version 2, June 19, 2026.

Nili Guy, Vita Bortnikov, Etai Lev Ran, Robert Shaw, and Clayton Coleman. Intelligent inference scheduling with llm-d. https://llm-d.ai/blog/intelligent-inference-scheduling-with-llm-d, 2025. Published September 3, 2025.

Jialiang Huang, Hongxuan Tang, Jingchang Chen, Yuxuan Liu, Yixiao Chen, Yuan Cheng, Yi Tao, Jingli Zhou, Yupeng Chen, Haoyu Chen, Jiarui Wang, Shengkai Lin, Chuqi Zhang, Bryan Lee Teng, Lian Guo, Zhe Fu, Wenjun Gao, Yisong Wang, Liang Zhao, Zehao Wang, Ziwei Xie, Yongqiang Guo, Peixin Cong, Ziyi Gao, Shuiping Yu, Hanwei Xu, Zuofan Wu, Zhizhou Ren, Yuyang Zhou, Bowei Zhang, Zhihuan Huang, Qihao Zhu, Lei Wang, Tianle Lin, Han Yu, Jiewen Hu, Dejian Yang, Shuo Yang, Shanghao Lu, Shaoyuan Chen, Junjie Qiu, Zhangli Sha, Yinmin Zhong, Yongtong Wu, Shiyu Wang, Wei Liu, Bingzheng Xu, Longhao Chen, Qiushi Du, Yuzhen Huang, Shirong Ma, Yaohui Wang, Mingshu Chen, Tongrui Xiong, Y. C. Yan, Haowen Luo, Haofen Liang, Xiaokang Zhang, Weihao Zeng, Runxin Xu, Peiyi Wang, Jinhua Zhu, Ruoyu Zhang, Wenkai Yang, Qi Tang,

Jiping Yu, Tian Ye, Ruizhe Pan, Honghui Ding, Xiaodong Liu, Lingxiao Luo, Zhihong Shao, Yuhan Wu, Jibai Lu, Wen Liu, Haoling Zhang, Jingcheng Hu, Yaoyang Ye, Chaofan Lin, Zhaochen Zhang, Jianan Tong, Hengxu Wu, Zhihao Li, Yicheng Wang, Luyao Wang, Yuzhuo Bai, Lingyue Fu, Ruifan Xu, Y. Z. Wang, Zonglin Li, Mingqi Wei, Haiyang Shen, Chengyuan Zhang, Chao Jin, Zili Zhang, R. H. Yang, Xinbo Xu, Jian Zhou, Ruidong Zhu, Yuzhe Guo, Zelun Pan, Shaoheng Nie, Erhang Li, Shuhan Lin, Zheng Liu, Anshuo Chen, Zilong Lyu, Sinuo Cao, Rui Yu, Chuhao Wang, Junyi Guo, Junxiao Song, Kaifeng Chen, Menghao Ye, Junxian Li, Di Wu, Haiyang Ma, Yilun Wang, Haoran Yang, Yizai Cai, Shichun Liu, Yiping Wang, Junbo Sun, Shicheng Xu, Xiao Bi, Ying He, Yichao Zhang, Mingxing Zhang, Liyue Zhang, Panpan Huang, and Wenfeng Liang. Deepseek elastic compute (dsec): A sandbox infrastructure for effective agentic training at scale, 2026. URL https://arxiv.org/abs/2609.22978.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world Github issues? In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Hao Kang, Ziyang Li, Weili Xu, Xinyu Yang, Yinfang Chen, Junxiong Wang, Beidi Chen, Tushar Krishna, Chenfeng Xu, and Simran Arora. ThunderAgent: A simple, fast and program-aware agentic inference system. *arXiv preprint arXiv:2602.13692*, 2026. URL https://arxiv.org/abs/2602.13692v3. Version 3, June 30, 2026.

Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer, 2004. doi: 10.1007/978-3-540-24777-7.

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 Symposium on Operating Systems Principles*, 2023. URL https://arxiv.org/abs/2309.06180v1.

Hanchen Li, Runyuan He, Qiuyang Mang, Qizheng Zhang, Huanzhi Mao, Xiaokun Chen, Hangrui Zhou, Huanchen Zhang, Alvin Cheung, Joseph Gonzalez, and Ion Stoica. Continuum: Efficient and robust multi-turn LLM agent scheduling with KV cache time-to-live. *arXiv preprint arXiv:2511.02230*, 2025. URL https://arxiv.org/abs/2511.02230v7. Version 7, September 8, 2026.

Xiaocheng Li, Chunlin Sun, and Yinyu Ye. Simple and fast algorithm for binary integer and online linear programming. *arXiv preprint arXiv:2003.02513*, 2020. URL https://arxiv.org/abs/2003.02513v3.

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*, pp. 929–945. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/lin-chaofan.

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. *arXiv preprint arXiv:2502.13965*, 2025. URL https://arxiv.org/abs/2502.13965v1.

Andreas Maurer and Massimiliano Pontil. Empirical bernstein bounds and sample variance penalization. *arXiv preprint arXiv:0907.3740*, 2009.

mini-SWE-agent Contributors. mini-SWE-agent. https://mini-swe-agent.com/latest/, 2026. Official documentation; accessed September 8, 2026.

Hongqiu Ni, Han Tian, Chi Zhang, Guopeng Li, and Haisheng Tan. TOPAS: Workflow-aware prefix-state scheduling for multi-agent LLM serving. *arXiv preprint arXiv:2608.25523*, 2026. URL https://arxiv.org/abs/2608.25523v1. Version 1, August 26, 2026.

NVIDIA. Nvidia dynamo: A distributed inference framework. https://docs.nvidia.com/dynamo/, 2025. Technical documentation; scheduler code audited in the literature ledger at commit 1e3478dfe09bdd305d7c58f412fda2dfe5bec7ed.

OpenAI. Computer use. https://developers.openai.com/api/docs/guides/tools-computer-use, 2026. Official documentation; accessed September 11, 2026.

OpenCode Contributors. Improve bash timeout retry hint. https://github.com/anomalyco/opencode/pull/22390, 2026. Pull request 22390, merged April 14, 2026; timeout-termination metadata.

OpenHands Contributors. OpenHands: Remote sandbox. https://docs.openhands.dev/openhands/usage/sandboxes/remote, 2026. Official documentation; accessed September 12, 2026.

Zaifeng Pan, Ajjkumar Dahyalal Patel, Yipeng Shen, Zhengding Hu, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. KVFlow: Efficient prefix caching for accelerating LLM-based multi-agent workflows. In *Advances in Neural Information Processing Systems*, volume 38, pp. 126246–126265, 2025. doi: 10.52202/085713-4208. URL https://papers.neurips.cc/paper_files/paper/2025/hash/b7971d31a7d5eb0f1eed2f8f6f368195-Abstract-Conference.html.

You Peng, Youhe Jiang, Wenshuang Li, Xu Xu, Ke Zhou, Jiawei Jiang, Chen Wang, and Binhang Yuan. Hexagent: Efficient agentic llm serving via workflow- and heterogeneity-aware scheduling, 2026. URL https://arxiv.org/abs/2605.16637.

David Pisinger. An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research*, 114(3):528–541, 1999. doi: 10.1016/S0377-2217(98)00120-9.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation—a KVCache-centric architecture for serving LLM chatbot. In *23rd USENIX Conference on File and Storage Technologies*, pp. 155–170. USENIX Association, 2025. URL https://www.usenix.org/conference/fast25/presentation/qin.

SemiAnalysis. AgentX MVP: Multi-turn coding-agent trace replay. https://github.com/SemiAnalysisAI/agentx-harness/blob/master/docs/tutorials/agentx-mvp.md, 2026a. Official documentation; accessed September 8, 2026.

SemiAnalysis. InferenceX: Open source continuous inference benchmark research platform. https://github.com/SemiAnalysisAI/InferenceX, 2026b. Official repository; accessed September 8, 2026.

SGLang Team. SGLang Model Gateway. https://docs.sglang.io/docs/advanced_features/sgl_model_gateway, n.d. Technical documentation; accessed September 7, 2026.

David B. Shmoys and Eva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461–474, 1993. doi: 10.1007/BF01585178.

SmoothAgent Contributors. SmoothAgent. https://github.com/PanZaifeng/SmoothAgent, 2026. Software repository; README accessed September 12, 2026; commit 7c2b7743e3c3d3e699eca78e244fac02234a5489 audited in the earlier literature ledger.

Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. Preble: Efficient distributed prompt scheduling for LLM serving. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5bc342f48de8264779952fac378f96dc-Abstract-Conference.html.

Yifan Sui, Han Zhao, Rui Ma, Zhiyuan He, Hao Wang, Jianxun Li, Kaiqiang Xu, Kai Chen, and Yuqing Yang. Parallelizing tool execution and LLM generation for low-latency agent serving. *arXiv preprint arXiv:2603.18897*, 2026. URL https://arxiv.org/abs/2603.18897v3.

GLM-4.5 Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, Yifan An, Yilin Niu, Yuanhao Wen, Yushi Bai, Zhengxiao Du, Zihan Wang, Zilin Zhu, Bohan Zhang, Bosi Wen, Bowen Wu, Bowen Xu, Can Huang, Casey Zhao, Changpeng Cai, Chao Yu, Chen Li, Chendi Ge, Chenghua Huang, Chenhui Zhang, Chenxi Xu, Chenzheng Zhu, Chuang Li, Congfeng Yin, Daoyan Lin, Dayong Yang, Dazhi Jiang, Ding Ai, Erle Zhu, Fei Wang, Gengzheng Pan, Guo Wang, Hailong Sun, Haitao Li, Haiyang Li, Haiyi Hu, Hanyu Zhang, Hao Peng, Hao Tai, Haoke Zhang, Haoran Wang, Haoyu Yang, He Liu, He Zhao, Hongwei Liu, Hongxi Yan, Huan Liu, Huilong Chen, Ji Li, Jiajing Zhao, Jiamin Ren, Jian Jiao, Jiani Zhao, Jianyang Yan, Jiaqi Wang, Jiayi Gui, Jiayue Zhao, Jie Liu, Jijie Li, Jing Li, Jing Lu, Jingsen Wang, Jingwei Yuan, Jingxuan Li, Jingzhao Du, Jinhua Du, Jinxin Liu, Junkai Zhi, Junli Gao, Ke Wang, Lekang Yang, Liang Xu, Lin Fan, Lindong Wu, Lintao Ding, Lu Wang, Man Zhang, Minghao Li, Minghuan Xu, Mingming Zhao, Mingshu Zhai, Pengfan Du, Qian Dong, Shangde Lei, Shangqing Tu, Shangtong Yang, Shaoyou Lu, Shijie Li, Shuang Li, Shuang-Li, Shuxun Yang, Sibo Yi, Tianshu Yu, Wei Tian, Weihan Wang, Wenbo Yu, Weng Lam Tam, Wenjie Liang, Wentao Liu, Xiao Wang, Xiaohan Jia, Xiaotao Gu, Xiaoying Ling, Xin Wang, Xing Fan, Xingru Pan, Xinyuan Zhang, Xinze Zhang, Xiuqing Fu, Xunkai Zhang, Yabo Xu, Yandong Wu, Yida Lu, Yidong Wang, Yilin Zhou, Yiming Pan, Ying Zhang, Yingli Wang, Yingru Li, Yinpei Su, Yipeng Geng, Yitong Zhu, Yongkun Yang, Yuhang Li, Yuhao Wu, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yuxuan Zhang, Zezhen Liu, Zhen Yang, Zhengda Zhou, Zhongpei Qiao, Zhuoer Feng, Zhuorui Liu, Zichen Zhang, Zijun Yao, Zijun Yao, Zikang Wang, Ziqiang Liu, Ziwei Chai, Zixuan Li, Zuodong Zhao, Wenguang Chen, Jidong Zhai, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025. URL https://arxiv.org/abs/2508.06471.

Terminal-Bench Contributors. Terminal-Bench: Measuring and evolving with the frontier of agent work. https://github.com/harbor-framework/terminal-bench, 2026. Official repository; accessed September 8, 2026.

vLLM Production Stack Team. vLLM Production Stack. https://docs.vllm.ai/projects/production-stack/en/latest/, n.d. Technical documentation; accessed September 7, 2026.

Noppanat Wadlom, Junyi Shen, and Yao Lu. Efficient llm serving for agentic workflows: A data systems perspective. *arXiv preprint arXiv:2603.16104*, 2026. URL https://arxiv.org/abs/2603.16104.

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. *arXiv preprint arXiv:2607.08565*, 2026a. URL https://arxiv.org/abs/2607.08565v1.

Yifei Wang, Hancheng Ye, Yechen Xu, Cong Guo, Chiyue Wei, Qinsi Wang, Dongting Li, Tingjun Chen, Hai "Helen" Li, Danyang Zhuo, and Yiran Chen. MARS: Efficient, adaptive co-scheduling for heterogeneous agentic systems. *arXiv preprint arXiv:2604.26963*, 2026b. URL https://arxiv.org/abs/2604.26963v2. Version 2, June 15, 2026.

LLM-Core Xiaomi. Mimo-v2.6: Scaling reinforcement learning towards self-improvement.

Yechen Xu, Xinhao Kong, Tingjun Chen, and Danyang Zhuo. Conveyor: Efficient tool-aware LLM serving with tool partial execution. *arXiv preprint arXiv:2406.00059*, 2024. URL https://arxiv.org/abs/2406.00059.

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, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Yichao Yuan, Mosharaf Chowdhury, and Nishil Talati. Kairos: Stateful, context-aware power-efficient agentic inference serving, 2026. URL https://arxiv.org/abs/2604.16682.

Rui Zhang, Chaeeun Kim, Shaoting Feng, Kuntai Du, Yuhan Liu, Yi Zhong, Cheng-Wei Ching, Junchen Jiang, and Liting Hu. Learning agent execution for KV-cache management in agentic serving. *arXiv preprint arXiv:2608.14624*, 2026. URL https://arxiv.org/abs/2608.14624v1.

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. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583, 2024. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

A HISTORICAL TOOL-DURATION ESTIMATOR

The history-based predictor in Section 3.4 uses a lightweight summary of completed calls. A finite-sample margin augments the historical mean, allowing the estimate to account for observed variability and the amount of available history.

For calls without direct timing evidence, we maintain statistics of completed durations keyed by the structured tool name f. Let n_f, $\bar{d}_f$, s_f^2, and R_f denote the sample count, mean, sample variance, and observed duration range. We use the following conservative upper estimate as a duration heuristic (Maurer & Pontil, 2009):

$$\widehat{D}_f = \begin{cases} \bar{d}_f, & n_f = 1, \\ \bar{d}_f + \sqrt{\dfrac{2s_f^2 \log(3/\delta)}{n_f}} + \dfrac{3R_f \log(3/\delta)}{n_f}, & n_f \geq 2, \end{cases} \tag{8}$$

with $\delta = 0.2$. We use global completed-call statistics when tool-specific history is empty. Subtracting elapsed time from $\widehat{D}_f$ gives the remaining-duration estimate. The finite-sample margin is a scheduling heuristic; it does not provide calibrated coverage for an individual call. Continuum also exploits tool-duration history to choose a KV-retention TTL (Li et al., 2025). Our estimate converts this historical timing evidence into a signal for scheduler admission.

If both tool-specific and global histories are empty, the implementation uses a five-second duration default. The remaining-time estimate subtracts elapsed time and clips at zero, as in equation (5). Exposed timing hints take precedence as described in the main text.

B EXTENDED METHOD DETAILS

B.1 JOINT SELECTION AND PLACEMENT ALGORITHM

Algorithm 1: TAO joint selection and placement

Input: frozen candidates P, nodes B, capacities C, eligibility E, quantum Δ
Output: a revalidated set of agent session–node assignments
Compute w_i and $\boldsymbol{v}_{i,b} = (k_{i,b}, r_i, t_i)$ through session evaluation
$\bar{w}_i \leftarrow \lceil w_i/\Delta \rceil$; $\bar{C}_b \leftarrow \lfloor C_b/\Delta \rfloor$
Embed each value to obtain $\boldsymbol{u}_{i,b}$
Initialize $D_0[\mathbf{0}] \leftarrow \mathbf{0}$; other states are unreachable
for candidate $j = 1, \ldots, |P|$ **do**
  $D_j \leftarrow D_{j-1}$, recording skip predecessors
  for each reachable state $\boldsymbol{s}$ in D_{j-1} **do**
    for each eligible node b with $s_b + \bar{w}_j \leq \bar{C}_b$ **do**
      $\boldsymbol{s}' \leftarrow \boldsymbol{s} + \bar{w}_j \boldsymbol{e}_b$
      $\boldsymbol{z} \leftarrow D_{j-1}[\boldsymbol{s}] + \boldsymbol{u}_{j,b}$
      if $\boldsymbol{s}'$ is new or $\boldsymbol{z}$ improves $D_j[\boldsymbol{s}']$ **then**
        Store $\boldsymbol{z}$ and predecessor $(\boldsymbol{s}, j, b)$
  If the budget is exceeded, obtain a BFD plan and skip to revalidation
Otherwise choose the best final state and backtrack to recover the plan
Revalidation:
Discard the plan if a selected agent session's live signature or eligibility changed
return the longest plan prefix feasible under current admission capacities

At every scheduling time, w_i is updated with the memory demand of the total token count minus the shared token capacity for session i, and C_b is updated with the remaining capacity of node b from the metric observation. Algorithm 1 summarizes TAO, from session evaluation to joint scheduling and live-state revalidation.

B.2 ASYNCHRONOUS EXECUTION

The scheduler solves a frozen snapshot in an isolated worker. Before applying the plan, it checks that selected agent sessions remain eligible and match their recorded state signatures, then applies

the longest prefix permitted by current capacities. Changed selected-agent session signatures cause the plan to be discarded. Revalidation preserves feasibility under current admission constraints. We quantify the effect of stale values, discarded plans, and partial application through a loss bound relative to the fresh-snapshot optimum, and measure these execution outcomes alongside solve time and plan age.

Conditional loss bound. Let s and t be planning and application times, x_s the exact snapshot plan, and y_t the revalidated plan (empty if discarded). Write $\mathcal{X}_s^\Delta$ and $\mathcal{X}_t^\Delta$ for the quantized feasible sets. For analysis, represent the lexicographic objective by one fixed scalar score U_τ that preserves its order on the finite plans and values under comparison; the implementation still compares vectors. Define

$$\begin{aligned} \epsilon_{s,t} &= \max_{x \in \mathcal{X}_s^\Delta} |U_t(x) - U_s(x)|, \\ A_{s,t} &= \max_{x \in \mathcal{X}_t^\Delta} U_t(x) - \max_{x \in \mathcal{X}_s^\Delta \cap \mathcal{X}_t^\Delta} U_t(x), \\ T_{s,t} &= [U_t(x_s) - U_t(y_t)]_+. \end{aligned} \tag{9}$$

These terms capture value drift, opportunities introduced by feasibility changes, and value lost during execution. The empty plan ensures a nonempty intersection. Then

$$\max_{x \in \mathcal{X}_t^\Delta} U_t(x) - U_t(y_t) \le A_{s,t} + 2\epsilon_{s,t} + T_{s,t}. \tag{10}$$

The proof in Appendix C inserts the best common feasible plan and applies snapshot optimality plus two value-drift terms. If every plan has at most K assignments and each scalar assignment value changes by at most $\eta_{s,t}$, then $\epsilon_{s,t} \le K\eta_{s,t}$. With an additional drift bound $\eta_{s,t} \le L(t-s)$, the value-drift term becomes $2KL(t-s)$; readiness jumps require separate allowances. An unquantized comparator adds the quantization gap, and a fallback adds its snapshot suboptimality gap.

The bound connects planner freshness to retained scheduling value: short plan age limits value drift under the stated assumptions, while revalidation losses appear explicitly in $T_{s,t}$. Its comparator is the fresh placement optimum on the same realized state; Appendix C details this scope.

C Asynchronous Snapshot Loss: Proof and Scope

Loss relative to a fresh snapshot. We compare the applied plan with the optimal placement at application time on the same observed system state. Let s be the planning time and t the application time. Write $\mathcal{X}_s^\Delta$ and $\mathcal{X}_t^\Delta$ for the discretized feasible sets, embedding both in the union of candidate–node assignments. The empty plan belongs to both sets. Let x_s be the exact snapshot solution and $y_t \in \mathcal{X}_t$ the applied plan, feasible under the current admission contract; a discarded plan has $y_t = 0$.

To express loss numerically, let $U_\tau(x)$ be a fixed positive weighted sum of the objective coordinates in equation (3), with weights that preserve their lexicographic order on the finite collection of plans and objective values considered at s and t. Such weights exist for finite sets, with a scale determined by the coordinate ranges and gaps. The scalar score serves the analysis; the implementation compares vectors directly. In particular, x_s maximizes U_s over $\mathcal{X}_s^\Delta$. Extend the additive objective to all assignments in the union, including plans that have become infeasible.

Define the value variation, loss from changes in feasibility, and execution loss as

$$\begin{aligned} \epsilon_{s,t} &= \max_{x \in \mathcal{X}_s^\Delta} |U_t(x) - U_s(x)|, \\ A_{s,t} &= \max_{x \in \mathcal{X}_t^\Delta} U_t(x) - \max_{x \in \mathcal{X}_s^\Delta \cap \mathcal{X}_t^\Delta} U_t(x), \\ T_{s,t} &= [U_t(x_s) - U_t(y_t)]_+. \end{aligned} \tag{11}$$

Here $[z]_+ = \max(z, 0)$. New candidates, new eligibility, and additional capacity can contribute to $A_{s,t}$; truncation or discard contributes to $T_{s,t}$. Changes in feature values contribute to $\epsilon_{s,t}$.

Conditional bound. For every exact snapshot solve and feasible applied plan,

$$\max_{x \in \mathcal{X}_t^\Delta} U_t(x) - U_t(y_t) \le A_{s,t} + 2\epsilon_{s,t} + T_{s,t}. \tag{12}$$

Table 4: Low-concurrency fallback check for DeepSeek-V4 Flash on SWE-Lite, one TP8 node (8×H100), with a 4–64-worker sweep. Each run serves 300 tasks. Latency is the mean task E2E time, TTFT is the mean client time to first token, and decode is aggregate throughput. Boldface and underlining indicate the best and second-best distinct values.

Workers	Method	E2E latency (s)	TTFT (s)	Decode (tok/s)
4	vLLM Scheduler w/ RR	87.911	0.126	317.855
4	vLLM Scheduler w/ CA	88.390	0.125	318.357
4	Dynamo	92.549	0.102	307.382
4	ThunderAgent	90.099	0.120	312.368
4	TAO (Ours)	89.479	0.117	315.629
8	vLLM Scheduler w/ RR	102.868	0.139	528.074
8	vLLM Scheduler w/ CA	101.698	0.142	555.818
8	Dynamo	103.813	0.112	565.893
8	ThunderAgent	100.610	0.130	527.068
8	TAO (Ours)	102.913	0.132	522.838
16	vLLM Scheduler w/ RR	121.158	0.160	831.120
16	vLLM Scheduler w/ CA	126.907	0.176	861.089
16	Dynamo	126.670	0.130	884.575
16	ThunderAgent	147.709	0.789	722.445
16	TAO (Ours)	126.702	0.153	861.198
32	vLLM Scheduler w/ RR	160.981	0.211	1085.654
32	vLLM Scheduler w/ CA	166.657	0.218	1232.259
32	Dynamo	172.228	0.171	1147.924
32	ThunderAgent	272.845	3.919	766.776
32	TAO (Ours)	171.389	0.195	1240.768
64	vLLM Scheduler w/ RR	225.967	0.278	1788.345
64	vLLM Scheduler w/ CA	225.513	0.279	1658.347
64	Dynamo	229.670	0.231	1704.425
64	ThunderAgent	475.033	9.332	794.991
64	TAO (Ours)	226.268	0.423	1667.524

To prove this, choose z maximizing U_t over the intersection of the two feasible sets. Snapshot optimality gives $U_s(z) \le U_s(x_s)$, and hence

$$\begin{aligned} U_t(z) - U_t(y_t) \le [U_t(z) - U_s(z)] + [U_s(x_s) - U_t(x_s)] \\ + [U_t(x_s) - U_t(y_t)] \le 2\epsilon_{s,t} + T_{s,t}. \end{aligned}$$

Adding $A_{s,t}$ proves equation (12). If the comparator uses the original, undiscretized capacities, add the nonnegative quantization gap $Q_t = \max_{x \in \mathcal{X}_t} U_t(x) - \max_{x \in \mathcal{X}_t^\Delta} U_t(x)$ to its right-hand side.

The bound isolates conditions under which asynchronous planning retains value. If every plan contains at most K assignments and each scalar assignment value changes by at most $\eta_{s,t}$, then $\epsilon_{s,t} \le K\eta_{s,t}$. A further assumption $\eta_{s,t} \le L(t-s)$ yields a variation term $2KL(t-s)$; binary readiness require explicit jump allowances and do not satisfy this assumption automatically. Summing equation (12) over scheduling ticks bounds cumulative loss against fresh-snapshot optima on the realized states, using one common scalarization. It does not bound end-to-end latency or regret against a globally optimal online policy, whose earlier decisions can produce different future states. A uniform numerical bound requires corresponding bounds on state variation, feasibility changes, and discarded value. A fallback solve additionally contributes its snapshot suboptimality gap.

D SINGLE-NODE SWE-LITE EXPERIMENTS ON EIGHT H100 GPUS

DeepSeek-V4 Flash as a low-concurrency fallback check. Table 4 evaluates five schedulers on a single TP8 node across 4–64 workers. Each run serves the 300-task SWE-Lite workload through MiniSWE. This experiment checks that TAO remains competitive when concurrency is low and scheduling has less opportunity to improve performance. At 4–8 workers, its mean task latency stays within 2.3% of the best baseline, it retains over 92% of the best baseline's decode throughput, and its mean TTFT is only 15–20 ms above the best baseline. Thus, TAO incurs little task-latency degradation at low concurrency while maintaining competitive throughput. Together with its leading performance in the main evaluation (Section 4.2), this fallback behavior supports choosing TAO as the default scheduler across the evaluated operating regimes.

Table 5: InferenceX replay on 2 nodes with 4×H100 per node. Latency is reported in seconds per request (s/req). TTFT is recovered client time (s), and decode is node-window decode TPS. These replay measurements use a fixed trace corpus and are reported separately from natural-live task results.

Method	16 workers			32 workers			64 workers		
	Latency (s/req)	TTFT (s)	Decode (tok/s)	Latency (s/req)	TTFT (s)	Decode (tok/s)	Latency (s/req)	TTFT (s)	Decode (tok/s)
Dynamo	5.545	10.844	133.158	3.234	26.542	204.257	5.154	163.674	71.085
vLLM Scheduler w/ RR	7.121	10.861	107.478	5.069	23.058	127.991	4.589	45.852	87.050
vLLM Scheduler w/ CA	5.916	7.421	115.947	**3.103**	**9.006**	211.262	3.825	**25.145**	109.413
TAO (Ours)	**5.382**	**4.756**	**138.701**	3.234	10.302	**215.075**	**3.532**	27.190	**120.469**
Rel. Imprv. (%)	+2.9	+35.9	+4.2	-4.2	-14.4	+1.8	+7.7	-8.1	+10.1

E INFERENCEX REPLAY

TAO also achieves the highest decode throughput at every tested concurrency level on InferenceX, a fixed trace corpus served by DeepSeek on 2 nodes with 4×H100 per node. The replay uses 393 AgentX traces with a 1M-token context limit and 1,800-second profiling windows (SemiAnalysis, 2026a;b); workers count the concurrent session trees. Table 5 reports client-visible TTFT and decode throughput, together with seconds per request computed as $60/R$, where R is native request throughput in requests per minute. This last metric is the reciprocal of aggregate request throughput within the profiling window. Decode throughput improves on the best competing method by 4.2%, 1.8%, and 10.1% at 16, 32, and 64 workers, respectively. TAO also achieves the lowest TTFT at 16 workers; cache-aware scheduling has lower TTFT at 32 and 64 workers. Figure 6 shows the accompanying reuse advantage: TAO maintains the highest prefix-cache hit rate throughout, even as hit rates decrease with concurrency for all four methods.

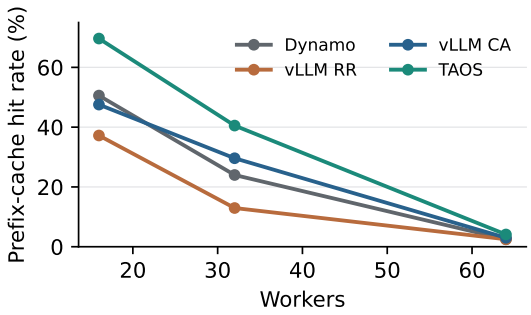

Figure 6: InferenceX prefix-cache hit rate on 2 nodes with 4×H100 per node. TAO maintains higher cache reuse across the tested worker counts.

F SENSITIVITY TO LONGER TOOL-RETURN DELAYS

We examine how longer tool-return delays affect scheduling using a replay of SWE-Lite sessions recorded with MiniSWE. The replay fixes the source prompts, 6,995 model-request records, and 6,942 tool records, while model outputs are generated live. Each session receives a fixed additional tool-return delay drawn from 1–5 s; multipliers of 1, 2, and 4 produce additional delays of 1–5, 2–10, and 4–20 s, respectively. DeepSeek-V4 Flash runs on TP4×2 with 32 workers, a 33,024-token context limit, and a 512-token per-session reserve.

Table 6 compares three configurations: the historical TAO w/o readiness reference, TAO with structured-history tool-duration predictions, and Oracle with the replay's known effective tool duration. The historical reference also disables cached-token decay and shared-token refresh, which both current configurations enable; comparisons with that reference therefore measure the combined configuration change. Acting-token decay remains disabled in both current configurations. All six current cells pass the replay validator, covering all 6,942 tool records. Each cell is a single exploratory run with batch-invariant execution disabled and live model generation.

Table 6: Tool-return delay scaling on SWE-Lite replay sessions. Scales 1, 2, and 4 add 1–5, 2–10, and 4–20 s per tool return, respectively. Oracle uses accurate timing supplied by the known replay tool duration. MAE is logged duration-prediction error, including predictions unused by the no-readiness arm. Decision change is the percentage of exact snapshot comparisons whose plan changes when readiness is removed. Rel. Imprv. compares TAO with TAO w/o readiness: positive percentages denote makespan and mean E2E reductions; the last column gives the decision-change difference in percentage points (pp). Oracle results are a timing-information reference.

Scale	Prediction arm	Makespan (s)	Mean E2E (s)	MAE (s)	Decision change (%)
1	TAO w/o readiness	1,919.906	229.714	6.531	0.000
1	TAO	1,522.076	180.167	9.490	1.647
1	Oracle	1,523.532	180.030	0.030	4.167
	Rel. Imprv.	+20.72%	+21.57%	–	+1.65 pp
2	TAO w/o readiness	2,816.496	352.502	2.593	0.000
2	TAO	2,354.050	279.986	6.315	3.941
2	Oracle	2,301.282	278.011	0.030	4.059
	Rel. Imprv.	+16.42%	+20.57%	–	+3.94 pp
4	TAO w/o readiness	5,090.251	599.155	4.512	0.000
4	TAO	4,108.865	491.829	5.837	7.933
4	Oracle	4,110.209	493.595	0.025	7.398
	Rel. Imprv.	+19.28%	+17.91%	–	+7.93 pp

TAO sustains gains as tool delays grow. Relative to the historical configuration, TAO reduces makespan by 20.72%, 16.42%, and 19.28% and mean session latency by 21.57%, 20.57%, and 17.91% at scales 1, 2, and 4, respectively. These gains span additional tool-return delays from 1–5 s to 4–20 s.

Longer delays make readiness relevant to more placement decisions. For TAO, removing readiness from the same captured state changes 11/668 (1.65%), 53/1,345 (3.94%), and 200/2,521 (7.93%) of the exact DP plan comparisons at multipliers 1, 2, and 4. The rate increases by 6.29 percentage points from scale 1 to scale 4, indicating that tool-return timing affects a growing fraction of placement decisions as delays lengthen. These same-state comparisons isolate readiness's effect on individual placement decisions within the TAO runs.

History-based predictions approach oracle runtime performance. Providing the known replay duration reduces mean absolute prediction error by 99.53–99.69%, to 0.025–0.030 s. At scale 2, it also reduces makespan by 2.24% and mean session latency by 0.71% relative to TAO. Across all three scales, the signed makespan improvements range from -0.10% to +2.24%, and mean session-latency improvements range from -0.36% to +0.71%. Thus, the history-based predictor achieves runtime performance close to the accurate-timing variant in this sweep, even as longer delays increase the frequency of readiness-sensitive decisions.

G Joint Placement Case Studies

Reconstructed snapshots from the 32-worker DeepSeek replay show how joint placement preserves more cache value and admits more sessions than a short-first heuristic (Section 4.3). The figures compare scheduler-level assignment decisions and their cache credit; vLLM manages the physical KV cache. The short-first heuristic is evaluated offline on the same snapshots.

A representative cache-credit snapshot is shown in Figure 7. DP retains 16,791 tokens of cache credit, compared with 13,624 for the short-first heuristic, a 23.2% increase with the same total safe-token remainder. A separate reconstructed snapshot in Figure 8 shows the selection effect: DP admits seven sessions, including long-context session 177, while the heuristic admits six and defers that session.

H Deferred sessions and tail behavior

The lexicographic objective prioritizes larger token counts and thus tends to schedule longer sessions first. In practice, we audit deferral and waiting tails in the same 32-worker DeepSeek TP4×2 evidence. Across 35 reconstructable snapshots, exact DP improved the lexicographic objective in six cases and

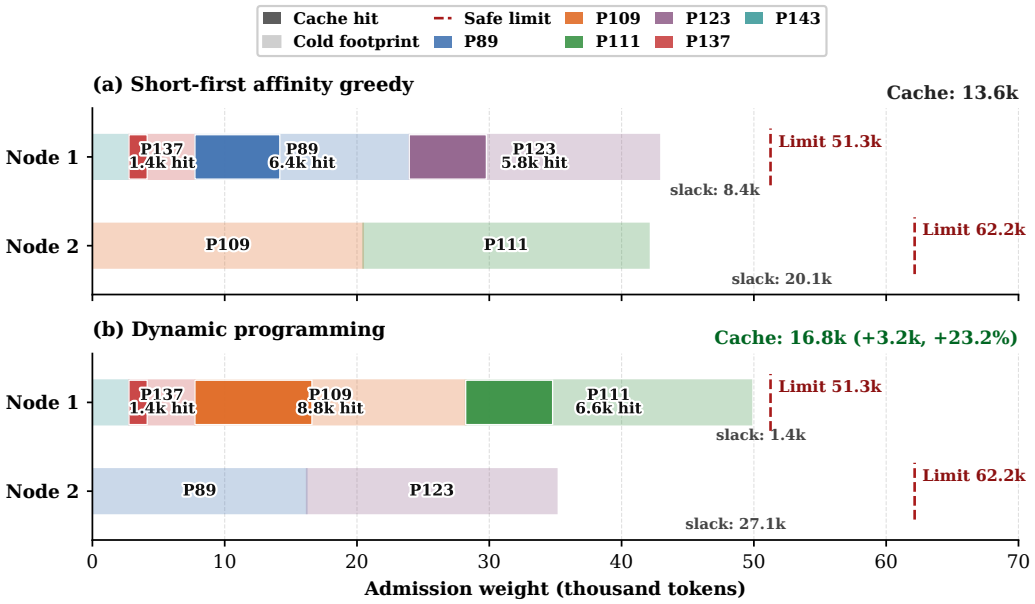

Figure 7: Cache utilization in one placement snapshot. Each program has a distinct hue; the translucent segment is its admission weight and the saturated overlay represents retained cache credit (shown as token labels when space permits). DP retains 16,791 cache-credit tokens versus 13,624 for short-first placement (23.2% more), with the same total safe-token remainder.

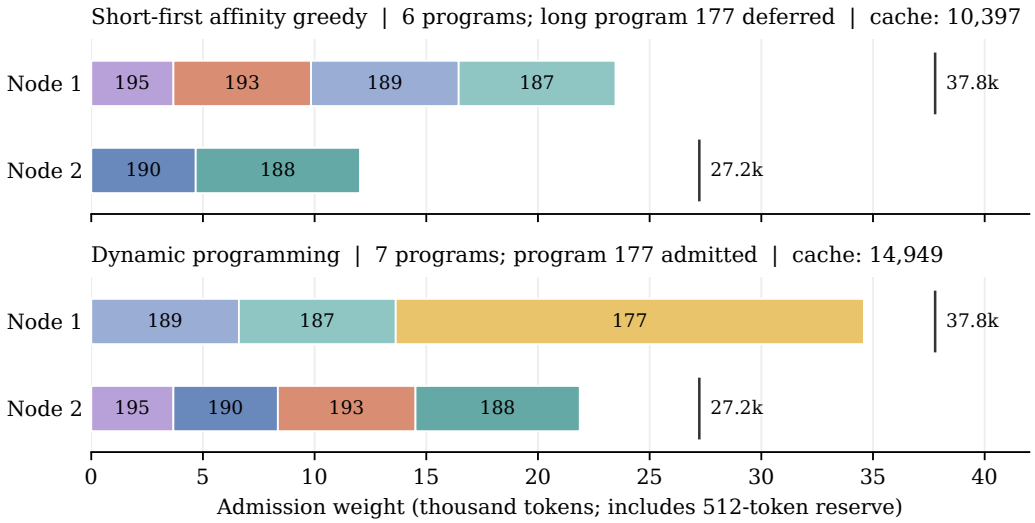

Figure 8: A reconstructed replay snapshot in which DP admits seven sessions while the short-first affinity-greedy heuristic admits six. Candidate widths represent admission weights, including the 512-token reserve. DP admits the long-context session 177; the heuristic leaves it deferred after filling capacity with shorter programs.

matched the short-first heuristic in the remaining 29. In a tie snapshot, programs 168 and 166 had equal cache and readiness coordinates; the capacity constraint selected the larger-context program 168, while program 166 resumed in the next plan. This illustrates a temporary deferral and the capacity-aware tie break rather than a fairness guarantee.

In a separate validated fixed-length replay at the same topology and worker count (219 programs and 6,995 replay records), TAO reduces pause-wait mean/P90 from $13.022/26.288$ s for the ThunderAgent reference to $1.109/1.162$ s, a 91.5%/95.6% reduction. Its task E2E P90 falls from 204.157 to 154.831 s (24.2%), pause events fall from 633 to 187 (70.5%), throughput rises from 477.3 to 565.1 requests/min (18.4%), and prefix-hit rate rises by 2.41 percentage points; both cells report zero vLLM preemptions. The tool-end-to-first-token P90 also falls from 0.764 to 0.409 s.

The matched natural-live component comparison reports the same direction at this scale: the complete configuration has tool-end-to-first-token mean/P90 $0.203/0.299$ s and pause-wait mean/P90 $0.972/1.051$ s, whereas the separately measured *w/o Readiness* variant reports $0.227/0.358$ s and $1.067/1.079$ s (Table 3).

I Scheduler Overhead Analysis

Across the four-replica Qwen runs at 96–160 workers, planning submissions rise from 98 to 1,681. At 160 workers, completed assignment solving averages 1.59 ms with a P95 of 3.68 ms; the two measured parts together account for 4.03 s (0.112% of the run). Thus, the recorded scheduling work remains below 0.12% at four replicas and up to 160 workers.

Extensibility. The assignment-value interface allows the placement solver to evolve with the serving pool while retaining the same workload features. As the number of node capacity dimensions grows, a future solver could relax $x_{i,b} \in \{0, 1\}$ to $0 \leq x_{i,b} \leq 1$ and solve a linear program (LP), optimizing the value coordinates in order while fixing each preceding optimum. Capacity-aware rounding, followed by feasibility checks and capacity repair, would replace combinatorial state enumeration with approximate placement (Shmoys & Tardos, 1993). Rounding trades the DP's lexicographic optimality for scalability. The evaluated implementation uses budgeted DP with a best-fit-decreasing fallback; the detailed accounting below reports fallback frequency and the limits of the timing coverage.

Table 7 expands the scheduling work and planner outcomes summarized above. State preparation runs synchronously in the scheduler, while the planner worker solves assignments asynchronously. Result delay measures dispatch, solving, and delivery together. The two- and four-replica deployments exercise this design under different models and concurrency levels; they are separate deployment evaluations rather than a controlled replica-scaling sweep.

The state budget bounds DP search, with best-fit-decreasing placement handling exhausted searches. Fallback occurs in 5.24% of submissions at 64 DeepSeek workers and 3.06–8.98% in the Qwen sweep. Measured work covers state preparation and completed solves; exhausted searches and fallback execution are untimed, so work/run is a partial accounting rather than a measure of CPU utilization or request-latency overhead. Discards and partial applications record how live revalidation handles state changes before application. The 16-worker DeepSeek run submits one plan, so its timing entry describes that single solve.

J Extended Related Work

Capacity-constrained optimization. Dynamic programming and exact multiple-knapsack algorithms provide established tools for selecting items and assigning them to capacity-limited resources (Bellman, 1957; Kellerer et al., 2004; Pisinger, 1999). Node-dependent assignment values place our formulation within the multiple-knapsack/generalized-assignment family. Approximation algorithms address the computational cost of this family (Chekuri & Khanna, 2005; Shmoys & Tardos, 1993), while online resource-allocation algorithms analyze decisions under specified arrival models (Li et al., 2020). TAO uses this classical optimization machinery to connect agentic session evaluation with joint admission and node placement. Its additive, lexicographic objective makes cache opportunity, tool readiness, and token footprint precedence explicit, and its asynchronous executor revalidates assignments against live state. Section 3.5 and Appendix B.2 specify snapshot optimality and the bounds for asynchronous application.

KV locality and agent scheduling. PagedAttention improves physical KV allocation and sharing, while SGLang's RadixAttention reuses prefixes within the serving runtime (Kwon et al., 2023; Zheng

Table 7: Scheduler overhead on SWE-Lite (300 tasks per run). TP4×2 uses two replicas; TP2×4 uses four. State preparation is the time to collect the current candidates and capacities; assignment solving is the completed asynchronous planning time; result delay includes dispatch and delivery. Measured work sums state preparation and completed solving, excluding failed attempts and BFD execution. Work fractions divide by run duration. Outcome entries are count (%): fallback/discard rates divide by submissions, partial rates by application events (including empty plans).

	DeepSeek-V4 Flash, TP4×2			Qwen3-32B, TP2×4		
Workers	16	32	64	96	128	160
Run duration (s)	2321.1	1514.2	1199.0	3151.2	3077.9	3583.0
State preparation total (ms)	1.37	30.58	394.95	50.67	1024.09	1588.17
Assignment solving total (ms)	0.55	48.97	3163.36	52.77	1351.51	2437.68
Assignment solving mean (ms)	0.55	1.48	12.50	0.56	1.04	1.59
Assignment solving p95 (ms)	0.55	2.08	88.84	0.62	2.43	3.68
Result delay mean (ms)	337.49	20.83	27.86	6.07	4.23	4.89
Result delay p95 (ms)	337.49	22.83	105.09	5.29	9.03	9.99
State preparation/run (%)	0.00006	0.00202	0.03294	0.00161	0.03327	0.04433
Assignment solving/run (%)	0.00002	0.00323	0.26384	0.00167	0.04391	0.06804
Measured scheduler work/run (%)	0.00008	0.00525	0.29678	0.00328	0.07718	0.11236
Plan submissions	1	33	267	98	1395	1681
Completed plans	1	33	253	95	1301	1530
Plan applications	1	33	252	95	1299	1527
Nonempty applications	1	33	240	89	678	687
BFD fallback	0 (0.00)	0 (0.00)	14 (5.24)	3 (3.06)	94 (6.74)	151 (8.98)
Discarded	0 (0.00)	0 (0.00)	1 (0.37)	0 (0.00)	2 (0.14)	3 (0.18)
Partial applications	0 (0.00)	0 (0.00)	2 (0.79)	0 (0.00)	0 (0.00)	0 (0.00)

et al., 2024). Preble co-optimizes distributed prefix reuse and computation balance; Mooncake combines a distributed KV store with disaggregated serving (Srivatsa et al., 2025; Qin et al., 2025). Dynamo and production gateways expose cache- and load-aware request scheduling (NVIDIA, 2025; vLLM Production Stack Team, n.d.; SGLang Team, n.d.), and llm-d provides pluggable scoring components (Guy et al., 2025). SMetric exploits session structure by balancing first-turn requests and assigning later turns to nodes with reusable cache, using a global cache tier to limit the cost of redistribution (Wang et al., 2026a). TAO builds on these locality and load-balancing principles with joint optimization of assignments for waiting agent sessions under shared capacity. This joint view accounts for how one admission consumes capacity needed by other session–node assignments.

Program-level schedulers also preserve information across model calls. Autellix uses attained service and critical-path information to prioritize single-threaded and distributed programs (Luo et al., 2025). ThunderAgent introduces the program abstraction, state-aware pausing, elapsed-time discounting of tool execution programs' effective token demand, and a global waiting queue (Kang et al., 2026). TAO adds node-specific cache values and readiness to the joint placement objective. Tool-gap cache management addresses a complementary decision: InferCept chooses among preservation, swapping, and recomputation by modeling resource waste (Abhyankar et al., 2024); Continuum selects a cache-retention TTL using historical tool-duration distributions, reconstruction cost, and returning programs' queueing cost, together with TTL-aware program ordering (Li et al., 2025). These systems control physical KV retention. TAO coordinates scheduler-level admission opportunities, with predicted readiness distinguishing plans that have equal higher-precedence cache value. The serving engine controls the resulting cache residency.

Workflow, prediction, and tool execution. Workflow structure provides information beyond an isolated request. Parrot exposes inter-request dependencies through Semantic Variables, and Helium compiles workflow query plans for proactive reuse and cache-aware execution (Lin et al., 2024; Wadlom et al., 2026). KVFlow uses an Agent Step Graph and steps-to-execution to guide eviction and prefetching (Pan et al., 2025); CacheScout learns execution transitions online without a predefined graph (Zhang et al., 2026). Their predictions guide context residency and prefetching; readiness supplies a binary near-term admission-opportunity signal derived from tool timing. SAGA combines workflow graphs, session-affinity batching, and work stealing (Guo et al., 2026). TOPAS also uses

heuristic state searching: it scores prefix-residency and request-admission states using remaining workflow paths, movement costs, and downstream reuse, with bounded candidate search and greedy packing (Ni et al., 2026). Its task-dependent utility and engine-level residency decisions differ from our fixed additive value contract and exact multi-node scheduling problem.

Prediction also supports latency-oriented scheduling. ELIS predicts response length, while Kairos combines workflow-aware priority with memory-aware dispatch (Choi et al., 2025; Chen et al., 2025). Recent work separates already-ready turns from their release, using a mean–CVaR objective and an adaptive budget for committed inference work (Feng et al., 2026). This addresses tail-aware release after dependencies complete; TAO additionally evaluates near-term return opportunities during external tool execution. Complementary mechanisms change the execution timeline itself: Conveyor exposes partial tool execution, Sutradhara overlaps tool work and prefill through orchestrator–engine cooperation, and PASTE speculates on future tools while coordinating tool execution with returning model agent sessions (Xu et al., 2024; Biswas et al., 2026; Sui et al., 2026). SmoothAgent prepares transformed context on a lookahead stream, and Pie exposes programmable generation and KV handlers (SmoothAgent Contributors, 2026; Gim et al., 2025). TAO operates in the scheduler, using timing evidence to coordinate admission and node placement.

Deployment and workloads. AgentSysBench documents substantial non-LLM latency and deployment-dependent resource demands (Chang et al., 2026). MARS explicitly coordinates GPU inference, CPU tool pressure, admission, and KV retention in heterogeneous serving (Wang et al., 2026b). In our CGDC scenario, distributed executors manage tool CPU and memory, and their resource pressure reaches the GPU scheduler through tool-return timing. TAO captures this interaction through readiness when allocating GPU admission capacity. OpenHands supports separate execution environments (OpenHands Contributors, 2026). Coding and terminal workloads, together with InferenceX/AgentX trace replay, provide complementary evidence about task completion and serving behavior (mini-SWE-agent Contributors, 2026; Jimenez et al., 2024; Terminal-Bench Contributors, 2026; SemiAnalysis, 2026b;a), with workload-specific timing metrics defined in the evaluation.