SPECROUTE: MULTI-STEP SPECULATION FOR EFFICIENT LLM AGENT SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents interleave model reasoning with tool execution, forming observation-dependent sequences of model calls that incur substantial end-to-end latency. Conventional serving optimizations focus on individual LLM calls, while single-step action speculation overlaps the next tool invocation with ongoing reasoning. Both leave the agent's multi-turn execution chain largely on the critical path. We present SpecRoute, a serving runtime that accelerates LLM agents through multi-step speculative execution. A lightweight draft agent speculates on future tool actions, interleaving tool execution with draft-model inference while the target processes its current context. At target tool-call boundaries, the target performs incremental, prefill-based verification of speculative observations and incorporates accepted results with KV reuse. A slack-aware scheduler coordinates batch formation and target–draft engine switching on a shared GPU, balancing speculative progress against interference with target execution. Across six agent benchmarks, SpecRoute achieves 1.28–1.56× single-request speedups over vLLM with comparable task performance. Under online serving, mean completion time decreases by 28–36% at low load and 9–20% at the highest tested loads.

1 INTRODUCTION

LLM agents carry out complex tasks by combining language model reasoning with actions in external environments (Yao et al., 2023; Schick et al., 2023). This integration enables agents to retrieve knowledge adaptively in agentic retrieval-augmented generation (Jin et al., 2025a), investigate open-ended questions through deep research (OpenAI, 2025; Tongyi DeepResearch Team et al., 2025), and develop and debug software (Yang et al., 2024; Wang et al., 2025). During execution, agents use observations and feedback to refine their plans and guide subsequent actions (Yao et al., 2023; Shinn et al., 2023). This interaction introduces dependencies between successive steps. When an action depends on earlier tool feedback, the model must receive and process that feedback before deciding what to do next. Repeated model invocations and tool waits therefore accumulate along the request's critical path. When a large model drives this loop, its reasoning cost is incurred repeatedly, making end-to-end latency a central challenge for agent execution.

Efforts to accelerate LLM agents span model inference, agent serving, and speculative execution. General LLM serving systems improve batching, prefill scheduling, and KV-cache management (Yu et al., 2022; Kwon et al., 2023; Agrawal et al., 2024; Zheng et al., 2024), while token-level speculative methods accelerate generation within a model call (Leviathan et al., 2023; Cai et al., 2024). Agent-serving systems exploit continuity across model calls through cache retention, offloading, and coordination between tool execution and inference (Li et al., 2025; Bian et al., 2025; Biswas et al., 2026). At the action level, speculative execution overlaps tool calls with ongoing computation (Ye et al., 2026), while selective speculation and semantic verification can reduce the need for full reasoning at individual steps (Huang et al., 2025; Zhong et al., 2026). Building on these advances, we study how completed observations from multiple speculative tool interactions can reduce the reasoning and tool use needed to complete an agent task.

Two complementary opportunities emerge from the reasoning demands and execution characteristics of LLM agents. First, a sequence of dependent tool interactions need not entail an equally long sequence of expensive reasoning steps. When the available context makes the next action sufficiently predictable, lightweight reasoning can advance through several interactions, using each actual tool

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.

response to guide the next prediction. The resulting observations can provide information that would otherwise require additional rounds of large-model reasoning to obtain. Second, decoding accounts for a substantial share of execution time in profiled agent workloads (Kim et al., 2026). For tool calls that use CPUs or external services, the distinct resource requirements create an opportunity to hide tool execution latency behind ongoing GPU decoding. If several speculative interactions finish within this window and provide the information needed for subsequent decisions, the agent can bypass some intermediate LLM reasoning and keep the associated tool waits off the critical path.

We present SpecRoute, a speculative serving framework that advances the acquisition of tool observations to reduce the target model's remaining reasoning and tool interactions. A lightweight draft agent explores ahead through a bounded sequence of actions, using actual tool feedback to guide each step. At a tool-call boundary, the target verifies accumulated observations in draft execution order, stopping at the first rejection and incorporating only the accepted prefix into its context. This lets it continue from information gathered across multiple interactions. Target-side KV states produced during verification are reused when the causal prefix is unchanged. If no observation is accepted, the target proceeds with normal tool execution. To turn reduced target work into end-to-end savings, SpecRoute also manages the competition between draft and target inference for a shared GPU. A slack-aware scheduler allocates draft execution according to the time remaining before the target's next tool-call boundary, balancing candidate readiness against interference with target progress. It interleaves draft bursts with target inference and lets target execution continue while speculative tool calls are in flight.

We implement SpecRoute on vLLM and evaluate it on six benchmarks under single-request execution and online serving. Compared with a vLLM-based ReAct agent, SpecRoute achieves single-request speedups of 1.28 to 1.56$\times$ and reduces target-side search iterations by 40% to 55%. Under online serving, it lowers mean latency by 28% to 36% at low load and by 9% to 20% at the highest tested arrival rates. Answer accuracy is 0.3 percentage points lower on average, with the largest decrease of 2.0 percentage points on DeepSearch. These results demonstrate latency reductions across the evaluated workloads and serving loads, with task-dependent changes in accuracy.

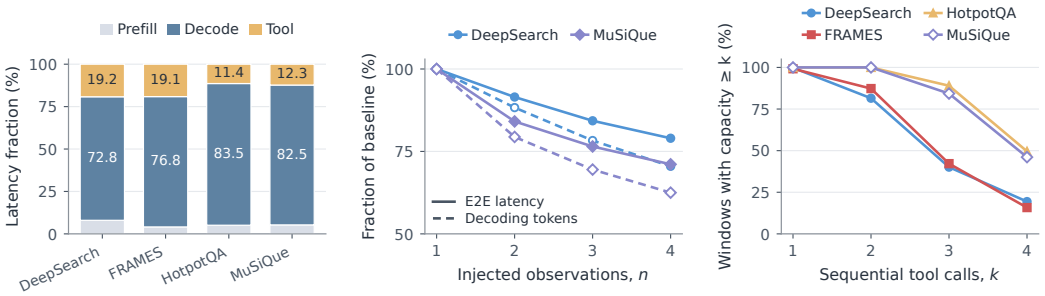
2 Background and Motivation

Tool-using agents alternate between model inference and tool execution (Yao et al., 2023; Schick et al., 2023). In each round, the model prefills newly available context and decodes until it emits a tool call or a final answer. Each tool call suspends the request until its output, or *observation*, becomes available, so prefill, decoding, and tool waits accumulate across rounds. With Qwen3.6-27B on a single NVIDIA A100 GPU, decoding contributes 72.8% to 83.5% of request latency and tool waits another 11.4% to 19.2% (Figure 1(a)). General-purpose inference optimizations and agent-aware KV-cache management accelerate LLM calls and reduce repeated context processing (Kwon et al., 2023; Zheng et al., 2024; Li et al., 2025; Bian et al., 2025), but do not directly reduce the number of reasoning rounds. We therefore investigate whether incorporating observations from subsequent tool interactions can eliminate intermediate reasoning rounds and how much of the associated tool latency can be hidden behind ongoing decoding.

2.1 Reducing Reasoning with Future Observations

Intermediate LLM reasoning often serves to acquire information needed later in a trajectory. Making several tool results available together may therefore reduce the reasoning required to obtain that information. To test this possibility, we record a complete baseline trajectory for each request, then replay the request up to its first tool call. Before dispatching the call, we inject the first n recorded tool outputs into the model's context and resume generation. With $n = 1$, the model receives only the pending call's result, while $n = 4$ also provides three future results.

Figure 1(b) compares end-to-end replay latency and cumulative decoding tokens from injection to request completion, averaging per-request ratios relative to $n = 1$. Increasing n from one to four reduces latency by 21.0% on DeepSearch and 28.9% on MuSiQue, while reducing decoding tokens by 29.5% and 37.5%, respectively. Since the replay uses outputs collected offline, it isolates this benefit from the cost of acquiring them during online execution.



(a) Latency breakdown (b) Multi-observation replay (c) Tool overlap potential

Figure 1: **Motivation for multi-step speculation.** (a) Request latency by execution phase. (b) End-to-end replay latency and cumulative decoding tokens from injection to request completion. The first n recorded tool outputs are injected at the first tool-call boundary. (c) Fraction of decoding windows with capacity $C \geq k$, where C is the number of sequential tool executions that fit within a window, irrespective of the number of remaining calls.

2.2 Hiding Tool Latency within Decoding Windows

Tools running on CPUs or external services can execute concurrently with GPU decoding, allowing tool latency to be hidden within decoding windows. Figure 1(c) reports the fraction of windows that can accommodate at least k sequential tool executions. On DeepSearch and FRAMES, 81.5% and 87.3% of windows accommodate at least two executions, respectively, while 40.1% and 42.2% accommodate at least three. On HotpotQA and MuSiQue, 89.0% and 84.3% accommodate at least three, and 49.6% and 46.1% accommodate at least four. These window capacities indicate an opportunity for multi-step execution during decoding, before accounting for the additional computation needed to generate tool calls and verify them.

2.3 Design Challenges

Generating and verifying multi-step speculation. A lightweight *draft model* can predict upcoming tool calls while the main *target model* continues reasoning. Since future actions depend on the results of preceding tool calls, predicting multiple steps without feedback can compound errors. This motivates a *draft agent* that advances through actual tool interactions. Exact token matching can reject semantically useful results. Verification must therefore reliably assess whether speculative tool calls provide the information the target needs, with sufficiently low overhead to preserve the potential latency savings.

Scheduling draft and target execution. The heterogeneous draft and target models run in separate inference engines that time-share a GPU. The runtime must choose when to advance each engine and which ready requests to batch within it. Advancing drafts can deliver observations sooner but delay the target, while favoring targets can leave speculation unfinished. Engine selection and batch formation must therefore account for both observation readiness and target progress to preserve the anticipated latency savings.

3 SpecRoute

SpecRoute uses real tool feedback to draft observations ahead of target execution, then incorporates an accepted prefix at a tool-call boundary. Figure 2 shows its drafting, verification, and scheduling mechanisms.

3.1 Execution Model and Overview

Tool calls specify an operation and its arguments, returning data, computed results, or execution feedback. We assume speculative calls do not modify task-relevant external state. Target and draft

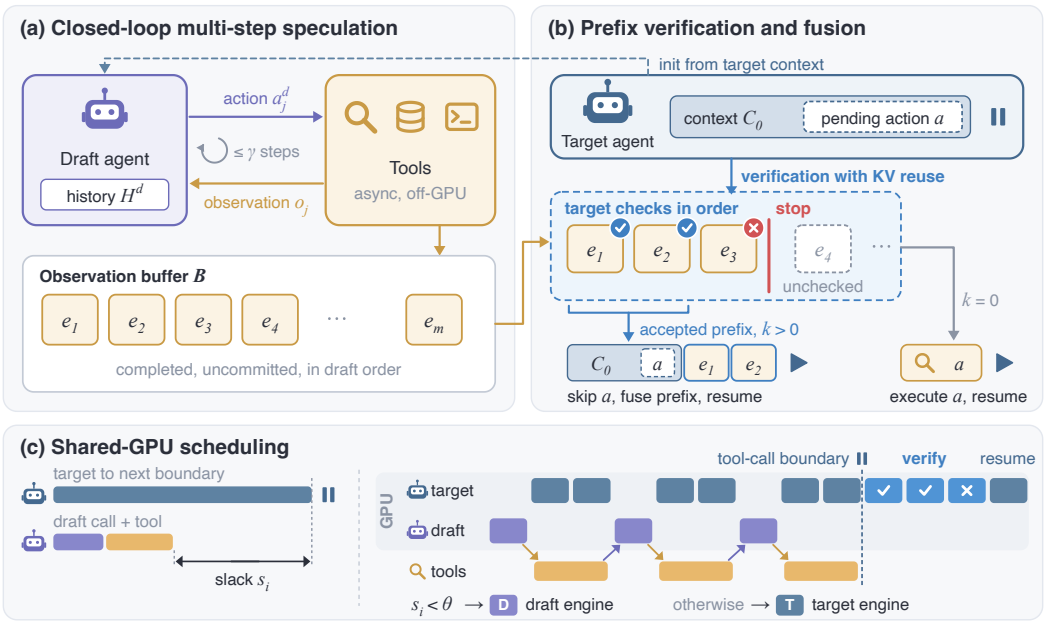


Figure 2: **SpecRoute overview.** (a) The draft agent alternates actions a_j^d with real tool observations o_j, buffering them in B at most γ steps ahead. (b) The target checks candidates in order, stops at the first rejection, and resumes from the accepted prefix ($k > 0$) or executes the pending call a ($k = 0$). (c) Each iteration serves the drafter when some $s_i < \theta$ and the target otherwise, while tool calls run off the GPU.

inference share one GPU through separate engines, while CPU-based tools and external services can execute asynchronously. The target controls observation acceptance and produces the final answer.

Each request maintains a canonical target context C, a draft-local history H^d, and a buffer B of completed, uncommitted observations. Draft reasoning and unaccepted observations remain outside C. Verification occurs at a *tool-call boundary*, after the target generates a complete call a but before dispatch. Thus, C already contains the target's reasoning and pending call.

3.2 Closed-Loop Multi-Step Speculation

SpecRoute advances the drafter through actual tool interactions, using each returned observation to inform its next action. Each request has at most one paired drafter, initialized from a condensed task summary and the canonical target context. With draft model D, the drafter appends its jth action a_j^d and actual result o_j to its local history,

$$a_j^d \sim D(H_j^d), \qquad o_j = \mathrm{Tool}(a_j^d), \qquad H_{j+1}^d = \mathrm{Update}(H_j^d, a_j^d, o_j). \tag{1}$$

The runtime stores completed observations in B in draft execution order, with provenance and context-cost metadata for verification.

Once the drafter emits a complete tool call, the runtime dispatches it and yields the GPU, and the next dependent draft step becomes ready when the result returns. Dependencies thus stay sequential within each draft trajectory, while tool waits can overlap target computation when resources permit. A candidate several steps ahead therefore costs both the intervening draft inference and tool execution time. A per-boundary horizon γ bounds this lookahead and buffer capacity bounds the accumulated observations. A larger horizon can surface information for later decisions earlier, at the cost of more draft computation and tool calls.

3.3 PREFIX VERIFICATION AND OBSERVATION FUSION

Different tool calls can yield observations that support the same subsequent decision. The target therefore verifies observation usefulness without requiring exact call matching, allowing reuse across action trajectories.

Prefix acceptance. Let C_0 be the target context at the boundary, a its pending call, and $e_1, \ldots, e_m$ the completed candidates in draft execution order, each a draft tool call paired with its tool response. The target checks them in order. Each verification prompt pairs the current context (with a and any previously accepted observations) with the next candidate and a usefulness instruction, and the target returns Accept or Reject. Each Accept appends the candidate and permits the next check, and the first Reject ends verification at this boundary. With k candidates accepted, the committed context is

$$C_{\text{commit}} = \text{Append}(C_0, e_1, \ldots, e_k), \qquad 0 \le k \le m, \tag{2}$$

where Append serializes these call-response pairs into the target context and returns C_0 when $k = 0$. Thus $k = j - 1$ if e_j is the first rejection and $k = m$ if all are accepted. On rejection, the runtime discards the temporary verification branch but keeps the accepted prefix, so the rejected candidate and all later ones stay outside the committed context and unchecked at this boundary.

Fusion and fallback. If the accepted prefix is nonempty ($k > 0$), the runtime skips the pending call and resumes target generation from C_{commit}. Otherwise, when no candidate is available or the first is rejected, it executes the pending call and resumes after the result returns. This *observation fusion* can supply information from multiple tool interactions at a single resumption without a separate summary, and may remove later target tool calls and their reasoning from the critical path.

Verification cost and KV reuse. Each verification prompt places the usefulness instruction after the candidate. After an Accept, the runtime truncates the instruction and verdict tokens together with their key-value (KV) entries. The next check and resumed generation then reuse the candidate's KV directly, so accepted candidates incur no prefill recomputation. Verification overhead is thus limited to the trailing instruction and verdict of each check, at most $k + 1$ per boundary and far shorter than typical candidates, plus the discarded prefill of the first rejected candidate.

3.4 SHARED-GPU SCHEDULING

Speculation reduces end-to-end latency only when the target time it saves exceeds the delay added by drafting, verification, and resource contention. Serving the drafter on a shared GPU can postpone target computation and hence the next tool-call boundary, so Algorithm 1 selects an engine per iteration from predicted candidate completion time and target progress, then batches its work.

Engine selection. The eligible set E holds active drafters whose tool feedback has arrived and whose lookahead and buffer budgets permit further speculation. It excludes drafters whose target already waits at a tool-call boundary, since a new candidate cannot join the verification there and would only delay that verification. Each $i \in E$ receives the slack

$$s_i = \widehat{T}_i^{\text{boundary}} - \widehat{T}_i^{\text{pipeline}}, \tag{3}$$

where $\widehat{T}_i^{\text{boundary}}$ predicts the time to the target's next tool-call boundary from decoding progress and recent time per token, as future decoding length is unknown online. The estimate $\widehat{T}_i^{\text{pipeline}}$ covers the remaining tool-call generation, expected tool latency, and any draft KV reload. Both estimates are refreshed as the engines receive service. The draft engine is selected if the urgent set $U = \{i \in E \mid s_i < \theta\}$ is nonempty and the target engine otherwise, so a positive θ doubles as a margin for prediction error. Because the rule weighs timing urgency rather than candidate utility, U also admits negative-slack drafters, whose candidates are predicted to miss the boundary. If the selected engine has no admissible work, the other runs instead, including non-urgent drafters when no target work is ready, so the GPU idles only when neither engine can progress.

Batch formation. Each batch builder admits ready work within its token and memory limit (L_D or L_T) and orders each queue internally. Target batches follow the priority order of Q, namely

Algorithm 1 Slack-Driven Engine Selection and Batch Formation

Require: request state R, ready target queues Q, batch limits L_D, L_T, threshold θ
at each engine-iteration boundary
Refresh R and Q from completed inference and tool events
$E \leftarrow$ drafters with ready feedback and budget whose target is not at a boundary
$s_i \leftarrow \widehat{T}_i^{\text{boundary}} - \widehat{T}_i^{\text{pipeline}}$ for each $i \in E$, estimated from R
$U \leftarrow \{i \in E \mid s_i < \theta\}$
$B_D \leftarrow \text{DraftBatch}(U, L_D)$ ▷ empty if $U = \emptyset$
if $B_D = \emptyset$ **then**
    $B_T \leftarrow \text{TargetBatch}(Q, L_T)$
    if $B_T = \emptyset$ **then** ▷ GPU otherwise idle
        $B_D \leftarrow \text{DraftBatch}(E, L_D)$
    end if
end if
if $B_D \neq \emptyset$ **then**
    Reload required draft KV and run one draft-engine iteration on B_D
    Dispatch fully generated draft tool calls and offload their draft KV
else if $B_T \neq \emptyset$ **then**
    Run one target-engine iteration on B_T
    Apply prefix decisions, reset drafters after rejections, and dispatch target tool calls
end if
Update request progress and phases for the next scheduling decision

verification Q_{ver}, resumed-request prefill Q_{resume}, new-request prefill Q_{new}, and decoding Q_{dec}, extending mixed prefill/decode scheduling (Agrawal et al., 2024; Patel et al., 2024) to agent phases while keeping target and draft batches separate. Because this order applies only after engine selection, verification does not preempt a selected draft iteration. Within a request, the first check is ready at the boundary and each later check only after the preceding Accept, whereas checks from different requests can share a batch. Engine selection sets no fairness quota, so sustained urgent drafting can delay target service, and the target priority order does not bound the wait of lower-priority work.

4 Evaluation

We evaluate whether SpecRoute reduces end-to-end completion time, whether its benefit persists under concurrent load, and how it changes target reasoning. Because semantic reuse of observations can change the agent's trajectory, task quality is reported alongside latency.

4.1 Experimental Setup

System and models. The testbed has NVIDIA A100 80 GB GPUs attached through PCIe Gen 4, an Intel Xeon Platinum 8369B CPU, and 500 GB of host memory. SpecRoute builds on vLLM (Kwon et al., 2023) and uses Qwen3.6-27B as the target and Qwen3.5-2B as the drafter, both in BF16. The two models run in separate engines, and the runtime schedules their inference iterations together with target-side verification and asynchronous tool execution.

Workloads and baselines. The six benchmarks span deep research, retrieval, and multi-hop question answering. DeepSearch (xbench) (Chen et al., 2025) and FRAMES (Krishna et al., 2025) involve repeated evidence acquisition and synthesis, while HotpotQA (Yang et al., 2018), 2WikiMultiHopQA (Ho et al., 2020), MuSiQue (Trivedi et al., 2022), and Bamboogle (Press et al., 2023) add complementary question structures and reasoning depths. All systems share the input queries, decoding parameters, search backend, replayable search results, and answer evaluator.

We compare SpecRoute with three baselines, whose names refer to the implementations and configurations evaluated here. **vLLM** runs the target alone as a sequential ReAct agent (Yao et al., 2023) with continuous batching and paged KV management. **Continuum** adds Continuum-style KV time-to-live scheduling (Li et al., 2025) to this target-only agent, retaining context across tool pauses.

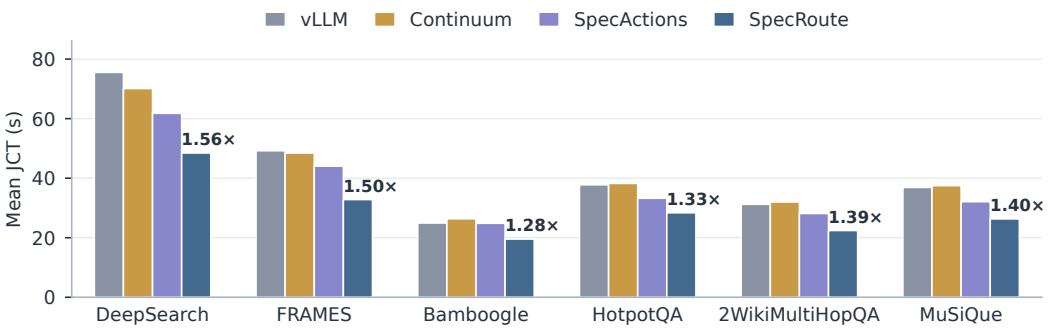


Figure 3: Single-request mean JCT on six benchmarks. SpecRoute reduces JCT by 1.28–1.56× over vLLM (annotated), with the largest gain on DeepSearch, where it removes the most target iterations.

Table 1: Judge-evaluated answer accuracy (%).

	DeepSearch	FRAMES	Bamboogle	2Wiki	HotpotQA	MuSiQue	Avg.
vLLM	57.0	65.2	84.8	86.0	71.2	56.5	70.1
SpecRoute	55.0	64.4	85.6	85.1	73.0	55.9	69.8

SpecActions adapts Speculative Actions (Ye et al., 2026) with the same Qwen3.5-2B drafter, which predicts a single search action verified by exact action matching.

Measurements. Single-request experiments report end-to-end job completion time (JCT) and the number of target reasoning iterations. Online experiments use Poisson arrivals and report mean and P90 JCT as the offered request rate increases. An LLM judge scores task quality by comparing each final answer with its reference and checking the supporting evidence. The evaluation covers only search-based agents, not other tool domains.

4.2 End-to-End Latency and Task Quality

SpecRoute has the lowest single-request JCT on all six benchmarks (Figure 3). Its speedup over vLLM averages $1.41\times$ and ranges from $1.28\times$ on Bamboogle to $1.56\times$ on DeepSearch, where JCT falls from 75.5 s to 48.5 s, saving approximately 27 s per request. The reported speedups over SpecActions range from $1.17\times$ to $1.34\times$ ($1.26\times$ on average). Of the baselines, Continuum averages approximately $1.00\times$ over vLLM and SpecActions $1.12\times$. SpecRoute improves on both by combining completed speculative observations with subsequent target reasoning, although attributing the gain to individual components requires controls beyond these full-system comparisons.

These gains come with a small, task-dependent change in accuracy (Table 1), while SpecRoute removes 40% to 55% of target iterations (Appendix A.1). Mean accuracy is 69.8% for SpecRoute and 70.1% for vLLM. Relative to vLLM, the largest decrease is 2.0 percentage points on DeepSearch, while accuracy rises by 0.8 points on Bamboogle and 1.8 points on HotpotQA. These point estimates do not establish lossless execution or statistical non-inferiority.

4.3 Online Serving

SpecRoute has the lowest mean and P90 JCT at every rate from 0.05 to 6 req/s on DeepSearch, FRAMES, 2WikiMultiHopQA, and MuSiQue (Figure 4). Relative to vLLM, it reduces mean JCT by 28–36% at the lowest rate and by 9–20% at the highest, where it remains 7–16% faster than SpecActions. Single-request speedups thus do not transfer unchanged to concurrent serving.

The relative gain narrows as load rises, consistent with queueing delay, which every system incurs at a given arrival rate, taking a larger share of JCT while speculative inference and verification compete with more target work. The absolute reduction nevertheless grows with load on DeepSearch,

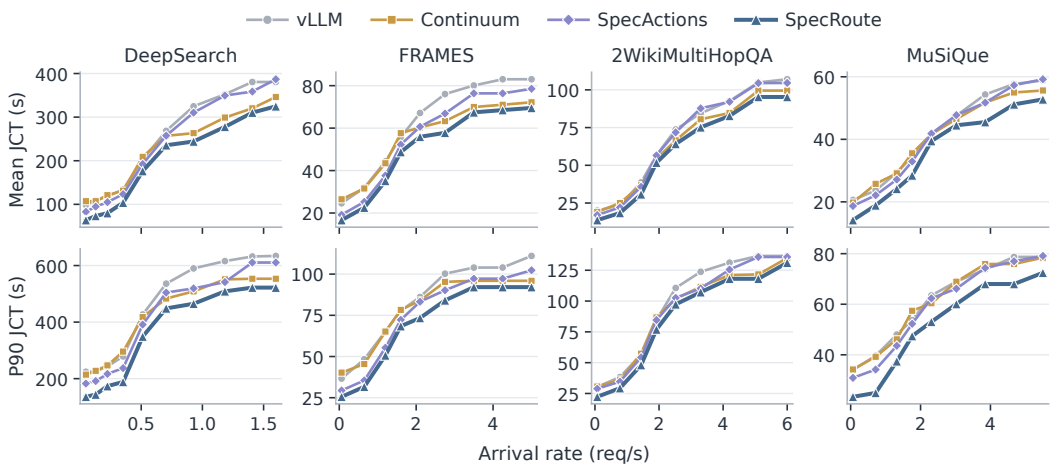
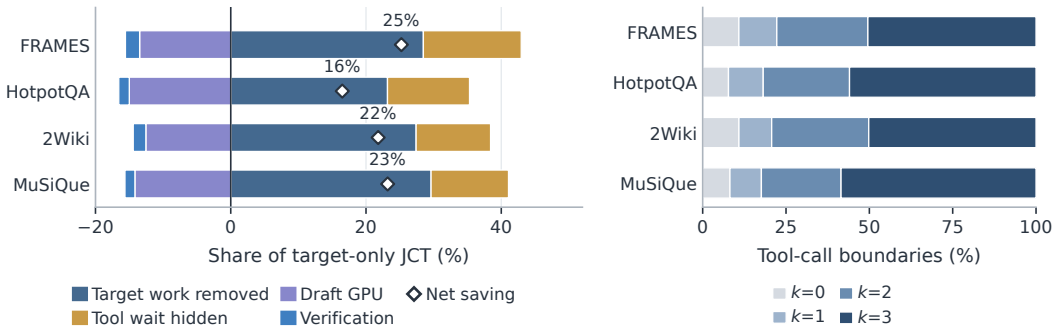


Figure 4: Mean (top) and P90 (bottom) JCT under Poisson arrivals with increasing request rates, each column on its own scales. SpecRoute has the lowest latency at every rate on all four benchmarks. Its advantage is largest at low load and narrows as queueing delay grows for all systems.



(a) Sources of the JCT saving

(b) Accepted observations per boundary

Figure 5: Anatomy of the speedup. (a) JCT saving relative to the target-only agent, split into removed target work and hidden tool wait minus draft GPU time and the verification of rejected candidates. Diamonds mark the net saving. (b) Candidates accepted per tool-call boundary, where $k = 0$ falls back to ordinary execution.

FRAMES, and 2WikiMultiHopQA. Fewer target iterations per request shorten each request's residence time and hence the number of in-flight requests at a given arrival rate, which slows the growth of queueing delay.

The baselines follow opposite trends. SpecActions is 8–24% faster than vLLM at the lowest rate, where hiding one search latency shortens the critical path, but its advantage largely disappears at the highest rates because it does not remove target iterations. Continuum instead gains under load, consistent with KV capacity becoming a bottleneck, where retaining the KV state of requests waiting on tools avoids recomputation after eviction. At the highest rates it outperforms vLLM on all four benchmarks and ranks second. These curves show overall latency gains without isolating scheduling or verification effects.

4.4 Decomposing the Speedup

Removed target work accounts for most of the saving. Skipped target iterations save 23% to 30% of target-only JCT and overlapped searches a further 11% to 14% (Figure 5a). Pausing the target for

draft decoding costs 13% to 15%, verifying rejected candidates about 2%, and contention and other overheads another 2%, leaving a net saving of 16% to 25%. Accepted candidates add no verification cost of their own, because the target must prefill each observation it uses, so draft decoding is the dominant cost of speculation.

Speculation pays off only when several observations are accepted. Acceptance falls with candidate position, from 80–86% for the first candidate to 66–75% for the third, and 78–82% of boundaries accept at least two observations (Figure 5b). This pattern is consistent with a per-boundary drafting and verification cost that a single accepted observation does not recover.

5 Related Work

LLM serving and KV reuse. Modern LLM serving systems improve the efficiency of model invocations. Orca introduces iteration-level batching, vLLM manages KV memory through paged attention, and SGLang optimizes structured language model programs (Yu et al., 2022; Kwon et al., 2023; Zheng et al., 2024). Sarathi-Serve coordinates prefill and decoding through chunked prefills, while Splitwise and DistServe separate these phases to accommodate their different resource demands (Agrawal et al., 2024; Patel et al., 2024; Zhong et al., 2024). PromptCache and RAGCache reuse prompt and retrieval states, CacheBlend combines cached contexts with selective recomputation, and Mooncake adopts KV-centric serving (Gim et al., 2024; Jin et al., 2025b; Yao et al., 2025; Qin et al., 2025). SpecRoute complements these optimizations by using verified tool observations to reduce the reasoning and tool interactions needed to complete a task.

Agent serving. Agent-serving systems coordinate dependent model calls, tool execution, and persistent state. Parrot captures dependencies among application calls (Lin et al., 2024), while SystemX and AIMS manage agent workflows across cloud or edge resources (Chaudhry et al., 2026; Liu et al., 2026a). Continuum and TokenCake manage KV retention and placement across agent turns (Li et al., 2025; Bian et al., 2025), and Sutradhara coordinates tool execution with model computation (Biswas et al., 2026). These systems improve execution around tool boundaries. SpecRoute uses those boundaries to incorporate observations acquired by a lightweight draft agent. When the accepted information supports subsequent decisions, the target can continue without reproducing every intermediate interaction that produced it.

Speculation for LLMs and agents. Speculative decoding accelerates generation through target verification of draft tokens (Leviathan et al., 2023), with extensions for tree verification, disaggregated execution, and adaptive speculation depth (Miao et al., 2024; Zhang et al., 2026; Li et al., 2026). Lookahead Reasoning explores reasoning-aware speculative decoding (Fu et al., 2025), while SpecReason uses a larger model to assess reasoning steps proposed by a lightweight model (Pan et al., 2025). For tool-using agents, Speculative Actions and speculative tool-call execution overlap predicted actions with ongoing inference (Ye et al., 2026; Nichols et al., 2025), while SPAgent and DualSpec explore selective reasoning and heterogeneous action drafting (Huang et al., 2025; Zhong et al., 2026). Recent work also studies adaptive multi-step planning and speculation over observations or action chains (Guan et al., 2026; Saberi et al., 2026; Chen et al., 2026; Liu et al., 2026b). SpecRoute verifies completed observations from a feedback-driven draft trajectory to reduce subsequent target work. Its scheduler coordinates draft execution with target progress on a shared GPU, accounting for contention that can offset the savings from speculation.

6 Conclusion

We presented SpecRoute, a serving framework that accelerates agents using read-only tools by making future observations available earlier. A lightweight draft agent advances through multiple interactions using actual tool feedback. The target verifies the resulting observations and continues from accepted information, reducing its remaining reasoning and tool calls. Shared-GPU scheduling balances speculative progress against interference with target execution while allowing off-GPU tool calls to overlap with decoding. The resulting gains depend on observation utility and timeliness, as well as verification and resource overheads.

Reproducibility statement. Section 3 specifies the execution model, multi-step speculation, prefix verification with KV reuse, and shared-GPU scheduling, and Algorithm 1 gives the scheduling procedure in pseudocode. Section 4.1 reports the hardware, the vLLM-based implementation, the target and draft models with their numerical precision, the six public benchmarks, the three baseline configurations, and the evaluation measures. All systems share the same input queries, decoding parameters, search backend, replayable search results, and answer evaluator. Both models run with frozen pretrained weights, and no training or fine-tuning is performed.

Ethics statement. This work studies inference-time acceleration of pretrained LLM agents on existing public benchmarks. It involves no human subjects and collects no private or personally identifying data. SpecRoute speculatively executes only read-only tools, so discarded speculation changes no external state, although its additional tool calls add load to the tool providers.

AI use statement. Generative AI tools were used to assist with language editing, presentation and organization, literature search, and feedback on the communication of experimental results. The research questions, methodology, implementation, experiments, and scientific conclusions were developed and verified by the authors. All AI-assisted material was reviewed and revised by the authors, who take full responsibility for the paper.

REFERENCES

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 117–134, 2024.

Zhuohang Bian, Feiyang Wu, Zhuoran Li, Teng Ma, and Youwei Zhuo. TokenCake: A KV-Cache-centric Serving Framework for LLM-based Multi-Agent Applications. *arXiv preprint arXiv:2510.18586*, 2025. URL https://arxiv.org/abs/2510.18586v4.

Anish Biswas, Kanishk Goel, 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.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads. In *International Conference on Machine Learning*, volume 235, pp. 5209–5235. PMLR, 2024. URL https://proceedings.mlr.press/v235/cai24b.html.

Gohar Irfan Chaudhry, Esha Choukse, Haoran Qiu, Íñigo Goiri, Rodrigo Fonseca, Adam Belay, and Ricardo Bianchini. SystemX: Resource-efficient agentic workflow orchestration in cloud platforms. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*. USENIX Association, 2026.

Hao Mark Chen, Jinnan Guo, Wayne Luk, and Hongxiang Fan. AOSpec: Action and Observation Co-Speculation for Low-Latency Agent Serving. *arXiv preprint arXiv:2608.00881*, 2026. URL https://arxiv.org/abs/2608.00881.

Kaiyuan Chen, Yixin Ren, Yang Liu, Xiaobo Hu, Haotong Tian, Tianbao Xie, Fangfu Liu, Haoye Zhang, Hongzhang Liu, Yuan Gong, Chen Sun, Han Hou, Hui Yang, James Pan, Jianan Lou, Jiayi Mao, Jizheng Liu, Jinpeng Li, Kangyi Liu, Kenkun Liu, Rui Wang, Run Li, Tong Niu, Wenlong Zhang, Wenqi Yan, Xuanzheng Wang, Yuchen Zhang, Yi-Hsin Hung, Yuan Jiang, Zexuan Liu, Zihan Yin, Zijian Ma, and Zhiwen Mo. xbench: Tracking Agents Productivity Scaling with Profession-Aligned Real-World Evaluations. *arXiv preprint arXiv:2506.13651*, 2025. URL https://arxiv.org/abs/2506.13651.

Yichao Fu, Rui Ge, Zelei Shao, Zhijie Deng, and Hao Zhang. Scaling speculative decoding with lookahead reasoning. *Advances in Neural Information Processing Systems*, 38:172454–172486, 2025.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338, 2024.

Yilin Guan, Qingfeng Lan, Fei Sun, Dujian Ding, Devang Acharya, Chi Wang, William Yang Wang, and Wenyue Hua. Dynamic Speculative Agent Planning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/pdf/0b98c9fcf66a2a3ac799c3c6824030256c2f7ce3.pdf.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020.

Zixiao Huang, Wen Zeng, Tianyu Fu, Tengxuan Liu, Yizhou Sun, Ke Hong, Xinhao Yang, Chengchun Liu, Yan Li, Quanlu Zhang, et al. Reducing latency of llm search agent via speculation-based algorithm-system co-design. *arXiv preprint arXiv:2511.20048*, 2025.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan O. Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to Reason and Leverage Search Engines with Reinforcement Learning. In *Conference on Language Modeling*, 2025a. URL https://openreview.net/forum?id=Rwhi91ideu.

Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Shufan Liu, Xuanzhe Liu, and Xin Jin. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *ACM Transactions on Computer Systems*, 44(1):1–27, 2025b.

Jiin Kim, Byeongjun Shin, Jinha Chung, and Minsoo Rhu. The cost of dynamic reasoning: Demystifying ai agents and test-time scaling from an ai infrastructure perspective. In *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–16. IEEE, 2026.

Satyapriya Krishna, Kalpesh Krishna, Anhad Mohananey, Steven Schwarcz, Adam Stambler, Shyam Upadhyay, and Manaal Faruqui. Fact, fetch, and reason: A unified evaluation of retrieval-augmented generation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4745–4759, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

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/2511.02230.

Zikun Li, Zhuofu Chen, Remi Delacourt, Gabriele Oliaro, Zeyu Wang, Qinghan Chen, Shuhuai Lin, April Yang, Zhihao Zhang, Zhuoming Chen, et al. Adaserve: Accelerating multi-slo llm serving with slo-customized speculative decoding. In *Proceedings of the 21st European Conference on Computer Systems*, pp. 36–54, 2026.

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)*, pp. 929–945, 2024.

Shiyi Liu, Haiying Shen, Shuai Che, Mahdi Ghandi, and Mingqin Li. Aims: Cost-efficient llm-based agent deployment in hybrid cloud-edge environments. In *Proceedings of the 21st European Conference on Computer Systems*, pp. 1862–1878, 2026a.

Zeyu Liu, Souvik Kundu, and Peter A. Beerel. Speculative Macro Commit for Faster Tool-Using Agents. *arXiv preprint arXiv:2609.03236*, 2026b. URL https://arxiv.org/abs/2609.03236.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 932–949, 2024.

Daniel Nichols, Prajwal Singhania, Charles Jekel, Abhinav Bhatele, and Harshitha Menon. Optimizing agentic language model inference via speculative tool calls. *arXiv preprint arXiv:2512.15834*, 2025.

OpenAI. Deep Research System Card. Technical report, OpenAI, February 2025. URL https://openai.com/index/deep-research-system-card/.

Rui Pan, Yinwei Dai, Zhihao Zhang, Gabriele Oliaro, Zhihao Jia, and Ravi Netravali. SpecReason: Fast and Accurate Inference-Time Compute via Speculative Reasoning. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/12c45a68e8433b21b91cd47731387fa4-Abstract-Conference.html.

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)*, pp. 118–132. IEEE, 2024.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023.

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 (FAST 25)*, pp. 155–170, 2025.

Mehrdad Saberi, Keivan Rezaei, and Soheil Feizi. SpecHop: Continuous Speculation for Accelerating Multi-Hop Retrieval Agents. *arXiv preprint arXiv:2605.21965*, 2026. URL https://arxiv.org/abs/2605.21965.

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. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Tongyi DeepResearch Team, Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, Kuan Li, Liangcai Su, Litu Ou, Liwen Zhang, Pengjun Xie, Rui Ye, Wenbiao Yin, Xinmiao Yu, Xinyu Wang, Xixi Wu, Xuanzhong Chen, Yida Zhao, Zhen Zhang, Zhengwei Tao, Zhongwang Zhang, Zile Qiao, Chenxi Wang, Donglei Yu, Gang Fu, Haiyang Shen, Jiayin Yang, Jun Lin, Junkai Zhang, Kui Zeng, Li Yang, Hailong Yin, Maojia Song, Ming Yan, Minpeng Liao, Peng Xia, Qian Xiao, Rui Min, Ruixue Ding, Runnan Fang, Shaowei Chen, Shen Huang, Shihang Wang, Shihao Cai, Weizhou Shen, Xiaobin Wang, Xin Guan, Xinyu Geng, Yingcheng Shi, Yuning Wu, Zhuo Chen, Zijian Li, and Yong Jiang. Tongyi DeepResearch Technical Report, 2025. URL https://arxiv.org/abs/2510.24701.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Daniel Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a4b6ad6b48850c0c331d1259fc66a69c-Abstract-Conference.html.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

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*, pp. 2369–2380, 2018.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving for rag with cached knowledge fusion. In *Proceedings of the twentieth European conference on computer systems*, pp. 94–109, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Naimeng Ye, Arnav Ahuja, Georgios Liargkovas, Yunan Lu, Kostis Kaffes, and Tianyi Peng. Speculative actions: A lossless framework for faster ai agents. In *The Fourteenth International Conference on Learning Representations*, 2026.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX symposium on operating systems design and implementation (OSDI 22)*, pp. 521–538, 2022.

Ziyi Zhang, Ziheng Jiang, Chengquan Jiang, Menghan Yu, Size Zheng, Haibin Lin, Xin Liu, and Henry Hoffmann. Swiftspec: Disaggregated speculative decoding and fused kernels for low-latency llm inference. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 2197–2211, 2026.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Shuzhang Zhong, Baotong Lu, Qi Chen, Chuanjie Liu, Fan Yang, and Meng Li. Dualspec: Accelerating deep research agents via dual-process action speculation. *arXiv preprint arXiv:2603.07416*, 2026.

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)*, pp. 193–210, 2024.

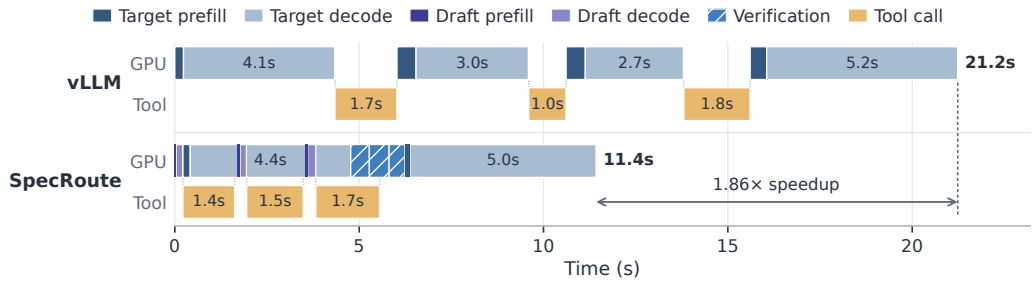


Figure 6: A representative 2WikiMultiHopQA execution. Separate lanes show GPU work and external tool execution. Draft inference and verification consume GPU time, while tool execution can overlap it.

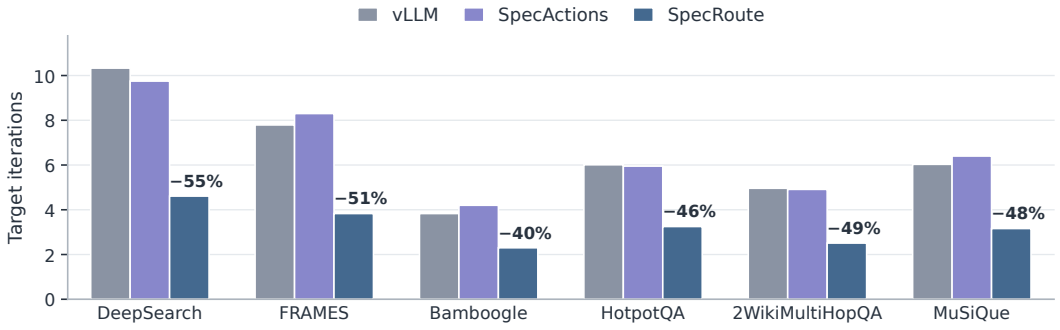


Figure 7: Average target iteration counts across six benchmarks in the web setting. Annotations report the reduction for SpecRoute relative to vLLM.

A Appendix

A.1 Execution Timeline and Iteration Counts

SpecRoute reduces the target iteration count by approximately 40% to 55%, averaging 48% across benchmarks in Figure 7. DeepSearch decreases from 10.34 to 4.61 iterations, while Bamboogle decreases from 3.83 to 2.30. Their relative reductions align with the largest and smallest latency improvements, respectively. This pattern is consistent with observations arriving early enough to eliminate some intermediate target reasoning. Iteration reduction is nevertheless larger than the end-to-end latency reduction, as the remaining target work, speculation, and verification still require time.

Figure 6 illustrates this distinction for one 2WikiMultiHopQA request. The baseline completes four target iterations in 21.2 s, including approximately 16.7 s of target inference and 4.5 s of exposed tool waits. SpecRoute completes two target iterations in 11.4 s, a 1.86× speedup. The execution includes approximately 0.7 s of draft GPU work and 1.5 s of verification. External tool calls overlap GPU work, while draft inference and target-side verification share the GPU with target decoding. This example illustrates the execution pattern rather than isolating the causal effect of any individual component.