AGENTREPLAY: TOKEN-WISE TRACE REPLAY IS ESSENTIAL FOR FAIR SERVING SYSTEM PERFORMANCE BENCHMARKING

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-based agents execute multi-turn workflows with interleaved model inference and tool calls, making efficient serving increasingly important. However, evaluating serving optimizations is challenging because identical tasks can produce different execution trajectories. Changes in generated tokens can alter subsequent prompts, tool calls, and reasoning turns, making it difficult to distinguish system improvements from workload variation. Greedy decoding does not guarantee identical outputs, while replaying only sequence lengths loses token information that affects prefix caching and mixture-of-experts (MoE) routing. To address these problems, we propose AgentReplay, a configurable trace record-and-replay framework for agent serving. AgentReplay records input/output tokens, expert selections, request dependencies, and tool durations. During replay, it forces the recorded output tokens while performing normal autoregressive computation, with optional controls for MoE expert selection and tool delays. This allows different systems to execute the same recorded workload while retaining their own batching, scheduling, and parallelization decisions. We further separate trajectory generation from performance evaluation, enabling compatible smaller models to replay long-horizon traces collected with more capable models. Our experiments show that token-wise replay in AgentReplay effectively eliminates workload variation that greedy decoding and length-wise replay cannot avoid, enabling fairer performance comparisons across serving configurations.

1 INTRODUCTION

Large language model (LLM)-based agents solve complex tasks through repeated reasoning and tool use (Yao et al., 2023). Their multi-turn execution and growing contexts introduce substantial computation and memory costs. To improve serving efficiency, existing systems optimize scheduling, batching, memory management, and decoding (Kwon et al., 2023; Zheng et al., 2024a; Leviathan et al., 2023). Fair evaluation requires both the baseline and optimized systems to execute the same workload.

However, using the same initial tasks does not ensure the same workload. LLM outputs can vary across runs, and these outputs determine the agent's subsequent actions. For example, a few different tokens can change a tool call or cause early termination, producing a different trajectory. In such cases, a lower execution time may simply come from fewer tokens or reasoning turns rather than a more efficient system. This makes it difficult to attribute performance gains to the optimization being evaluated.

Greedy decoding and length-wise replay only partially address this problem. Setting the temperature to zero removes sampling randomness, but numerical differences across batch sizes and serving configurations can still change the highest-scoring token (Yuan et al., 2026; He & Thinking Machines Lab, 2025; Qiang et al., 2026). Length-wise replay fixes input and output lengths without preserving token content. This can affect prefix caching: if a response differs from the recorded history in the next prompt, its KV cache may no longer be reusable (Zheng et al., 2024a; Qin et al., 2025; Liu et al., 2025). For mixture-of-experts (MoE) models, different tokens can also select different experts, changing load balance, memory access, and communication (Yu et al., 2026; Yao et al.,

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.

2024; DeepSeek-AI, 2025). Therefore, matching sequence lengths alone does not ensure comparable execution work.

Besides LLM generation, tool execution introduces another source of variation. A longer tool call delays the next LLM request and leaves its KV cache idle for longer. This changes request concurrency and memory pressure, affecting performance comparisons even when the optimization targets only LLM serving.

To address these problems, we propose AgentReplay, a configurable trace record-and-replay framework for vLLM (Kwon et al., 2023). The key idea is to replay recorded agent behavior while measuring how each serving configuration executes it. Specifically, *token-wise replay* supplies recorded input tokens and forces each output token while performing normal autoregressive computation. *MoE router replay* restores logical expert selections at each layer and token position, while *tool-duration replay* reproduces the delay of each tool call. The replay driver follows recorded request dependencies, allowing batching, scheduling, and completion times to change with the serving system. Users can enable these controls independently.

We further use token-wise replay to support long-horizon evaluation with smaller models. Small models reduce resource requirements but may fail before reaching long contexts or many interaction turns. AgentReplay allows a compatible smaller model to execute trajectories collected with a more capable model. The target tokenizer defines a fixed token workload that retains the recorded content and turn structure. This supports lower-cost performance evaluation without requiring the smaller model to solve the original tasks.

Our experiments show that token-wise replay in AgentReplay effectively eliminates workload variation that greedy decoding and length-wise replay cannot avoid. By preserving the recorded token sequences rather than only the initial tasks or sequence lengths, AgentReplay enables fairer comparisons of serving performance under the same agent workload. This paper makes three contributions:

- We identify how output and tool-time variation affects agent serving benchmarks and motivate token-wise replay for fair comparisons.
- We propose AgentReplay, supporting configurable token-wise, MoE router, and tool-duration replay while retaining model computation.
- We enable long-horizon serving evaluation with smaller models by separating trajectory generation from performance measurement.

2 BACKGROUND

Agent workloads. LLM agents solve long-horizon tasks through repeated interactions with users, tools, and external environments (Yao et al., 2023; Shinn et al., 2023; Yang et al., 2024; Wang et al., 2025). As shown in the upper half of Figure 1, each agent turn submits the accumulated context to an LLM, which prefills the input and decodes an output token by token. The output may answer the user, invoke a tool, or determine the agent's next action. After a tool or the environment returns an observation, the agent appends it to the interaction history and starts another turn. Requests from one agent are therefore stateful and intermittent: later turns depend on earlier model outputs and tool observations, while variable external work separates consecutive LLM invocations.

An execution graph captures dependencies among LLM and tool invocations, including concurrent branches. Together with token sequences, tool delays, and task arrivals, it defines the workload observed by the serving engine.

LLM serving. The lower half of Figure 1 shows the main components of a modern LLM serving engine. Incoming requests first enter a queue, and the scheduler forms iteration-level batches to share accelerator capacity across prefill and decode (Yu et al., 2022). The model executor runs GPU kernels and collectives under tensor, pipeline, expert, or context parallelism (Shoeybi et al., 2019; Liu et al., 2024; Jacobs et al., 2023; Fedus et al., 2022), while the memory manager places model parameters and KV caches across GPU memory, CPU memory, and secondary storage. Serving performance depends on decisions at each layer. Prior work improves batching and request scheduling (Yu et al., 2022; Agrawal et al., 2024; Zheng et al., 2024b), kernels and operator execution (Ye et al., 2025; Dong et al., 2024; Pan et al., 2025a; Dao et al., 2022; Sanovar et al., 2025), parallel and disaggregated

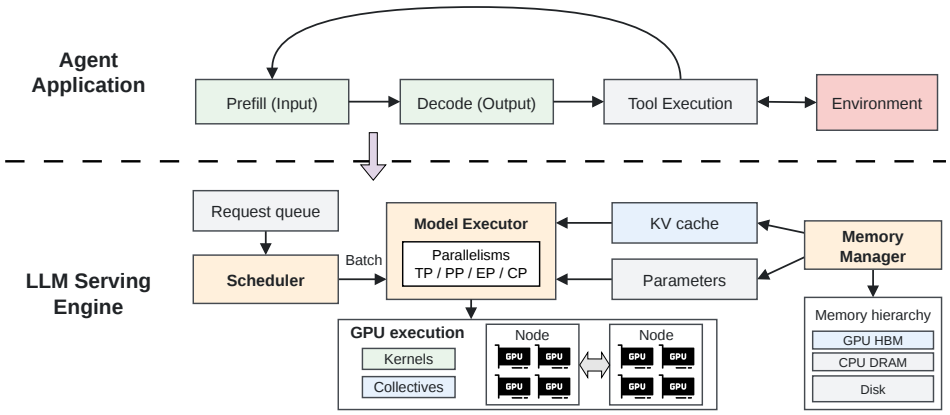

Figure 1: Agent execution and its LLM serving backend. The agent application repeatedly alternates between model inference and external tool or environment interactions. The serving engine queues and batches the resulting requests, executes the model under the configured parallelism, and manages parameters and KV caches across the memory hierarchy.

execution (Zhong et al., 2024; Patel et al., 2024; Zhu et al., 2025; DeepSeek-AI et al., 2025), and KV-cache management (Qin et al., 2025; Xie et al., 2026; Liu et al., 2025; Gao et al., 2024). Agent-oriented systems further exploit tool timing or workflow structure to schedule requests and manage cached state (Abhyankar et al., 2024; Li et al., 2025; Pan et al., 2025b; 2026a; Chen et al., 2026; Kang et al., 2026; Pan et al., 2026b). These optimizations must be evaluated under a controlled workload so that changes in agent behavior do not obscure system effects.

Content affects serving work. Sequence lengths determine tensor dimensions, but token contents also affect serving performance. For example, prefix caching reuses KV tensors when requests share the same token prefix (Kwon et al., 2023; Zheng et al., 2024a). Suppose the first turn generates a response c that is included in the next prompt. If length-wise replay generates c' instead, the system may need to recompute the KV cache for c, even when $|c| = |c'|$. In MoE models, the router selects a subset of experts for each token (Yang et al., 2025). Different selections change the batch size of each expert and, with expert parallelism, the devices that exchange activations. To compare expert placement or communication optimizations, we therefore distinguish which logical experts are selected from where and how they execute.

Timing affects subsequent work. Tool calls create gaps between dependent LLM invocations. Many tools execute as CPU processes, perform file or network I/O, or invoke remote services. OS scheduling, resource contention, and remote-service load make tool durations variable. Real coding-agent traces accordingly exhibit tool-dependent, heavy-tailed latencies (Zhu et al., 2026). During each gap, other requests continue executing, and the serving system may keep, offload, or evict the waiting agent's KV cache (Abhyankar et al., 2024; Li et al., 2025). A change in tool duration therefore affects both the arrival time of the next request and the memory pressure while the agent waits. Tool timing matters even when the benchmark evaluates only the LLM serving system.

Benchmarking objective. A serving benchmark should expose each compared configuration to the same workload properties that affect the optimization under study, while leaving the serving mechanisms themselves free to change. Given trace τ, we compare its execution cost $C(S_1, \tau)$ and $C(S_2, \tau)$ under serving configurations S_1 and S_2. The trace fixes the enabled workload properties while the serving engine controls batching, scheduling, cache placement, kernels, communication, and physical expert placement. Repeating the same trace measures residual runtime variation without allowing a changed agent trajectory to redefine the work. Live-agent evaluation remains necessary to assess task success and behavioral effects. Length-wise replay is sufficient only when the mechanism under study is insensitive to token content and other omitted workload properties.

3 AGENTREPLAY

Figure 2 shows the design of AgentReplay. The recorder first collects a real agent execution, including LLM requests and tool calls. The replay driver then uses this trace to evaluate different serving configurations. Three replay controls preserve token sequences, logical expert selections, and tool delays, respectively. Our evaluation uses vLLM (Kwon et al., 2023).

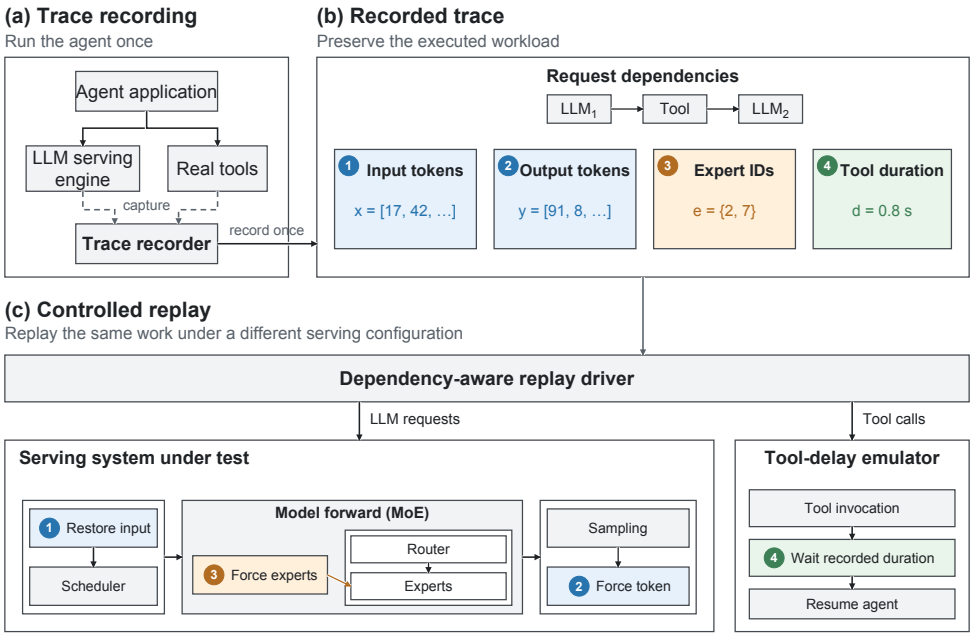

Figure 2: Overview of AgentReplay. (a) The recorder captures LLM and tool invocations during agent execution. (b) The trace stores request dependencies, input/output tokens, expert selections, and tool durations. (c) The replay driver follows these dependencies and submits requests to the serving system and tool-delay emulator. Users can select replay controls independently, while model computation, scheduling, and expert placement remain configurable. Token IDs and durations in the diagram are examples.

3.1 TRACE RECORDING

We represent a trace as $\tau = (G, X, Y, E, D)$. G records invocation dependencies, X and Y store input and output tokens, E stores optional logical expert selections, and D stores tool durations. The experiment separately specifies an arrival policy A, using either task release times or a fixed number of concurrent tasks. To capture dependencies correctly, the recorder obtains them from the application or orchestration runtime rather than inferring them from execution timestamps. The graph and recorded inputs also capture context edits, retries, branch decisions, and tool results.

Each invocation has a stable identifier, with metadata describing the model, tokenizer, prompt serialization, and sampling settings. Inputs are stored as token IDs to avoid changes introduced by decoding and re-tokenizing text. The recorder also retains termination reasons and, when needed for validation or application replay, tool names, arguments, and results. During replay, the driver executes the recorded graph directly instead of asking the agent to choose its actions again. As a result, tool-delay emulation does not repeat the external side effects of trace collection.

Replay fixes reuse opportunities and logical work, not the resulting cache hits, expert placement, or communication costs. Those remain effects of the serving configuration under evaluation.

3.2 MULTI-LEVEL TRACE REPLAY

Token-wise replay. For invocation r, the driver supplies input X_r and the recorded output sequence $Y_r = (y_1, \ldots, y_m)$. The model performs prefill to produce the first output token and then decodes autoregressively. At output position j, the replay hook commits y_j instead of the sampled token. The hook replaces the token before updating the next-step input, request history, and token-dependent state. Changing only the returned text would not reproduce the model workload. Replay also follows the recorded stopping position, overriding conflicting EOS and stop rules while respecting the target's context limit. Unsupported histories or decoding constraints are rejected rather than silently truncated.

Our design retains the forward pass, language-model head, and native sampling computation before replacing the selected token. It therefore measures autoregressive execution rather than returning stored responses or processing all output tokens in one prefill pass. Invocation IDs and output positions locate recorded tokens after batch reordering.

MoE router replay. Token-wise replay fixes the token sequence and recorded agent trajectory. However, when benchmarking MoE optimizations, even small differences in expert selections can affect the comparison. Identical tokens do not guarantee identical routing decisions, as numerical differences across execution configurations can change router scores (Ma et al., 2025). AgentReplay therefore provides optional router replay for comparisons that require the same expert workload. For top-k routing, router replay fixes the logical expert IDs $E_{r,p,\ell}$ at token position p and layer ℓ of invocation r. The design retains router-score and native selection computation, then substitutes the recorded IDs before logical-to-physical expert mapping and dispatch. The experts execute normally, while their placement and communication remain system decisions. Fixing expert IDs does not require identical router scores or hidden states.

Tool-duration replay. Tool-time variation changes subsequent arrivals and batch composition even when token sequences are fixed. For the recorded graph, the replay driver releases each request after its required predecessors complete. For a tool invoked at time t with recorded invocation-to-result duration d, the emulator makes its result available after a nonblocking wait until $t + d$, subject to timer and scheduling error. Independent tools wait asynchronously, while a join waits for all required predecessors. Faster LLM execution can therefore release later requests earlier, rather than preserving their original absolute timestamps.

Many serving optimizations target GPU execution, while asynchronous scheduling overlaps serving-side CPU work with GPU computation (The SGLang Team, 2024). When tool execution and its resource contention are outside the optimization scope, duration replay removes tool-time variation while retaining the waits and dependencies seen by the serving engine. The serving engine still executes its own CPU scheduling and GPU computation normally. This mode does not reproduce tool CPU work, I/O contention, or load-dependent service latency. For systems that jointly optimize retrieval or other application stages with LLM generation (Hu et al., 2025; Jin et al., 2025; Tan et al., 2025; Hu et al., 2026), users should disable tool-duration replay and execute the tools to measure these interactions.

3.3 LONG-HORIZON EVALUATION WITH SMALLER MODELS

Resource constraints often motivate evaluation with smaller models, but their live executions may terminate early or enter repetitive tool loops. These behaviors distort the context growth and reuse patterns of the intended long-horizon workload. To address this problem, AgentReplay separates trajectory generation from performance evaluation. A capable model first generates long trajectories, recording their prompts, responses, and tool dependencies. A compatible smaller model then executes these trajectories through normal forward passes. We compare serving configurations on this smaller model using the same traces, exposing context growth and token reuse that it may not reach through live generation.

Compatible tokenizers, special tokens, and prompt serialization allow direct token replay. Otherwise, AgentReplay retokenizes the recorded messages and tool observations for the target, fixing that sequence across target-system comparisons. This preserves content and dependencies rather than source token IDs or counts, subject to the target's context limit. Expert IDs are architecture-specific, so router replay requires target-side route recording.

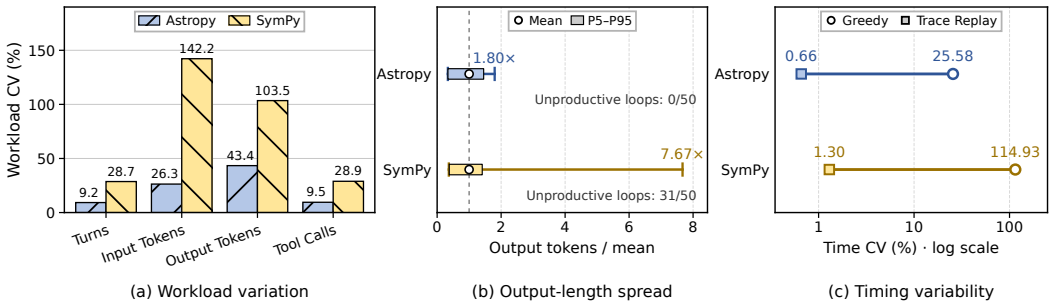

Figure 3: Greedy decoding changes the realized workload. (a) Run-level CVs for turns, input/output tokens, and tool calls. (b) Output tokens relative to the task mean; annotations report runs trapped in unproductive tool-calling or generation loops. Circles mark the mean, boxes span P5–P95, and whiskers show extrema. (c) CV of cumulative model-plus-tool time under greedy execution and trace replay (log scale). Each condition contains 50 complete runs.

4 Experiments

Setup and metrics. We evaluate Qwen3.6-35B-A3B on Astropy and SymPy agent tasks using vLLM. Sections 4.1 and 4.3 measure complete multi-turn runs, while Section 4.2 measures batch trajectories. For workload quantity w, we aggregate all requests in run i into w_i and report $\mathrm{CV}(w) = s(w_1, \ldots, w_n)/\bar{w}$, where s is the sample standard deviation. Thus, turns count LLM requests, and cumulative input tokens include context resubmitted across turns. Model time sums request durations from first scheduling to the final token, and cumulative time adds tool durations. Each complete run is one statistical sample, rather than treating correlated requests as independent observations. Time summaries use mean $\pm$ standard deviation.

4.1 Does Greedy Decoding Preserve the Workload?

Greedy decoding changes the realized workload. We repeat each task 50 times with unchanged greedy-decoding settings. Figure 3(a) shows substantial variation despite fixing the initial task and decoding configuration. Astropy has CVs of 9.24% in turns, 26.33% in cumulative input tokens, and 43.38% in output tokens. SymPy varies more, with corresponding CVs of 28.66%, 142.21%, and 103.49%. Tool-call CVs of 9.51% and 28.91% further reflect changing control flow.

Divergence produces long-tailed episodes. In Figure 3(b), the largest Astropy and SymPy runs produce 1.80× and 7.67× their task's mean output tokens. Among the 50 SymPy runs, 31 become trapped in unproductive cycles of repeated tool calls and text generation, compared with none of the Astropy runs. Together with the long output tail, these cases show that greedy execution can repeatedly generate work without making progress toward task completion.

Trace replay stabilizes repeated measurement. Token-and-tool trace replay reduces cumulative-time CV from 25.58% to 0.66% on Astropy and from 114.93% to 1.30% on SymPy (Figure 3(c)). The reductions, computed before rounding, are 38.9× and 88.7×. They measure repeatability rather than speedup because greedy runs and replay may contain different realized work. Fixed tool durations also contribute to mean cumulative time without adding variance.

4.2 Does Length-Wise Replay Preserve MoE Performance?

Matching lengths changes the expert workload. Length-wise replay preserves input and output lengths but substitutes random tokens, whereas token-wise replay executes the recorded tokens. Random tokens can concentrate routing on a subset of experts (Ma et al., 2026; Abramovich et al., 2026). Figure 4 compares layers 0 and 39 using a shared length-wise frequency ordering, so paired bars always represent the same expert. Many experts favored by length-wise replay receive little traffic under token-wise replay. Across all 40 layers, the top 32 length-wise experts receive 77.7% of prefill assignments and 56.8% of decode assignments on average. The same experts receive only

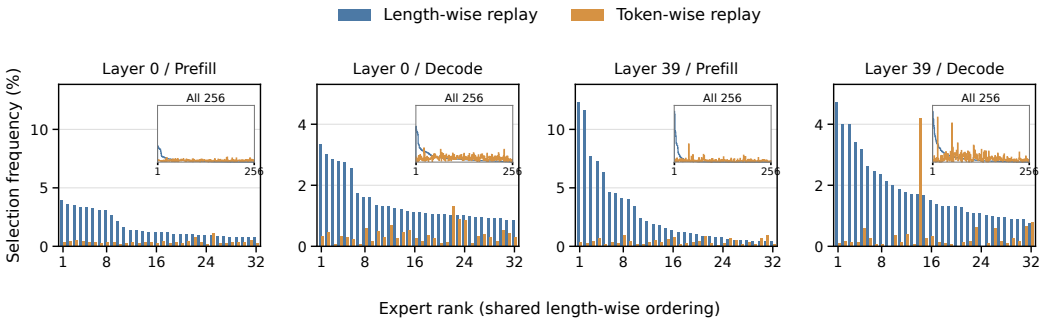

Figure 4: Expert selection under length-wise and token-wise replay. Panels show prefill and decode for layers 0 and 39. Experts are sorted by length-wise frequency within each layer and phase, with the same ordering for both modes. Main panels show the top 32 experts and insets show all 256. Frequencies sum to 100% over all experts in each layer. Vertical scales match between layers for each phase.

13.8% and 17.1% under token-wise replay. Even its own top 32 experts account for only 36.9% and 43.9% of token-wise assignments. Thus, random tokens change both concentration and hotspot identities.

We examine the resulting costs across batch sizes 1–32 with expert parallelism (EP), tensor parallelism (TP), and expert offloading. All times cover a complete batch trajectory, with prefill and decode reported separately. Kernel totals include only `fused_moe_kernel` across 40 layers and exclude routing, activation, communication, and warmup. Decode contains 188 steps. Full-phase timings are measured separately without profiling. Each ratio uses token-wise replay at the same batch size, phase, and configuration as its reference.

Expert concentration creates a prefill straggler under EP. We enable EP=2 with TP=2 and fix a placement that co-locates length-wise hot experts on rank 0. Figure 5(a) normalizes both ranks by the maximum token-wise rank total. The length-wise prefill rank-time ratio increases from 1.92× to 3.80×, while token-wise replay remains within 1.04–1.08×. At batch 32, the sum across ranks differs by less than 0.4%, but the larger length-wise rank total is 1.54× the token-wise reference. The full prefill phase takes 1.18× as long (Figure 5(b)). This result exposes the interaction between token content and expert placement.

A smaller expert working set underestimates TP decode cost. With EP disabled and TP=2, we compare the maximum of the two ranks' cumulative decode kernel times (Figure 5(c)). The two modes are nearly identical at batches 1–4, but length-wise time falls to 0.91× and 0.84× the token-wise reference at batches 16 and 32. Concentrating tokens on fewer experts can reduce distinct weight accesses and increase reuse in memory-bound decode GEMMs. At batch 1, both workloads select eight experts per token, leaving little scope for within-batch reuse. Counters from the offload experiment support the working-set explanation: at batch 32, length-wise replay touches 94.53 distinct experts per layer and decode step, compared with 117.27 for token-wise replay. These counters characterize access patterns but do not directly establish the TP kernels' bandwidth utilization.

Offloading amplifies sensitivity to token content. With EP disabled and TP=1, we place 128 of the 256 experts per layer in CPU DRAM. GPU kernels access their weights directly through UVA. Under random offloading, length-wise decode takes 0.90× and 0.82× the token-wise time at batches 16 and 32 (Figure 6). Batches 1–8 show small differences in the opposite direction. At batch 32, CPU-resident experts receive similar assignment shares, 51.50% for length-wise and 50.19% for token-wise replay, despite the different working-set sizes. Assignment share alone therefore does not characterize the memory-access workload. Offloading 128 length-wise hot experts widens the discrepancy: length-wise prefill and decode take 0.39× and 0.63× their respective token-wise times at batch 32. Together, these experiments show that matching lengths can overestimate EP prefill cost while underestimating TP decode and offloaded execution costs.

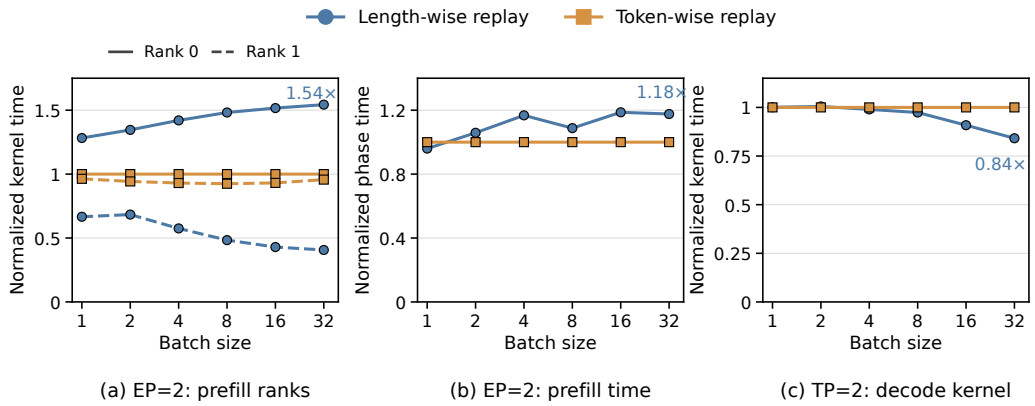

Figure 5: MoE execution times over a complete batch trajectory. (a) EP=2 prefill kernel totals per rank, normalized by the maximum token-wise rank total. Solid and dashed lines denote ranks 0 and 1. (b) Full prefill phase time with EP=2. (c) Maximum-rank decode kernel time with TP=2 and EP disabled. Token-wise replay is the reference in all panels.

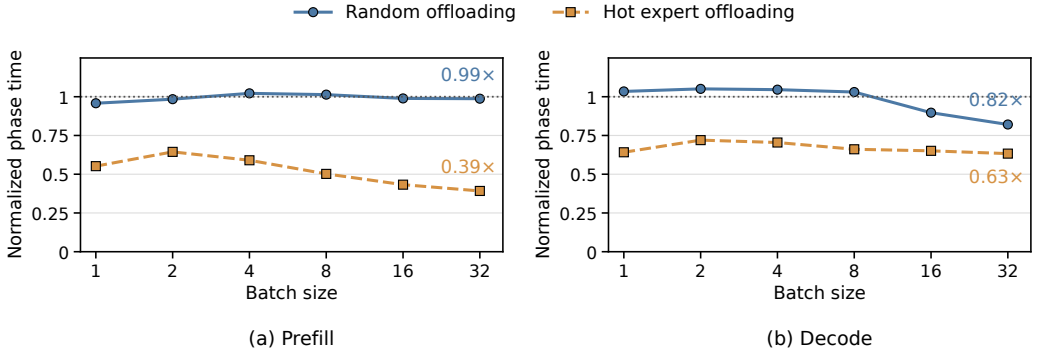

Figure 6: Offloaded execution with 128 of 256 experts per layer in CPU DRAM (TP=1, EP disabled). Curves report length-wise phase time divided by token-wise phase time for the same placement and batch size. Random offloading selects experts randomly, whereas hot expert offloading selects those favored by length-wise replay. The dotted line denotes equality.

4.3 Can a Smaller Model Replay a Capable Model's Workload?

Natural execution can miss the intended serving regime. The SymPy runs in Section 4.1 frequently exit early or become trapped in unproductive cycles of tool calling and text generation. Early exits omit the long contexts and KV-cache residence times that a long-horizon benchmark aims to exercise. These non-progressing cycles instead overrepresent a degenerate control-flow pattern. Both are valid agent behaviors, but their mixture changes the serving workload across repetitions and confounds comparisons of scheduling or cache policies.

A capable-model trace provides a fixed target workload. We collect a completed trajectory for SWE-bench instance `sympy__sympy-12419` using the DeepSeek-V4-Pro API and Mini-SWE-Agent, then replay it on Qwen3.6-35B-A3B. Because the tokenizers differ, replay preserves messages, tool observations, and invocation dependencies, then fixes the resulting target-token sequence. All 132 recorded turns and tool calls are retained, without requiring source and target token counts to match.

Natural execution takes 258.74 ± 297.37 seconds with a CV of 114.93%, while cross-model replay with fixed tool delays takes 261.21 ± 1.17 seconds with a CV of 0.45%. The CV decreases by $257.1\times$ using unrounded values, with a similar mean duration. This result demonstrates repeatable execution

of a completed long-horizon trace on the target, rather than autonomous task success or speedup. Representativeness beyond this single trajectory requires a broader trace collection.

5 Related Work

Nondeterminism in LLM inference. Greedy decoding removes sampling randomness, but it does not make model execution invariant to the serving configuration. Changes in batch shape, hardware, or parallel reduction order can introduce numerical differences that flip an argmax decision. Autoregressive decoding then propagates one changed token into a different continuation (Yuan et al., 2026; He & Thinking Machines Lab, 2025). Batch-invariant kernels (He & Thinking Machines Lab, 2025; Qiang et al., 2026; Zhang et al., 2025) remove this source of variation by enforcing a consistent computation. These techniques improve output reproducibility, but they do not replace controlled workload replay for serving-system evaluation. Requiring deterministic kernels constrains the kernels, parallelization strategies, and routing implementations available to an experiment. AgentReplay instead fixes selected workload properties while each system retains its native execution path. It does not require bitwise-identical activations or logits.

Agent traces and replay. TraceLab (Zhu et al., 2026) collects real-world coding-agent sessions and characterizes their long contexts, repeated tool use, tool-latency tails, and prefix-cache behavior. Its traces describe realistic workloads, but its goal is workload characterization rather than replaying model execution as a controlled serving experiment. AgentRR (Feng et al., 2025) records agent interactions and summarizes successful traces into reusable experiences that guide later agent behavior. This abstraction targets reliability, reuse, and large–small model collaboration. Because it may change subsequent execution, it does not define an identical serving workload.

Concurrent work XPerf (Wang et al., 2026) records input and output token IDs, invocation dependencies, and tool durations for agentic serving benchmarks and also replays accepted-token counts for speculative decoding. Both XPerf and AgentReplay use recorded trajectories to prevent live agent variation from dominating serving comparisons. Our study focuses on the additional distinction between length, token, route, and tool-delay controls, and on executing a capable model's trajectory with a compatible smaller model for long-horizon experiments.

MoE router replay. Rollout Routing Replay (Ma et al., 2025) records inference-time routing distributions and reuses them in the training engine to reduce rollout–training mismatch during MoE reinforcement learning. Megatron-LM (NVIDIA, 2026) similarly supports loading and enforcing recorded expert assignments for determinism, debugging, and profiling in training and inference. These mechanisms establish that expert selections can be recorded and enforced, but they target training consistency or isolated profiling rather than comparative serving benchmarks.

6 Discussion and Conclusion

Scope. Fixed-trace comparisons measure the cost of executing recorded behavior. They complement live evaluation when optimizations change output quality, agent decisions, or tool-resource demand. For example, replaying a recorded response cannot establish that quantization or approximate KV compression preserves task quality. Speculative decoding requires separate treatment of proposals, verification, and acceptance, which our autoregressive evaluation does not cover. Since replay fixes trajectories during measurement, its coverage still depends on collecting diverse, representative traces.

Conclusion. Agent serving benchmarks need to distinguish execution improvements from changes in the work being executed. AgentReplay controls token histories, optional logical MoE assignments, and tool delays while preserving model execution and system scheduling freedom. Our experiments show that greedy decoding changes the realized workload and that random-token replay distorts MoE expert access patterns and execution costs. A capable-model trace also provides a repeatable long-horizon workload on a smaller target model. These results support token-wise replay as a basis for controlled serving comparisons.

AI USE STATEMENT

The authors used generative AI tools to edit the manuscript, review literature, develop the experimental protocol, and write figure-generation code. The authors are responsible for the accuracy of all claims, citations, and results.

REFERENCES

Reyna Abhyankar, Zijian He, Vikranth Srivatsa, Hao Zhang, and Yiying Zhang. Infercept: Efficient intercept support for augmented large language model inference. *arXiv preprint arXiv:2402.01869*, 2024.

Talor Abramovich, Maor Ashkenazi, Izzy Putterman, Benjamin Chislett, Tiyasa Mitra, Bita Darvish Rouhani, Ran Zilberstein, and Yonatan Geifman. SPEED-Bench: A unified and diverse benchmark for speculative decoding. *arXiv preprint arXiv:2604.09557*, 2026.

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.

Qiaoling Chen, Zhisheng Ye, Tian Tang, Peng Sun, Boyu Tian, Guoteng Wang, Shenggui Li, Yonggang Wen, Zhenhua Han, and Tianwei Zhang. Concur: High-throughput agentic batch inference of llm via congestion-based concurrency control. *arXiv preprint arXiv:2601.22705*, 2026.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35: 16344–16359, 2022.

DeepSeek-AI. Expert parallelism load balancer (EPLB). `https://github.com/deepseek-ai/eplb`, 2025.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Erhang Li, Fangqi Zhou, Fangyun Lin, Fucong Dai, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoran Wei, Haowei Zhang, Haowen Luo, Haozhe Ji, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, Jialiang Huang, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jingchang Chen, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jinhua Zhu, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexin Huang, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Liang Zhao, Liangsheng Yin, Lihua Guo, Lingxiao Luo, Linwang Ma, Litong Wang, Liyue Zhang, M. S. Di, M. Y Xu, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Panpan Huang, Peixin Cong, Peiyi Wang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, S. H. Liu, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Songyang Zhou, Tao Ni, Tao Yun, Tian Pei, Tian Ye, Tianyuan Yue, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjun Gao, Wentao Zhang, Xi Gao, Xiangwen Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyuan Li, Xu Chen, Xuecheng Su, Xuehai Pan, Xuheng Lin, Xuwei Fu, Y. Q. Wang, Yang Zhang, Yanhong Xu, Yanru Ma, Yao Li, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yiliang Xiong, Ying He, Ying Zhou, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuduan Wang, Yue Gong, Yuhan Wu, Yuheng Zou, Yukun Li, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehua Zhao, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen

Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhiyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Zizheng Pan, Zongqing Yao, Bei Feng, Hui Li, J. L. Cai, Jiaqi Ni, Lei Xu, Meng Li, Ning Tian, R. J. Chen, R. L. Jin, S. S. Li, Shuang Zhou, Tianyu Sun, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xinnan Song, Xinyi Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, Dongjie Ji, Jian Liang, Jianzhong Guo, Jin Chen, Leyi Xia, Miaojun Wang, Mingming Li, Peng Zhang, Ruyi Chen, Shangmian Sun, Shaoqing Wu, Shengfeng Ye, T. Wang, W. L. Xiao, Wei An, Xianzu Wang, Xiaowen Sun, Xiaoxiang Wang, Ying Tang, Yukun Zha, Zekai Zhang, Zhe Ju, Zhen Zhang, and Zihua Qu. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. Flex attention: A programming model for generating optimized attention kernels. *arXiv preprint arXiv:2412.05496*, 2(3):4, 2024.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Erhu Feng, Wenbo Zhou, Zibin Liu, Le Chen, Yunpeng Dong, Cheng Zhang, Yisheng Zhao, Dong Du, Zhichao Hua, Yubin Xia, and Haibo Chen. Get experience from practice: LLM agents with record & replay. *arXiv preprint arXiv:2505.17716*, 2025.

Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. {Cost-Efficient} large language model serving for multi-turn conversations with {CachedAttention}. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pp. 111–126, 2024.

Horace He and Thinking Machines Lab. Defeating nondeterminism in llm inference. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tm1.20250910. https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/.

Zhengding Hu, Vibha Murthy, Zaifeng Pan, Wanlu Li, Xiaoyi Fang, Yufei Ding, and Yuke Wang. Hedrarag: Co-optimizing generation and retrieval for heterogeneous rag workflows. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, pp. 623–638, 2025.

Zhengding Hu, Zaifeng Pan, Prabhleen Kaur, Vibha Murthy, Zhongkai Yu, Yue Guan, Zhen Wang, Steven Swanson, and Yufei Ding. Pancake: Hierarchical memory system for multi-agent llm serving. *arXiv preprint arXiv:2602.21477*, 2026.

Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models. *arXiv preprint arXiv:2309.14509*, 2023.

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, 2025.

Hao Kang, Ziyang Li, Xinyu Yang, Weili Xu, 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.

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, Qiuyang Mang, Runyuan He, Qizheng Zhang, Huanzhi Mao, Xiaokun Chen, Alvin Cheung, Joseph E. Gonzalez, and Ion Stoica. Continuum: Efficient and robust multi-turn LLM agent scheduling with KV cache time-to-live. *arXiv preprint arXiv:2511.02230*, 2025.

Hao Liu, Matei Zaharia, and Pieter Abbeel. Ringattention with blockwise transformers for near-infinite context. In *International Conference on Learning Representations*, volume 2024, pp. 3992–4008, 2024.

Yuhan Liu, Yihua Cheng, Jiayi Yao, Yuwei An, Xiaokun Chen, Shaoting Feng, Yuyang Huang, Samuel Shen, Rui Zhang, Kuntai Du, and Junchen Jiang. Lmcache: An efficient kv cache layer for enterprise-scale llm inference. *arXiv preprint arXiv:2510.09665*, 2025.

Bole Ma, Jan Eitzinger, Harald Koestler, and Gerhard Wellein. Diagnosing overhead in dispatch operations: Cross-architecture observatory. *arXiv preprint arXiv:2605.20982*, 2026.

Wenhan Ma, Hailin Zhang, Liang Zhao, Yifan Song, Yudong Wang, Zhifang Sui, and Fuli Luo. Stabilizing MoE reinforcement learning by aligning training and inference routers. *arXiv preprint arXiv:2510.11370*, 2025.

NVIDIA. Megatron-LM: Design document: MoE router replay feature. `https://github.com/NVIDIA/Megatron-LM/blob/main/docs/api-guide/router_replay.md`, 2026.

Zaifeng Pan, Yitong Ding, Yue Guan, Zheng Wang, Zhongkai Yu, Xulong Tang, Yida Wang, and Yufei Ding. Fasttree: Optimizing attention kernel and runtime for tree-structured llm inference. In *Proceedings of Machine Learning and Systems*, 2025a.

Zaifeng Pan, Ajjkumar Patel, Zhengding Hu, Yipeng Shen, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. Kvflow: Efficient prefix caching for accelerating llm-based multi-agent workflows. *arXiv preprint arXiv:2507.07400*, 2025b.

Zaifeng Pan, Yipeng Shen, Zhengding Hu, Zhuang Wang, Aninda Manocha, Zheng Wang, Zhongkai Yu, Yue Guan, and Yufei Ding. Scalesim: Serving large-scale multi-agent simulation with invocation distance-based memory management. *arXiv preprint arXiv:2601.21473*, 2026a.

Zaifeng Pan, Qianxu Wang, Zhengding Hu, Chang Chen, Yue Guan, Yanbo Zhou, Steven Swanson, and Yufei Ding. Smoothagent: Efficient long-horizon LLM-based agent serving with lookahead context engineering. *arXiv preprint arXiv:2607.00151*, 2026b.

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.

Xinwei Qiang, Hongmin Chen, Shixuan Sun, Jingwen Leng, Xin Liu, and Minyi Guo. Dash: Deterministic attention scheduling for high-throughput reproducible llm training. In *International Conference on Learning Representations*, volume 2026, pp. 144643–144660, 2026.

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.

Rya Sanovar, Srikant Bharadwaj, Renee St Amant, Victor Rühle, and Saravan Rajmohan. Leanattention: Hardware-aware scalable attention mechanism for the decode-phase of transformers. *Proceedings of Machine Learning and Systems*, 7, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Xin Tan, Yimin Jiang, Yitao Yang, and Hong Xu. Towards end-to-end optimization of llm-based applications with ayo. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 1302–1316, 2025.

The SGLang Team. SGLang v0.4: Zero-overhead batch scheduler, cache-aware load balancer, faster structured outputs. https://www.lmsys.org/blog/2024-12-04-sglang-v0-4/, 2024.

Michael Wang, Yikang Yue, Shaobo Li, Yirui Eric Zhou, Chen Wang, and Jian Huang. Benchmarking llm serving systems for agentic ai workloads with xperf. *arXiv preprint arXiv:2608.20370*, 2026.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun 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 *ICLR*, 2025.

Zhiqiang Xie, Ziyi Xu, Mark Zhao, Yuwei An, Vikram Sharma Mailthody, Scott Mahlke, Michael Garland, and Christos Kozyrakis. Strata: Hierarchical context caching for long context language model serving. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*, pp. 1–16, 2026.

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. *arXiv preprint arXiv:2505.09388*, 2025.

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. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Jinghan Yao, Quentin Anthony, Aamir Shafi, Hari Subramoni, and Dhabaleswar K DK Panda. Exploiting inter-layer expert affinity for accelerating mixture-of-experts model inference. In *2024 IEEE International parallel and distributed processing symposium (IPDPS)*, pp. 915–925. IEEE, 2024.

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 (ICLR)*, 2023.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. Flashinfer: Efficient and customizable attention engine for llm inference serving. In *Eighth Conference on Machine Learning and Systems*, 2025.

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.

Zhongkai Yu, Yue Guan, Zihao Yu, Chenyang Zhou, Zhengding Hu, Shuyi Pei, Yangwook Kang, Yufei Ding, and Po-An Tsai. Patterns behind chaos: Forecasting data movement for efficient large-scale moe llm inference. In *2026 ACM/IEEE 53rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 68–84. IEEE, 2026.

Jiayi Yuan, Hao Li, Xinheng Ding, Wenya Xie, Yu-Jhe Li, Wentian Zhao, Kun Wan, Jing Shi, Xia Hu, and Zirui Liu. Understanding and mitigating numerical sources of nondeterminism in llm inference. *Advances in Neural Information Processing Systems*, 38:169819–169851, 2026.

Ziyang Zhang, Xinheng Ding, Jiayi Yuan, Rixin Liu, Huizi Mao, Jiarong Xing, and Zirui Liu. Deterministic inference across tensor parallel sizes that eliminates training-inference mismatch. *arXiv preprint arXiv:2511.17826*, 2025.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. *Advances in Neural Information Processing Systems*, 37:62557–62583, 2024a.

Zhen Zheng, Xin Ji, Taosong Fang, Fanghao Zhou, Chuanjie Liu, and Gang Peng. Batchllm: Optimizing large batched llm inference with global prefix sharing and throughput-oriented token batching. *arXiv preprint arXiv:2412.03594*, 2024b.

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.

Kan Zhu, Mathew Jacob, Chenxi Ma, Yi Pan, Stephanie Wang, Arvind Krishnamurthy, and Baris Kasikci. TraceLab: Characterizing coding agent workloads for LLM serving. *arXiv preprint arXiv:2606.30560*, 2026.

Ruidong Zhu, Ziheng Jiang, Chao Jin, Peng Wu, Cesar A Stuardo, Dongyang Wang, Xinlei Zhang, Huaping Zhou, Haoran Wei, Yang Cheng, et al. Megascale-infer: Serving mixture-of-experts at scale with disaggregated expert parallelism. *arXiv preprint arXiv:2504.02263*, 2025.