ThunderAgent: A Fast, Simple, and Program-Aware Agentic Inference System

Hao Kang*[1], Ziyang Li*[2], Xinyu Yang*[3], Weili Xu†[4], Yinfang Chen[4], Junxiong Wang[5], Beidi Chen[3], Tushar Krishna[1], Chenfeng Xu[5], and Simran Arora[5]

[1]Georgia Institute of Technology
[2]Individual Researcher
[3]Carnegie Mellon University
[4]University of Illinois Urbana-Champaign
[5]Together AI

Code Blog W Wiki

Abstract

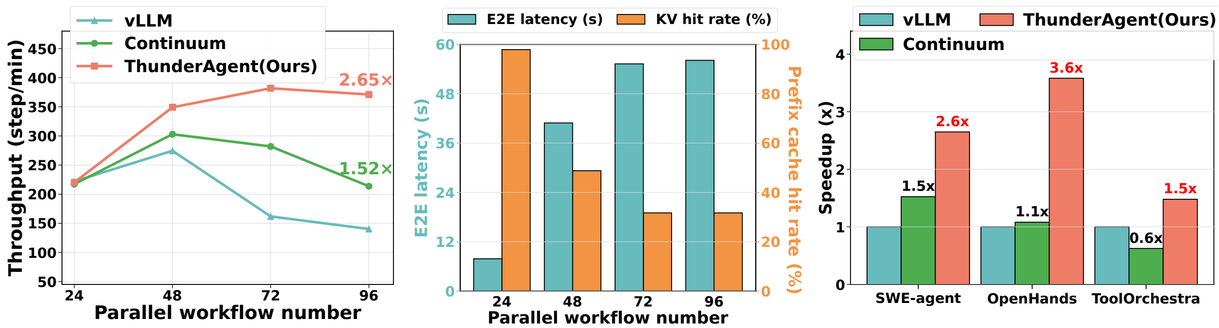
Large language models (LLMs) are now used to power complex multi-turn agentic workflows. Existing systems run agentic inference by loosely assembling isolated components: an LLM inference engine (e.g., vLLM) and a tool orchestrator (e.g., Kubernetes). Although agentic workflows involve multiple LLM and tool requests, these systems schedule and allocate resources separately on a per-request basis, without end-to-end knowledge of the workflow. This leads to sub-optimal management of KV cache and tool execution environments. To address the challenges, we propose ThunderAgent, a fast, simple, and program-aware agentic inference system. We first abstract agentic workflows as *LLM Programs*, enabling a unified view of heterogeneous resources, including KV caches, system states, and external tool assets such as disk memory and network ports. Built upon this abstraction, ThunderAgent introduces a program-aware scheduler and a tool resource manager designed to maximize KV cache hit rates, mitigate memory imbalances, and enable asynchronous environment preparation. Evaluations across coding, routing, and scientific discovery agents demonstrate that ThunderAgent achieves **1.5-3.6**× throughput improvements in serving, **1.8-3.9**× in RL rollout, and up to **4.2**× disk memory savings compared to state-of-the-art inference systems. To facilitate reproducibility and support future development, we open-source the system and kernel implementations of the whole ThunderAgent at: `https://github.com/HaoKang-Timmy/ThunderAgent`.

1 Introduction

Recent advances in language models have expanded their use beyond basic chatbots to complex agents [20, 26]. These agents address real-world problems in domains such as coding [7, 9] and computer-use [2, 24] by interleaving long reasoning with external tool calls (e.g., compilers, retrievers), often operating as autonomous systems that execute multi-step workflows without real-time human intervention. However, the throughput of modern inference systems degrades as the number of agentic requests being processed increases (Figure 1a). Meanwhile, rollout accounts for over **70%** of the total wall-clock time in reinforcement learning (RL) [4, 17].

As agentic workflows become increasingly autonomous at scale, overall system efficiency is governed by sustained throughput rather than tail latency, whereas human-in-the-loop appilcations are often dominated by user response times. Therefore, higher throughput directly reduces serving cost by amortizing hardware over more completed workflows. Moreover, in asynchronous RL, higher rollout throughput mitigates policy lag between the parameters used for data collection and those being updated. This allows the model to learn from data with reduced staleness, improving both convergence speed and final policy quality [4, 16, 17, 32].

*Equal contribution. †Core code contributor. Correspond to hkang342@gatech.edu



(a) Throughput degradation (b) KV cache thrashing (c) Speedup from ThunderAgent

Figure 1: **Performance comparison of ThunderAgent against prior agent inference systems as the parallel workflow number (i.e., batch size) increases.** We evaluate the GLM-4.6 MoE model serving SWE-Agent on SWE-Bench Lite (Figures a and b) and SWE-Agent, OpenHands, and ToolOrchestra (Figure c) on an 8×H100 GPU cluster. Results show that: (a) Current inference systems fail to maintain high throughput at large batch sizes. (b) Throughput degradation is primarily caused by low KV cache hit rates, which increase end-to-end request latency. (c) ThunderAgent achieves high throughput compared to prior inference systems by reducing KV-cache thrashing and managing the lifecycle of tool execution resources.

However, current agentic inference systems provide sub-optimal throughput because they are loosely combined from isolated components: an off-the-shelf model inference engine(e.g., vLLM [10] or SGLang [33]) coupled with a general-purpose tool orchestrator (e.g., Kubernetes). While agentic workflows involve multiple turns of model and tool requests, these components schedule and allocate resources separately on a *per-request basis*, without end-to-end knowledge of the entire workflow. This design gives rise to to three key challenges:

1. **KV cache thrashing.** The request-aware systems prematurely evict KV cache during tool-execution intervals, without foresight into future reuse hin the agent workflow. Thus when the tool call completes, the system needs to rerun prefill to recover its whole interaction history. The re-prefill cost increases the average end-to-end latency of agent workflows by up to **7.14**× (see Figure 1b) and decreases throughput.
2. **Cross-node memory imbalance.** The request-aware engines suffer from imbalanced utilization in multi-node inference setups. Existing engines pin all requests from the same agentic workflow to a fixed node to maximize the KV cache hit rate. However, as context lengths scale rapidly and unpredictably in agent workflows, some nodes reach capacity while others remain underutilized under this routing policy.
3. **Tool lifecycle obliviousness.** The request-aware orchestrators struggle to decide when to release and prepare resources and environments required for tool execution. Thus, unused sandboxes and API servers continue to occupy critical disk space and network ports, leading to cumulative resource exhaustion and system failures. Meanwhile, agentic workflows have to wait for extremely long setup time before reasoning.

This work introduces ThunderAgent, a system that adopts an end-to-end view of the agent workflow to enable high-throughput agent inference. Our specific contributions are:

1. **Program abstraction:** We abstract agent workflow as *LLM programs*. An LLM program is a first-class scheduling unit that persists across multiple model invocations and tool executions, exposing semantic state to the runtime. A program tracks metadata for the workflow's state (i.e., reasoning or acting), total state tokens, and tool resources. This decouples scheduling from execution backends (e.g., vLLM/SGLang), enabling easy integration of new agentic worflows.
2. **Program-aware scheduler:** Based on the program abstraction, we cast agentic inference scheduling as a constrained optimization problem to minimize both the re-prefill and caching overheads, and maximize decoding throughput, subject to GPU memory capacity. We use two key mechanisms:
   (a) **State-aware pausing:** If the execution backend experiences memory-pressure, we selectively pause workflows that are currently in the tool-acting state. This helps preserve GPU memory for workflows that are in the model reasoning state and eliminate arbitrary, sub-optimal KV cache evictions.
   (b) **Dynamic migration:** We migrate agent workflows across data parallel (DP) GPU nodes to resolve

memory imbalances. We accomplish this by letting the nodes share the same workflow waiting queue, rather than enforcing that requests from a workflow are always sent to the same node.

3. **Program-aware tool resource management:** In long-horizon agentic workloads, tool environments are persistent resources whose mismanagement directly limits sustained throughput. By tracking execution dependencies, ThunderAgent overlaps I/O-intensive environment initialization with LLM reasoning. For completed workflows, we implement a lifecycle-aware garbage collector that uses program termination signals to clean up resources such as Docker sandboxes and network ports. This prevents accumulated resource leakage and ensures sustained high throughput inference.

The above contributions cannot be achieved within request-aware inference engines. Without an explicit representation of workflow state and execution dependencies, request-aware schedulers cannot distinguish temporary tool waits from terminal completion or coordinate GPU memory with external resource lifecycles.

We evaluate ThunderAgent across diverse agentic workloads. For **serving**, we evaluate the ToolOrchestra [18] as routing agent on HLE-Bench [15], SWE-Agent [27] and OpenHands [23] as coding agent on SWE-bench [9], and OpenHands as scientific discovery agent on ScienceAgentBench [3], achieving **1.48–3.58**× throughput improvements as illustrated in Figure 1c. For **RL rollouts**, we further test the coding agents on distributed GPU nodes, achieving **1.79–3.92**× improvements compared with previous SOTA.

2 Background

In this section, we provide background on the properties and existing approaches to support agentic inference.

2.1 System Properties of Current Agentic Workflows

Current agentic workflows alternate between reasoning and acting during generation. Formally, at each step t, the agent receives an observation $o_t \in \mathcal{O}$ and produces an emission $e_t = (\ell_t, a_t) \in \mathcal{L} \times \mathcal{A}$, where ℓ_t denotes a thought and a_t represents an action. We define the cumulative context at step t as $c_t = (o_1, e_1, \ldots, o_t)$, which captures the interaction history of agentic workflows. Conditioned on c_t, e_t is sampled from a policy $\pi(e_t|c_t)$.

This workflow keeps two persistent states: *(i) GPU Memory*, where the KV cache of c_t serves as the workflow's memory. As the trace grows incrementally, c_{t+1} extends c_t as a prefix, enabling theoretical near-complete KV cache reuse rates across steps. *(ii) Tool Environment*, where external resources (e.g., sandboxes or database connections) initialized at $t = 1$ must remain consistent and accessible throughout the execution.

These stateful dependencies necessitate a *program-level* view of agentic inference trajectories, thereby enabling to system to coordinate heterogeneous resources and manage state across long-running workflows. However, existing inference systems treat each thought l_t and action a_t as an independent, stateless request.

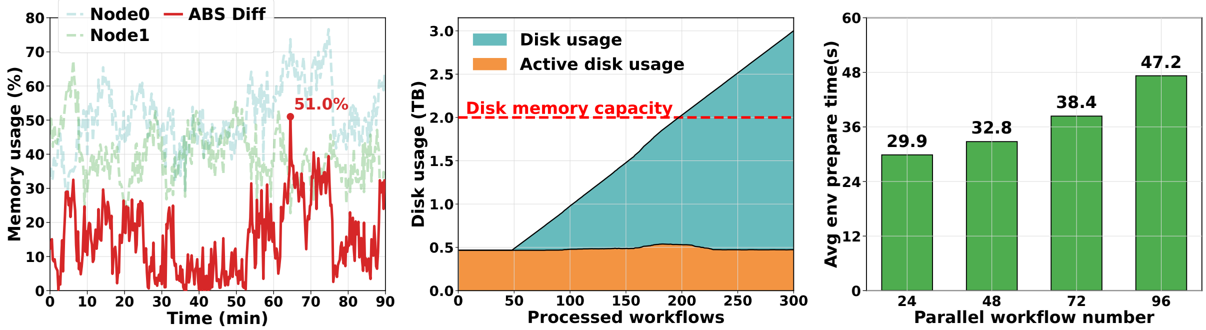
2.2 Existing Agentic Inference Systems

Prior work focuses on optimizing the individual components in agentic inference, including the LLM inference engine or tool orchestrator (Section A.1, Section A.2, but there are very few works that provide end-to-end optimization for agentic workflows across GPU, CPU, and remote resources. We review these prior systems.

Autellix models multi-turn agentic workflows as **GPU-only programs** and tracks the accumulated GPU execution time in a central process table [13]. However, it ignores workflow locality, allowing concurrent workflows to aggressively evict other's KV cache, triggering **KV cache thrashing** under heavy workloads.

Continuum is another recent serving system designed for multi-turn agentic workflows [11]. It employs a time-to-live (TTL) mechanism to pin KV caches in HBM, thereby mitigating context thrashing during tool execution. However, it fails to solve the KV cache eviction problem. The first reason is that most tools take an **unpredictable** amount of time (e.g. remote model APIs in ToolOrchestra [18], compilers in code agents, and web applications for computer use agents [35]). Such unpredictable tools trigger severe thrashing as well as stranded KV cache memory in Continuum due to incorrect TTL estimates. Moreover, once the decoding memory of the running workflow surpasses the GPU limit, the system preempts and evicts pinned KV cache as well. This leads to unavoidable thrashing and corresponding throughput degradation shown in Figure 1a.

These limitations underscore the need for a simple and fast system for agentic inference. We envision such a system as a program-aware scheduling layer for emerging agentic inference systems (e.g., Zhang et al. [31]).



(a) Memory imbalance in rollout (b) Runtime disk usage for tool envs. (c) Tool env. preparation time

Figure 2: **Demonstrations of the memory imbalance and tool resource management problems for current agentic inference systems.** We evaluate vLLM + Kubernetes on OpenHands RL rollout using the GLM 4.6 model on SWEBench-Lite with two 8×H100 GPU Nodes. The observations show: (a) Max memory imbalance can achieve 51% on 90 min rollout tests when applying vLLM KV-aware router. (b) Failure to garbage collect tool execution environments gradually causes resource usage to exceed system capacity. (c) Average tool execution environment preparation time grows fast as parallel workflow number increases.

3 Challenges in Existing Agentic Inference Systems

This section profile vLLM combined with Kubernetes as a representative baseline for multi-turn agentic inference, and synthesize its key inefficiencies. Notably, the identified limitations are not solved by replacing the inference engine (e.g., TensorRT or SGLang) or tool orchestrator, but rather require new program-aware abstractions. By default, we use GLM 4.6 model for OpenHands RL rollout on two 8×H100 GPU nodes.

3.1 KV Cache Thrashing

Agentic workflows exhibit a high theoretical KV cache reuse rate during their execution. However, in existing LLM serving systems, each step is served as an independent and stateless request. Under high concurrency, this request-level scheduling causes KV cache to be frequently evicted during tool execution to accommodate newly arriving requests, resulting in repeated eviction and reprefill, which we refer to as KV cache thrashing.

As shown in Figure 1b, this thrashing intensifies as the number of parallel workflows increases. The resulting degradation in cache hit rates triggers frequent and costly re-prefill, where the entire history must be recomputed upon tool completion. This redundancy significantly increases the end-to-end latency of each request by up to **7.14**× compared to a non-thrashing setting, leading to severe throughput degradation.

3.2 Cross-Node Memory Imbalance

Current policies for routing requests across data parallel (DP) nodes are also sub-optimal. Existing multi-turn schedulers [21, 33] greedily assign requests to the target DP nodes with the highest KV-cache locality in order to maximize cache reuse. However, this policy ignores the fact that the memory load can become imbalanced across nodes. For instance, the KV-aware router in vLLM [21] sends all requests from the same agentic workflow to the same node. Since different workflows can exhibit highly heterogeneous KV footprints and execution lifetimes, this policy often results in severe memory imbalance across nodes, with some nodes are overloaded while others remain lightly utilized. Similarly,the prefix-aware router in SGLang greedily routes workloads to nodes with matching prefixes to maximize cache hits. Since agentic system prompts are identical across workflows, this strategy will send almost all requests to the same node while leaving others idle.

As shown in Figure 2a, during a 90 minute snapshot of agentic RL rollout, the memory usage between two DP nodes diverges by **more than 20% for over 37 minutes, reaching a peak imbalance of 51%**.

3.3 Tool Lifecycle Obliviousness

Current agentic inference systems do not synchronize the external tool orchestrator's lifecycle with the LLM inference engine, resulting in silent resource wastage and latency overhead on the tool orchestrator side.

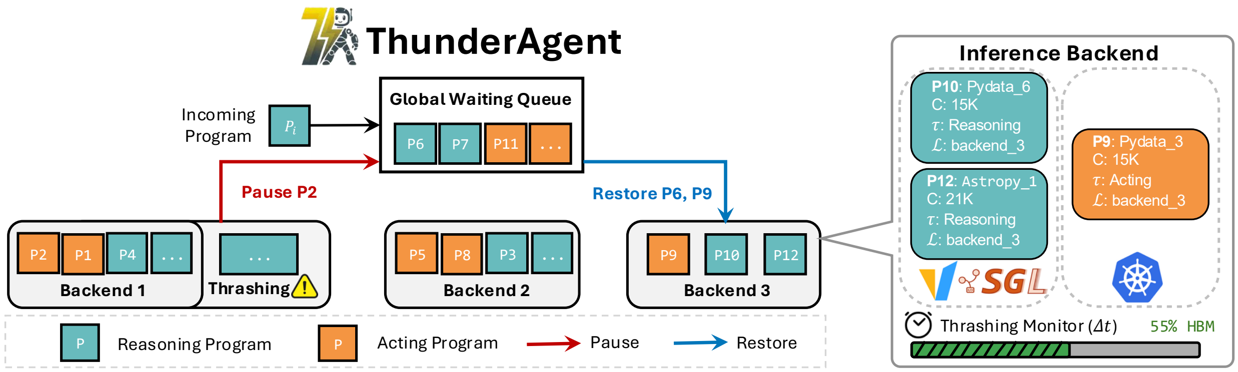


Figure 3: **An Overview of ThunderAgent.** We show the transition between scheduling states and memory management. ThunderAgent queries the state of each data parallel backend periodically every Δt time. Here, Backend #1 triggers thrashing, while Backend #3 is underutilized. The global waiting queue shared by all Backends then pauses and collects acting Program #2 back to the queue while releasing reasoning Program #6 and #9, to stop the KV-cache thrashing in Backend #1 and reduce memory imbalance of Backend #3.

Resource leakage and unused sandboxes. Figure 2b showcases that the total disk space consumption increases linearly with the number of processed workflows, eventually exceeding system capacity. This is because unused resources (e.g., Docker images of finished workloads) are not reclaimed when workflows complete. This inefficient garbage collection leads to fatal system instabilities for long-term agentic inference.

Costly environment preparation. We observed that most agentic workloads need to prepare environments before initiating the multi-turn trajectory. For example, coding agents need to pull dockers, install related packages and build repositories. Furthermore, this preparation time is costly and increases with the parallel workload number (i.e., batch size), as shown in Figure 2c. If the LLM inference engine needs to wait until the environments are fully prepared, this overhead will extend the end-to-end latency of the inference system.

4 ThunderAgent: A Program-Aware Agentic Inference System

With all findings in Section 3, we present ThunderAgent, a program-aware system for high throughput agentic inference. We model the Agentic Program in Section 4.1, which serves as our primary abstraction for scheduling. Section 4.2 formalizes a cost model to guide our system design. Built upon these foundations, we detail our KV cache scheduling policy in Section 4.3 and tool resource management strategy in Section 4.4.

Table 1: **Summary of Notations for modeling agentic programs.** Each program instance is characterized by its identity, execution phase, tool environments, resource footprint, and scheduling state in ThunderAgent.

Notation	Description
P	Agentic program instance
ID	Unique global identifier for the program
c	Number of tokens in the context
$\mathcal{T}$	Set of tool environments required by the program
$\mathcal{L}$	Backend (GPU node) placement for spatial locality
τ	Execution phase: Reasoning (**R**), Acting (**A**)
s	Scheduling status: {Active, Paused, Terminated}

4.1 Program Abstraction

The **Agentic Program** serves as a fundamental abstraction that encapsulates both the logical execution flow and the system-level dependencies of agentic workflows. Formally, we define an agentic program P as a tuple:

$$P = \langle ID, c, \mathcal{T}, \mathcal{L}, \tau, s \rangle, \tag{1}$$

where ID represents the unique global identifier. c denotes the number of tokens in the context, corresponding to the KV cache memory footprint during active execution. $\mathcal{T}$ tracks the set of tool environments used by the program, enabling garbage collection when no program requires them further. $\mathcal{L}$, τ, and s denote the node placement, execution phase, and scheduling status, respectively, facilitating program-level KV cache thrashing reduction and cross-node transferring. A metadata example is demonstrated on the right side of Figure 3.

ThunderAgent directly wraps existing LLM engines and tool orchestrators by interfacing with OpenAI-style endpoints. Program IDs allow the system to distinguish requests from different agentic workflows. We elaborate on the simplicity of integrating ThunderAgent with existing inference services in Appendix B.

4.2 Cost Model

During multi-turn agentic inference, only the resources used for active prefilling and decoding contribute to the system's effective throughput, while re-computation, used capacity, and idle caching constitute resource waste. We encompass this in a cost model for GPU resource consumption, which isolates effective costs from non-productive usage. We adopt the Space-Time Product (STP) [1] as our primary metric, defined as the integral of the memory footprint over processing time. The STP cost during a process phase is formalized as:

$$\mathrm{Cost_x} = \int_0^{t_x} M_x(t)\, dt, \tag{2}$$

where t_x is the duration of process x (e.g., prefill). Since memory usage $M_x(t)$ can be directly quantified by the KV cache token count used in LLMs, we define our cost model as the integral of token count over time.

The total cost of agentic inference comprises five distinct components: decoding, prefilling, recomputation, unused capacity, and idle caching. We explicitly distinguish incremental prefilling for tool execution results from recomputation over historical interactions, with the latter leading to significantly higher cost due to re-computing evicted KV cache over the full context. Formally, this yields the following cost decomposition:

$$\mathrm{Cost_{total}} \approx \mathrm{Cost_{decode}} + \mathrm{Cost_{prefill}} + \mathrm{Cost_{recompute}} + \mathrm{Cost_{unused}} + \mathrm{Cost_{caching}} \tag{3}$$

In this decomposition, $\mathrm{Cost_{decode}}$ and $\mathrm{Cost_{prefill}}$ represents the effective work that contributes to inference throughput. The remaining terms are wasted system overheads: $\mathrm{Cost_{recompute}}$ stems from KV cache thrashing (Section 3.1); $\mathrm{Cost_{unused}}$ reflects memory imbalance across data parallel (DP) inference backend replicas (Section 3.2); and $\mathrm{Cost_{caching}}$ accumulates while holding memory during external tool execution (Section 3.3).

4.3 Scheduling Policy

Based on the cost model above, the optimization target of our scheduling policy is to minimize the non-productive overhead components: $\mathrm{Cost_{recompute}}$, $\mathrm{Cost_{unused}}$, and $\mathrm{Cost_{caching}}$, thereby maximizing throughput.

4.3.1 Reducing Recomputation and Caching Costs via Program-Aware Waiting Queue

As identified in Section 3.1 and Figure 1b, KV cache thrashing serves as the primary bottleneck for throughput degradation. To address this limitation, the system must minimize $\mathrm{Cost_{recompute}}$ by explicitly controlling the number of active programs. ThunderAgent achieves this by introducing a program-aware waiting queue. Our system utilizes this queue to schedule program execution, determining which program should be executed in GPU versus which should be swapped out based on their token length c and execution phase τ. Here, we formalize the scheduler behavior using two primitive operations: **Restore** and **Pause**, as follows.

- **Restore.** This operation admits a program into active execution. Given a program $P = \langle ID, c, \mathcal{T}, \mathcal{L}, \tau, s \rangle$ with $s = \text{Paused}$ and $\mathcal{L} = \varnothing$, Restore($P$) assigns P to a backend $\mathcal{L}'$ with available capacity and updates

$$P \leftarrow \langle ID, c, \mathcal{T}, \mathcal{L}', \tau, \text{Active} \rangle, \tag{4}$$

- **Pause.** This operation removes a program from active execution. Given a program $P = \langle ID, c, \mathcal{T}, \mathcal{L}, \tau, s\rangle$ with $s = \text{Active}$, $\text{Pause}(P)$ unbinds P from its backend, releases its KV cache for preemption, and updates

$$P \leftarrow \langle ID, c, \mathcal{T}, \varnothing, \tau, \text{Paused}\rangle. \tag{5}$$

Building on these two operations, we next introduce our scheduling policy to minimize KV cache thrashing.

Periodic thrashing detection. The program abstraction in Section 4.1 provides us with the KV cache size of acting programs. Notably, this is unavailable in request-level systems (as in Section 3). We define the thrashing condition for a DP backend $\mathcal{L}$ as the state where program memory demand exceeds total capacity:

$$\mathrm{C}_{\text{total}} < \sum_{p\in\mathcal{L}} c_p \tag{6}$$

where $\mathrm{C}_{\text{total}}$ denotes the fixed token capacity of the KV cache pool for backend $\mathcal{L}$. During decoding, the context length c_p of agentic workflows grows rapidly, which can trigger memory thrashing *mid-execution* even without of new arrivals. Unlike baseline schedulers (e.g., Continuum) that only perform checks on whether to admit a workflow upon its arrival, we implement a **periodic monitor** that evaluates the memory usage at fixed intervals Δt, allowing preemptive detection and mitigation of memory pressure caused by context growth.

When KV cache thrashing is imminent, ThunderAgent invokes **Pause** operation to suspend active programs and free memory size $\Delta C = \sum_{p\in\mathcal{L}} c_p - \lambda_{\max} \cdot \mathrm{C}_{\text{total}}$ until the total memory usage falls below the limit $\lambda_{\max} \cdot \mathrm{C}_{\text{total}}$. Conversely, when the backend has available space, meaning $\sum_{p\in\mathcal{L}} c_p < \lambda_{\min} \cdot C_{\text{total}}$, ThunderAgent restores paused programs from the waiting queue via **Restore**, ensuring that the restored program keeps the total memory below $\lambda_{\max} \cdot \mathrm{C}_{\text{total}}$. Here, $\lambda_{\max}$ and $\lambda_{\min}$ denote the high- and low-watermarks of memory usage, respectively, together forming a hysteresis window that stabilizes our scheduling. In practice, we set both value to be 1, as the shared prompt across programs implicitly reserves sufficient memory buffer.

With this program-level periodic capacity check, ThunderAgent can guarantee that there will be no KV cache thrashing by reserving memory for active programs during the acting phase. However, the tradeoff is that when programs engage in long-running tool execution, the GPU memory occupied by acting programs is idle. To balance the cost of caching against recomputation, we incorporate a time-decay mechanism into the thrashing check that progressively discounts the effective weight of acting programs' tokens. This allows the scheduler to evict long-idling caching when memory pressure rises, rather than holding them indefinitely:

$$\mathrm{C}_{\text{total}} < \sum_{p\in\mathcal{L},\tau=\mathbf{R}} c_p + \sum_{q\in\mathcal{L},\tau=\mathbf{A}} c_q \times f(t_q) \tag{7}$$

Specifically, t_q is the tool execution time of program q in the current step. $f(t)$ is a time-decay function designed to balance $\text{Cost}_{\text{caching}}$ and $\text{Cost}_{\text{recompute}}$. By dynamically lowering the effective memory priority of acting programs over time, $f(t)$ encourages the scheduler to evict caches that remain idle. In Section E.1, we prove that when tool execution latencies satisfy the memoryless property (i.e., the remaining execution time is independent of the elapsed duration), the optimal decay function $f(t)$ takes the form of exponential decay.

Minimizing $\text{Cost}_{\text{recompute}}$ via Shortest-First Eviction. With the eviction and restoration conditions above, the remaining question in handling thrashing is to determine which subset of active programs to pause such that the recomputation cost is minimized. In this paragraph, we demonstrate that evicting programs with the smallest KV cache size yields the optimal solution, with a detailed proof provided in Section E.2.

Lemma 4.1 (Quadratic Recomputation Cost)**.** *Given a program P_i with context length c_i, the recomputation cost incurred by reprefilling its KV cache scales quadratically with c_i, i.e.,*

$$Cost_{recompute} = \int_0^{t_{\text{recompute}}} c_i(t)\, dt \propto c_i^2. \tag{8}$$

Definition 4.1 (Eviction Optimization Problem)**.** Based on Lemma 4.1, given a required memory release ΔC, the scheduler aims to select a subset S of programs to evict such that the released capacity satisfies the constraint while minimizing the total recomputation cost. This optimization problem is formulated as follows:

$$\min_S \sum_{i\in S} c_i^2 \quad \text{s.t.} \quad \sum_{i\in S} c_i \geq \Delta C. \tag{9}$$

The objective is strictly minimized by selecting smaller c_i. Thus, ThunderAgent's strategy is to greedily pause and evict programs with the **shortest context lengths**. We defer the formal proof to Appendix E.3. Based on these analyses, we employ the following scores for restoring and pausing a program in our scheduler:

$$S_{\text{restore}}(P) = \frac{1}{c_P} + \mathbb{I}(\tau = \mathbf{R}) \tag{10}$$

$$S_{\text{pause}}(P) = \frac{1}{c_P} + \mathbb{I}(\tau = \mathbf{A}) \tag{11}$$

where the indicator function $\mathbb{I}(\cdot)$ enforces strict prioritization of the program's execution state (τ) over context length. Both mechanisms follow the **shortest-first** policy to minimize recomputation cost. However, the state indicator $\mathbb{I}$ ensures that the scheduler prioritizes pausing Acting programs, thereby minimizing $\text{Cost}_{\text{caching}}$ by reclaiming cached memory, while prioritizing restore Reasoning programs to maximize $\text{Cost}_{\text{decode}} + \text{Cost}_{\text{prefill}}$.

4.3.2 Reducing Memory Imbalance via Global Program-Aware Waiting Queue

Section 3.1 and Figure 2a highlight that memory imbalance across nodes introduces significant $\text{Cost}_{\text{unused}}$, leading to unnecessary program pausing despite sufficient memory capacity from other nodes. To this end, ThunderAgent unify waiting queues of all backend replicas into a **global program-aware waiting queue**.

The key motivation of this design is that $\text{Cost}_{\text{unused}}$ arises only when paused programs remain in the waiting queue while some replicas have idle memory. Moreover, once a program is paused, its KV cache is assumed to be evicted, making its recomputation cost node-agnostic. This allows us to improve cross-node memory balance without sacrificing KV cache locality. The restore policy aligns with load balancing rather than strict KV-aware routing, enabling paused programs to be dispatched to any replica with available memory capacity. As a result, the global queue bounds the unused cost such that $\text{C}_{\text{unused}} < c_{\min} \cdot \Delta t$[1] for every node in the period of Δt, where $c_{\min}$ represents the minimum token length among paused programs. An overview of the scheduling policy and the global waiting queue in ThunderAgent is presented in Figure 3.

4.4 Tool Resource Management

Next, ThunderAgent mitigates the resource leakage and environment setup overheads detailed in Section 3.3.

Hook-based garbage collection. We implement lifecycle hooks that strictly couple the persistence of tool resources with the agentic program's scheduling status s. When a program is *Terminated*, the collector triggers an immediate teardown sequence, systematically reclaiming sandboxes, network sockets, and compute slots. The active disk usage in Figure 2b showcases that our resource management policy effectively prevents the accumulation of excessive resources, maintaining a near-constant disk memory consumption over time.

Asynchronous environment preparation. The latency involved in initializing a tool execution environment (e.g., starting a Docker container and installing dependencies) can be a bottleneck. To address this, ThunderAgent monitors the global waiting queue; when a high-priority program (high S_{restore}) approaches the restore threshold, the system asynchronously restores its execution environment before the GPU memory is allocated. This technique effectively **hides the initialization overhead**, significantly reducing end-to-end latency for tool-call heavy workloads like coding agents and science agents, as demonstrated in Figure 2c.

5 Experiments

In this section, we evaluate ThunderAgent on diverse agentic workflows, including coding, routing, and scientific research agents, and RL rollout across multiple hardware configurations ranging from RTX5090 to H100 clusters. Furthermore, we conduct extensive ablation studies to breakdown the end-to-end system runtime and to describe the system's sensitivity to the scheduler's hyperparameters, Δt and $f(t)$, in Section 5.4.

[1] Since Δt is much smaller than a program's lifetime, we ignore the impact of terminated programs within a single interval.

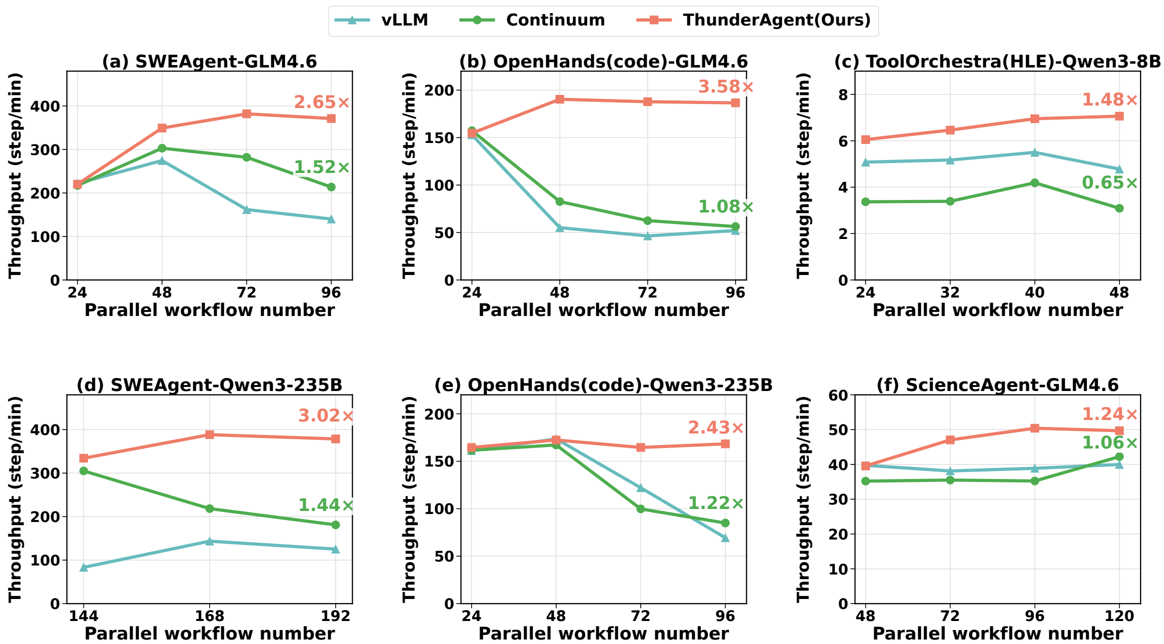


Figure 4: **Serving Evaluation Results.** ThunderAgent significantly outperforms vLLM and Continuum across three models, four agentic workflows, and three datasets. For workflows with predictable tool execution times (e.g., a, b, d, e), ThunderAgent outperforms vLLM and Continuum up to 2.43-3.56×. When workflows exhibit stochastic tool execution time, ThunderAgent still achieves the best throughput performance.

5.1 Experimental setup

Benchmarks and workflows. We evaluate ThunderAgent against diverse benchmarks and workloads:

1. **Coding agent serving.** We deploy **OpenHands** and **mini-SWEAgent** on the SWEBench-Lite [9] dataset. OpenHands represents a **heavy-initialization workflow** with an average disk footprint exceeding 10GB per sandbox, while mini-SWEAgent is a **lightweight workflow** with a minimal footprint (≈2GB).
2. **Other agent serving.** We apply ToolOrchestra on HLE [15] and OpenHands on ScienceAgentBench [3]. These workloads involve variable latencies driven by external API calls and complex scientific simulations.
3. **RL rollout.** We apply the same models, workflows, and samples for RL rollout on two 8× H100 nodes.

Models and deployments. We employ GLM-4.6 (355B) [20] and Qwen-3 (235B) [26] using both OpenHands [23] and mini-SWEAgent [27] frameworks. Models are quantized to FP8 with Tensor Parallelism (TP8) on 8×H100 nodes. For ToolOrchestra [18], we use Qwen3-8B with FP16 precision hosted on one RTX 5090. We deploy the LLM inference engine and Docker at different clusters. The LLM inference engine runs on GPU clusters hosting the models, while agent Docker environments are offloaded to a dedicated CPU cluster.

ThunderAgent configuration. We configure ThunderAgent with hyperparameters $\Delta t = 5$ and priority decay $f(t) = 2^{-t}$, defined in Section 4.3. vLLM is employed as our LLM inference engine. We use steps per minute as our throughput metric, where one step includes a reasoning and acting period of the workflow.

Baseline techniques. We compare against state-of-the-art systems with different scheduling paradigms:

- **vLLM (Inference):** A widely adopted, request-aware LLM inference engine that serves as a stateless baseline for inference performance, without incorporating any agent- or program-specific awareness.
- **Continuum (Inference):** The current SOTA system for multi-turn agentic workflows. It mitigates KV cache thrashing by predicting tool execution durations and pinning KV cache to HBM correspondingly.
- **vLLM + SGLang Gateway (Distributed Rollout):** The leading solution for large-scale distributed RL rollout. SGLang Gateway optimizes distributed inference by enhancing cross-node memory balancing and KV cache hit rates, making this combination a strong baseline for the distributed RL rollout setting.

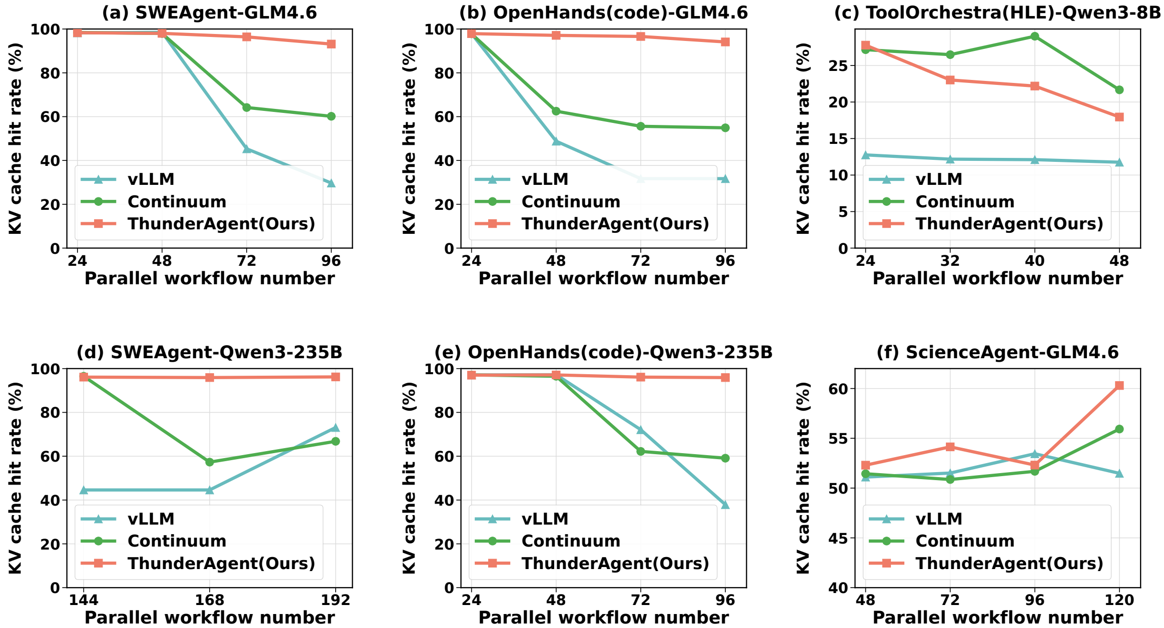


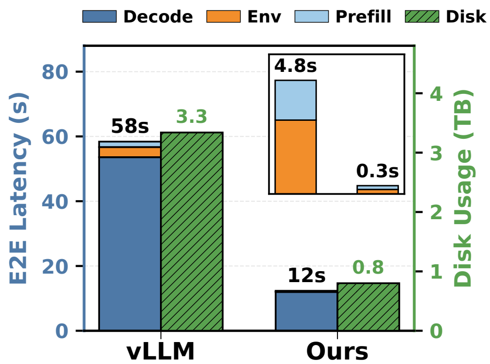
Figure 5: KV cache hit rate data statistics.

5.2 Serving Evaluation Results

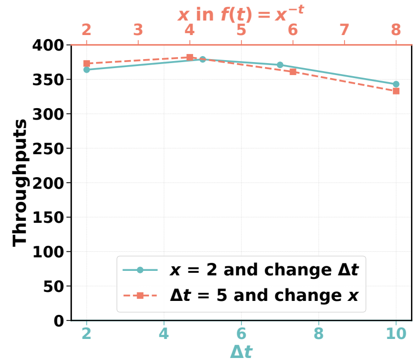
High throughput under high concurrency. As illustrated in Figure 4, ThunderAgent demonstrates superior throughput at high concurrency levels (e.g., 96 parallel programs), achieving a **1.48–3.58**× speedup over vLLM and **1.17–3.31**× speedup over Continuum across diverse base models and datasets. This gain comes from our program-aware scheduler, which maintains a near-optimal KV cache hit rate (≈100% for Mini-SWE-Bench and OpenHands, see Figure 5a,b,d,e) and enables the asynchronous preparation of environments. In contrast, Continuum suffers from performance degradation under high concurrency. As shown in Figure 5 (a, b, d, e), its KV cache hit rate drops significantly from >90% to ≈60%. This occurs because Continuum suffers from KV cache eviction among requests in different programs when there is not enough memory for ongoing requests' decoding. Consequently, active programs compete for limited memory and trigger thrashing.

Robustness performance to high concurrency. ThunderAgent maintains maximum achievable throughput even as the parallel workflow number scales beyond the GPU memory limit. As shown in Figure 4, ThunderAgent ensures that throughput remains stable with the number of parallel workflows, whereas baseline systems suffer from severe throughput collapse once the workload exceeds memory limits. In practical agentic serving, statically determining the optimal parallel workflow number to maximize utilization with limited KV-cache thrashing and caching cost is often infeasible due to the stochastic nature of agent environments and tool execution durations. ThunderAgent addresses this by automatically adapting to the maximum available capacity without manual tuning, a capability critical for robust real-world deployments.

Robustness across deterministic and stochastic tool executions. ThunderAgent outperforms baselines not only in workflows with deterministic tool patterns (Figure 4 a, b, d, e) but also under highly stochastic conditions (Figure 4 c, f). This comes from our dynamic program-aware waiting queue policy. vLLM's request-aware scheduler typically lacks reserved space for acting programs, forcing frequent re-computation. Conversely, Continuum statically reserves memory for all paused programs and mispredicts the tool execution time. These lead to severe memory underutilization during long, unpredictable tool calls. ThunderAgent navigates this trade-off via a time-decay function $f(t)$. This mechanism prioritizes retaining the KV-cache of programs with short tool calls while preemptively pausing long-running programs



(a) End-to-end Latency



(b) Ablation of Δt and f(t)

Figure 6: Ablation study of agentic RL rollout and parameter sensitivity of THUNDERAGENT.

to prevent memory underutilization. As shown in Figure 5 (c,f), although THUNDERAGENT exhibits a lower KV-cache hit rate than Continuum in stochastic settings, it achieves higher throughput by ensuring active GPU utilization.

5.3 Rollout Evaluation Results

We evaluate RL rollout using GLM-4.6 on a two-node H100 cluster (3-hour duration). Table 2 shows that THUNDERAGENT can maintain effective scalability, achieving a **1.79–3.92**× throughput increase over the vLLM + Gateway baseline, making it highly efficient for memory-intensive distributed RL workloads.

Table 2: THUNDERAGENT GLM-4.6 rollout ($N = 144$) on 2×H100 nodes.

Workflow	Serving System	Throughput
mini-SWEAgent	vLLM + Gateway	375.4
mini-SWEAgent	THUNDERAGENT	671.8 (**1.79**×)
OpenHands	vLLM + Gateway	69.1
OpenHands	THUNDERAGENT	270.8 (**3.92**×)

5.4 Ablation Study

End-to-end latency breakdown. Figure 6a decomposes the average end-to-end latency for OpenHands rollouts. The throughput gain stems primarily from significant reductions in **prefill and decode latency**. Additionally, the tool resource management (Section 4.4) contributes approximately **10%** to the latency improvement while providing **4.2**× disk memory savings. Per-step end-to-end latency are discussed in Appendix F.

Ablation on Δt and f(t). We study the sensitivity of detecting period Δt and the decaying function $f(t) = x^{-t}$. Figure 6b shows THUNDERAGENT offline serving mini-SWEAgent with GLM4.6 as base model on a single H100 node. We see that THUNDERAGENT maintains high throughput under different parameter settings, demonstrating the robustness of our method. Further increasing Δt might increase the KV-cache hit rate and thereby reduce throughput because thrashing might occur in the middle of detecting. Also, increasing x in $f(t)$ allows more aggressive eviction of acting programs, which trade recomputation costs to reduce caching costs. This will decrease throughput under mini-SWEAgent since the tool call in such a workflow is sufficient.

6 Conclusion

We introduce THUNDERAGENT, a fast and simple agentic system built on a program-level abstraction that tracks metadata throughout the entire lifecycle of each agentic workflow. THUNDERAGENT leverages the program abstraction for runtime scheduling and resource management. Specifically, THUNDERAGENT dynamically schedules program execution across GPU nodes to mitigate KV-cache thrashing and memory imbalance, while managing tool resources to prevent resource leakage. Experimental results showcase that THUNDERAGENT outperforms prior systems by **1.48–3.58**× for inference and **1.79–3.92**× for RL rollouts.

7 Acknowledgements

We are grateful to Together.ai for making this work possible. We thank Ben Athiwaratkun and Ce Zhang for assistance in developing the multibackend scheduler. We thank Wenyi Hong and Luke Huang for helpful feedback and discussions during this work.

References

[1] L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966. doi: 10.1147/sj.52.0078.

[2] Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Jang, and Zack Hui. Windows agent arena: Evaluating multi-modal os agents at scale, 2024. URL https://arxiv.org/abs/2409.08264.

[3] Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, Vishal Dey, Mingyi Xue, Frazier N. Baker, Benjamin Burns, Daniel Adu-Ampratwum, Xuhui Huang, Xia Ning, Song Gao, Yu Su, and Huan Sun. Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery, 2024. URL https://arxiv.org/abs/2410.05080.

[4] Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, Tongkai Yang, Binhang Yuan, and Yi Wu. Areal: A large-scale asynchronous reinforcement learning system for language reasoning, 2025. URL https://arxiv.org/abs/2505.24298.

[5] Wei Gao, Yuheng Zhao, Tianyuan Wu, Shaopan Xiong, Weixun Wang, Dakai An, Lunxi Cao, Dilxat Muhtar, Zichen Liu, Haizhou Zhao, et al. Rollart: Scaling agentic rl training via disaggregated infrastructure. *arXiv preprint arXiv:2512.22560*, 2025.

[6] Yiyuan He, Minxian Xu, Jingfeng Wu, Jianmin Hu, Chong Ma, Min Shen, Le Chen, Chengzhong Xu, Lin Qu, and Kejiang Ye. Banaserve: Unified kv cache and dynamic module migration for balancing disaggregated llm serving in ai infrastructure, 2025. URL https://arxiv.org/abs/2510.13223.

[7] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code, 2024. URL https://arxiv.org/abs/2403.07974.

[8] Dongfu Jiang, Yi Lu, Zhuofeng Li, Zhiheng Lyu, Ping Nie, Haozhe Wang, Alex Su, Hui Chen, Kai Zou, Chao Du, et al. Verltool: Towards holistic agentic reinforcement learning with tool use. *arXiv preprint arXiv:2509.01055*, 2025.

[9] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. URL https://arxiv.org/abs/2310.06770.

[10] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

[11] Hanchen Li, Qiuyang Mang, Runyuan He, Qizheng Zhang, Huanzhi Mao, Xiaokun Chen, Hangrui Zhou, Alvin Cheung, Joseph Gonzalez, and Ion Stoica. Continuum: Efficient and robust multi-turn llm agent scheduling with kv cache time-to-live, 2025. URL https://arxiv.org/abs/2511.02230.

[12] 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, 2025. URL https://arxiv.org/abs/2510.09665.

[13] Michael Luo, Xiaoxiang Shi, Colin Cai, Tianjun Zhang, Justin Wong, Yichuan Wang, Chi Wang, Yanping Huang, Zhifeng Chen, Joseph E. Gonzalez, and Ion Stoica. Autellix: An efficient serving engine for llm agents as general programs, 2025. URL https://arxiv.org/abs/2502.13965.

[14] Emanuel Parzen. *Stochastic processes.* SIAM, 1999.

[15] Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, Michael Choi, Anish Agrawal, Arnav Chopra, Adam Khoja, Ryan Kim, Richard Ren, Jason Hausenloy, Oliver Zhang, Mantas Mazeika, Dmitry Dodonov, Tung Nguyen, Jaeho Lee, Daron Anderson, Mikhail Doroshenko, Alun Cennyth Stokes, Mobeen Mahmood, Oleksandr Pokutnyi, Oleg Iskra, Jessica P. Wang, John-Clark Levin, Mstyslav Kazakov, Fiona Feng, Steven Y. Feng, Haoran Zhao, Michael Yu, Varun Gangal, Chelsea Zou, Zihan Wang, Serguei Popov, Robert Gerbicz, Geoff Galgon, Johannes Schmitt, Will Yeadon, Yongki Lee, Scott Sauers, Alvaro Sanchez, Fabian Giska, Marc Roth, Søren Riis, Saiteja Utpala, Noah Burns, Gashaw M. Goshu, Mohinder Maheshbhai Naiya, Chidozie Agu, Zachary Giboney, Antrell Cheatom, Francesco Fournier-Facio, Sarah-Jane Crowson, Lennart Finke, Zerui Cheng, Jennifer Zampese, Ryan G. Hoerr, Mark Nandor, Hyunwoo Park, Tim Gehrunger, Jiaqi Cai, Ben McCarty, Alexis C Garretson, Edwin Taylor, Damien Sileo, Qiuyu Ren, Usman Qazi, Lianghui Li, Jungbae Nam, John B. Wydallis, Pavel Arkhipov, Jack Wei Lun Shi, Aras Bacho, Chris G. Willcocks, Hangrui Cao, Sumeet Motwani, Emily de Oliveira Santos, Johannes Veith, Edward Vendrow, Doru Cojoc, Kengo Zenitani, Joshua Robinson, Longke Tang, Yuqi Li, Joshua Vendrow, Natanael Wildner Fraga, Vladyslav Kuchkin, Andrey Pupasov Maksimov, Pierre Marion, Denis Efremov, Jayson Lynch, Kaiqu Liang, Aleksandar Mikov, Andrew Gritsevskiy, Julien Guillod, Gözdenur Demir, Dakotah Martinez, Ben Pageler, Kevin Zhou, Saeed Soori, Ori Press, Henry Tang, Paolo Rissone, Sean R. Green, Lina Brüssel, Moon Twayana, Aymeric Dieuleveut, Joseph Marvin Imperial, Ameya Prabhu, Jinzhou Yang, Nick Crispino, Arun Rao, Dimitri Zvonkine, Gabriel Loiseau, Mikhail Kalinin, Marco Lukas, Ciprian Manolescu, Nate Stambaugh, Subrata Mishra, Tad Hogg, Carlo Bosio, Brian P Coppola, Julian Salazar, Jaehyeok Jin, Rafael Sayous, Stefan Ivanov, Philippe Schwaller, Shaipranesh Senthilkuma, Andres M Bran, Andres Algaba, Kelsey Van den Houte, Lynn Van Der Sypt, Brecht Verbeken, David Noever, Alexei Kopylov, Benjamin Myklebust, Bikun Li, Lisa Schut, Evgenii Zheltonozhskii, Qiaochu Yuan, Derek Lim, Richard Stanley, Tong Yang, John Maar, Julian Wykowski, Martí Oller, Anmol Sahu, Cesare Giulio Ardito, Yuzheng Hu, Ariel Ghislain Kemogne Kamdoum, Alvin Jin, Tobias Garcia Vilchis, Yuexuan Zu, Martin Lackner, James Koppel, Gongbo Sun, Daniil S. Antonenko, Steffi Chern, Bingchen Zhao, Pierrot Arsene, Joseph M Cavanagh, Daofeng Li, Jiawei Shen, Donato Crisostomi, Wenjin Zhang, Ali Dehghan, Sergey Ivanov, David Perrella, Nurdin Kaparov, Allen Zang, Ilia Sucholutsky, Arina Kharlamova, Daniil Orel, Vladislav Poritski, Shalev Ben-David, Zachary Berger, Parker Whitfill, Michael Foster, Daniel Munro, Linh Ho, Shankar Sivarajan, Dan Bar Hava, Aleksey Kuchkin, David Holmes, Alexandra Rodriguez-Romero, Frank Sommerhage, Anji Zhang, Richard Moat, Keith Schneider, Zakayo Kazibwe, Don Clarke, Dae Hyun Kim, Felipe Meneguitti Dias, Sara Fish, Veit Elser, Tobias Kreiman, Victor Efren Guadarrama Vilchis, Immo Klose, Ujjwala Anantheswaran, Adam Zweiger, Kaivalya Rawal, Jeffery Li, Jeremy Nguyen, Nicolas Daans, Haline Heidinger, Maksim Radionov, Václav Rozhoň, Vincent Ginis, Christian Stump, Niv Cohen, Rafał Poświata, Josef Tkadlec, Alan Goldfarb, Chenguang Wang, Piotr Padlewski, Stanislaw Barzowski, Kyle Montgomery, Ryan Stendall, Jamie Tucker-Foltz, Jack Stade, T. Ryan Rogers, Tom Goertzen, Declan Grabb, Abhishek Shukla, Alan Givré, John Arnold Ambay, Archan Sen, Muhammad Fayez Aziz, Mark H Inlow, Hao He, Ling Zhang, Younesse Kaddar, Ivar Ängquist, Yanxu Chen, Harrison K Wang, Kalyan Ramakrishnan, Elliott Thornley, Antonio Terpin, Hailey Schoelkopf, Eric Zheng, Avishy Carmi, Ethan D. L. Brown, Kelin Zhu, Max Bartolo, Richard Wheeler, Martin Stehberger, Peter Bradshaw, JP Heimonen, Kaustubh Sridhar, Ido Akov, Jennifer Sandlin, Yury Makarychev, Joanna Tam, Hieu Hoang, David M. Cunningham, Vladimir Goryachev, Demosthenes Patramanis, Michael Krause, Andrew Redenti, David Aldous, Jesyin Lai, Shannon Coleman, Jiangnan Xu, Sangwon Lee, Ilias Magoulas, Sandy Zhao, Ning Tang, Michael K. Cohen, Orr Paradise, Jan Hendrik Kirchner, Maksym Ovchynnikov, Jason O. Matos, Adithya Shenoy, Michael Wang, Yuzhou Nie, Anna Sztyber-Betley, Paolo Faraboschi, Robin Riblet, Jonathan Crozier, Shiv Halasyamani, Shreyas Verma, Prashant Joshi, Eli Meril, Ziqiao Ma, Jérémy Andréoletti, Raghav Singhal, Jacob Platnick, Volodymyr Nevirkovets, Luke Basler, Alexander Ivanov, Seri Khoury, Nils Gustafsson, Marco Piccardo, Hamid Mostaghimi, Qijia Chen, Virendra Singh, Tran Quoc Khánh, Paul Rosu, Hannah Szlyk, Zachary Brown, Himanshu Narayan, Aline Menezes, Jonathan Roberts, William Alley, Kunyang Sun, Arkil Patel, Max Lamparth, Anka Reuel, Linwei Xin, Hanmeng Xu, Jacob Loader, Freddie Martin, Zixuan Wang, Andrea Achilleos, Thomas

Preu, Tomek Korbak, Ida Bosio, Fereshteh Kazemi, Ziye Chen, Biró Bálint, Eve J. Y. Lo, Jiaqi Wang, Maria Inês S. Nunes, Jeremiah Milbauer, M Saiful Bari, Zihao Wang, Behzad Ansarinejad, Yewen Sun, Stephane Durand, Hossam Elgnainy, Guillaume Douville, Daniel Tordera, George Balabanian, Hew Wolff, Lynna Kvistad, Hsiaoyun Milliron, Ahmad Sakor, Murat Eron, Andrew Favre D. O., Shailesh Shah, Xiaoxiang Zhou, Firuz Kamalov, Sherwin Abdoli, Tim Santens, Shaul Barkan, Allison Tee, Robin Zhang, Alessandro Tomasiello, G. Bruno De Luca, Shi-Zhuo Looi, Vinh-Kha Le, Noam Kolt, Jiayi Pan, Emma Rodman, Jacob Drori, Carl J Fossum, Niklas Muennighoff, Milind Jagota, Ronak Pradeep, Honglu Fan, Jonathan Eicher, Michael Chen, Kushal Thaman, William Merrill, Moritz Firsching, Carter Harris, Stefan Ciobâcă, Jason Gross, Rohan Pandey, Ilya Gusev, Adam Jones, Shashank Agnihotri, Pavel Zhelnov, Mohammadreza Mofayezi, Alexander Piperski, David K. Zhang, Kostiantyn Dobarskyi, Roman Leventov, Ignat Soroko, Joshua Duersch, Vage Taamazyan, Andrew Ho, Wenjie Ma, William Held, Ruicheng Xian, Armel Randy Zebaze, Mohanad Mohamed, Julian Noah Leser, Michelle X Yuan, Laila Yacar, Johannes Lengler, Katarzyna Olszewska, Claudio Di Fratta, Edson Oliveira, Joseph W. Jackson, Andy Zou, Muthu Chidambaram, Timothy Manik, Hector Haffenden, Dashiell Stander, Ali Dasouqi, Alexander Shen, Bita Golshani, David Stap, Egor Kretov, Mikalai Uzhou, Alina Borisovna Zhidkovskaya, Nick Winter, Miguel Orbegozo Rodriguez, Robert Lauff, Dustin Wehr, Colin Tang, Zaki Hossain, Shaun Phillips, Fortuna Samuele, Fredrik Ekström, Angela Hammon, Oam Patel, Faraz Farhidi, George Medley, Forough Mohammadzadeh, Madellene Peñaflor, Haile Kassahun, Alena Friedrich, Rayner Hernandez Perez, Daniel Pyda, Taom Sakal, Omkar Dhamane, Ali Khajegili Mirabadi, Eric Hallman, Kenchi Okutsu, Mike Battaglia, Mohammad Maghsoudimehrabani, Alon Amit, Dave Hulbert, Roberto Pereira, Simon Weber, Handoko, Anton Peristyy, Stephen Malina, Mustafa Mehkary, Rami Aly, Frank Reidegeld, Anna-Katharina Dick, Cary Friday, Mukhwinder Singh, Hassan Shapourian, Wanyoung Kim, Mariana Costa, Hubeyb Gurdogan, Harsh Kumar, Chiara Ceconello, Chao Zhuang, Haon Park, Micah Carroll, Andrew R. Tawfeek, Stefan Steinerberger, Daattavya Aggarwal, Michael Kirchhof, Linjie Dai, Evan Kim, Johan Ferret, Jainam Shah, Yuzhou Wang, Minghao Yan, Krzysztof Burdzy, Lixin Zhang, Antonio Franca, Diana T. Pham, Kang Yong Loh, Joshua Robinson, Abram Jackson, Paolo Giordano, Philipp Petersen, Adrian Cosma, Jesus Colino, Colin White, Jacob Votava, Vladimir Vinnikov, Ethan Delaney, Petr Spelda, Vit Stritecky, Syed M. Shahid, Jean-Christophe Mourrat, Lavr Vetoshkin, Koen Sponselee, Renas Bacho, Zheng-Xin Yong, Florencia de la Rosa, Nathan Cho, Xiuyu Li, Guillaume Malod, Orion Weller, Guglielmo Albani, Leon Lang, Julien Laurendeau, Dmitry Kazakov, Fatimah Adesanya, Julien Portier, Lawrence Hollom, Victor Souza, Yuchen Anna Zhou, Julien Degorre, Yiğit Yalın, Gbenga Daniel Obikoya, Rai, Filippo Bigi, M. C. Boscá, Oleg Shumar, Kaniuar Bacho, Gabriel Recchia, Mara Popescu, Nikita Shulga, Ngefor Mildred Tanwie, Thomas C. H. Lux, Ben Rank, Colin Ni, Matthew Brooks, Alesia Yakimchyk, Huanxu, Liu, Stefano Cavalleri, Olle Häggström, Emil Verkama, Joshua Newbould, Hans Gundlach, Leonor Brito-Santana, Brian Amaro, Vivek Vajipey, Rynaa Grover, Ting Wang, Yosi Kratish, Wen-Ding Li, Sivakanth Gopi, Andrea Caciolai, Christian Schroeder de Witt, Pablo Hernández-Cámara, Emanuele Rodolà, Jules Robins, Dominic Williamson, Vincent Cheng, Brad Raynor, Hao Qi, Ben Segev, Jingxuan Fan, Sarah Martinson, Erik Y. Wang, Kaylie Hausknecht, Michael P. Brenner, Mao Mao, Christoph Demian, Peyman Kassani, Xinyu Zhang, David Avagian, Eshawn Jessica Scipio, Alon Ragoler, Justin Tan, Blake Sims, Rebeka Plecnik, Aaron Kirtland, Omer Faruk Bodur, D. P. Shinde, Yan Carlos Leyva Labrador, Zahra Adoul, Mohamed Zekry, Ali Karakoc, Tania C. B. Santos, Samir Shamseldeen, Loukmane Karim, Anna Liakhovitskaia, Nate Resman, Nicholas Farina, Juan Carlos Gonzalez, Gabe Maayan, Earth Anderson, Rodrigo De Oliveira Pena, Elizabeth Kelley, Hodjat Mariji, Rasoul Pouriamanesh, Wentao Wu, Ross Finocchio, Ismail Alarab, Joshua Cole, Danyelle Ferreira, Bryan Johnson, Mohammad Safdari, Liangti Dai, Siriphan Arthornthurasuk, Isaac C. McAlister, Alejandro José Moyano, Alexey Pronin, Jing Fan, Angel Ramirez-Trinidad, Yana Malysheva, Daphiny Pottmaier, Omid Taheri, Stanley Stepanic, Samuel Perry, Luke Askew, Raúl Adrián Huerta Rodríguez, Ali M. R. Minissi, Ricardo Lorena, Krishnamurthy Iyer, Arshad Anil Fasiludeen, Ronald Clark, Josh Ducey, Matheus Piza, Maja Somrak, Eric Vergo, Juehang Qin, Benjámin Borbás, Eric Chu, Jack Lindsey, Antoine Jallon, I. M. J. McInnis, Evan Chen, Avi Semler, Luk Gloor, Tej Shah, Marc Carauleanu, Pascal Lauer, Tran Duc Huy, Hossein Shahrtash, Emilien Duc, Lukas Lewark, Assaf Brown, Samuel Albanie, Brian Weber, Warren S. Vaz, Pierre Clavier, Yiyang Fan, Gabriel Poesia Reis e Silva, Long, Lian, Marcus Abramovitch, Xi Jiang, Sandra Mendoza, Murat Islam, Juan Gonzalez, Vasilios Mavroudis, Justin Xu, Pawan Kumar, Laxman Prasad Goswami, Daniel Bugas, Nasser Heydari, Ferenc Jeanplong, Thorben Jansen, Antonella Pinto, Archimedes Apronti, Abdallah Galal, Ng Ze-An, Ankit Singh, Tong Jiang, Joan of Arc Xavier, Kanu Priya Agarwal, Mohammed Berkani, Gang Zhang, Zhehang Du, Benedito Alves de Oliveira Junior, Dmitry Malishev, Nicolas Remy, Taylor D. Hartman, Tim Tarver, Stephen Mensah, Gautier Abou Loume, Wiktor Morak, Farzad Habibi, Sarah Hoback, Will Cai, Javier Gimenez, Roselynn Grace Montecillo, Jakub Łucki, Russell Campbell, Asankhaya Sharma, Khalida Meer, Shreen Gul, Daniel Espinosa Gonzalez, Xavier Alapont, Alex Hoover, Gunjan Chhablani, Freddie Vargus, Arunim Agarwal, Yibo Jiang, Deepakkumar Patil, David Outevsky, Kevin Joseph Scaria, Rajat Maheshwari, Abdelkader Dendane, Priti Shukla, Ashley Cartwright, Sergei Bogdanov, Niels Mündler, Sören Möller, Luca Arnaboldi, Kunvar Thaman, Muhammad Rehan Siddiqi, Prajvi Saxena, Himanshu Gupta, Tony Fruhauff, Glen Sherman, Mátyás Vincze, Siranut Usawasutsakorn, Dylan Ler, Anil Radhakrishnan, Innocent Enyekwe, Sk Md Salauddin, Jiang Muzhen, Aleksandr Maksapetyan, Vivien Rossbach, Chris Harjadi, Mohsen Bahaloohoreh, Claire Sparrow, Jasdeep Sidhu, Sam Ali, Song Bian, John Lai, Eric Singer, Justine Leon Uro, Greg Bateman, Mohamed Sayed, Ahmed Menshawy, Darling Duclosel, Dario Bezzi, Yashaswini Jain, Ashley Aaron, Murat Tiryakioglu, Sheeshram Siddh, Keith Krenek, Imad Ali Shah, Jun Jin, Scott Creighton, Denis Peskoff, Zienab EL-Wasif, Ragavendran P V, Michael Richmond, Joseph McGowan, Tejal Patwardhan, Hao-Yu Sun, Ting Sun, Nikola Zubić, Samuele Sala, Stephen Ebert, Jean Kaddour, Manuel Schottdorf, Dianzhuo Wang, Gerol Petruzella, Alex Meiburg, Tilen Medved, Ali ElSheikh, S Ashwin Hebbar, Lorenzo Vaquero, Xianjun Yang, Jason Poulos, Vilém Zouhar, Sergey Bogdanik, Mingfang Zhang, Jorge Sanz-Ros, David Anugraha, Yinwei Dai, Anh N. Nhu, Xue Wang, Ali Anil Demircali, Zhibai Jia, Yuyin Zhou, Juncheng Wu, Mike He, Nitin Chandok, Aarush Sinha, Gaoxiang Luo, Long Le, Mickaël Noyé, Michał Perełkiewicz, Ioannis Pantidis, Tianbo Qi, Soham Sachin Purohit, Letitia Parcalabescu, Thai-Hoa Nguyen, Genta Indra Winata, Edoardo M. Ponti, Hanchen Li, Kaustubh Dhole, Jongee Park, Dario Abbondanza, Yuanli Wang, Anupam Nayak, Diogo M. Caetano, Antonio A. W. L. Wong, Maria del Rio-Chanona, Dániel Kondor, Pieter Francois, Ed Chalstrey, Jakob Zsambok, Dan Hoyer, Jenny Reddish, Jakob Hauser, Francisco-Javier Rodrigo-Ginés, Suchandra Datta, Maxwell Shepherd, Thom Kamphuis, Qizheng Zhang, Hyunjun Kim, Ruiji Sun, Jianzhu Yao, Franck Dernoncourt, Satyapriya Krishna, Sina Rismanchian, Bonan Pu, Francesco Pinto, Yingheng Wang, Kumar Shridhar, Kalon J. Overholt, Glib Briia, Hieu Nguyen, David, Soler Bartomeu, Tony CY Pang, Adam Wecker, Yifan Xiong, Fanfei Li, Lukas S. Huber, Joshua Jaeger, Romano De Maddalena, Xing Han Lù, Yuhui Zhang, Claas Beger, Patrick Tser Jern Kon, Sean Li, Vivek Sanker, Ming Yin, Yihao Liang, Xinlu Zhang, Ankit Agrawal, Li S. Yifei, Zechen Zhang, Mu Cai, Yasin Sonmez, Costin Cozianu, Changhao Li, Alex Slen, Shoubin Yu, Hyun Kyu Park, Gabriele Sarti, Marcin Briański, Alessandro Stolfo, Truong An Nguyen, Mike Zhang, Yotam Perlitz, Jose Hernandez-Orallo, Runjia Li, Amin Shabani, Felix Juefei-Xu, Shikhar Dhingra, Orr Zohar, My Chiffon Nguyen, Alexander Pondaven, Abdurrahim Yilmaz, Xuandong Zhao, Chuanyang Jin, Muyan Jiang, Stefan Todoran, Xinyao Han, Jules Kreuer, Brian Rabern, Anna Plassart, Martino Maggetti, Luther Yap, Robert Geirhos, Jonathon Kean, Dingsu Wang, Sina Mollaei, Chenkai Sun, Yifan Yin, Shiqi Wang, Rui Li, Yaowen Chang, Anjiang Wei, Alice Bizeul, Xiaohan Wang, Alexandre Oliveira Arrais, Kushin Mukherjee, Jorge Chamorro-Padial, Jiachen Liu, Xingyu Qu, Junyi Guan, Adam Bouyamourn, Shuyu Wu, Martyna Plomecka, Junda Chen, Mengze Tang, Jiaqi Deng, Shreyas Subramanian, Haocheng Xi, Haoxuan Chen, Weizhi Zhang, Yinuo Ren, Haoqin Tu, Sejong Kim, Yushun Chen, Sara Vera Marjanović, Junwoo Ha, Grzegorz Luczyna, Jeff J. Ma, Zewen Shen, Dawn Song, Cedegao E. Zhang, Zhun Wang, Gaël Gendron, Yunze Xiao, Leo Smucker, Erica Weng, Kwok Hao Lee, Zhe Ye, Stefano Ermon, Ignacio D. Lopez-Miguel, Theo Knights, Anthony Gitter, Namkyu Park, Boyi Wei, Hongzheng Chen, Kunal Pai, Ahmed Elkhanany, Han Lin, Philipp D. Siedler, Jichao Fang, Ritwik Mishra, Károly Zsolnai-Fehér, Xilin Jiang, Shadab Khan, Jun Yuan, Rishab Kumar Jain, Xi Lin, Mike Peterson, Zhe Wang, Aditya Malusare, Maosen Tang, Isha Gupta, Ivan Fosin, Timothy Kang, Barbara Dworakowska, Kazuki Matsumoto, Guangyao Zheng, Gerben Sewuster, Jorge Pretel Villanueva, Ivan Rannev, Igor Chernyavsky, Jiale Chen, Deepayan Banik, Ben Racz, Wenchao Dong, Jianxin Wang, Laila Bashmal, Duarte V. Gonçalves, Wei Hu, Kaushik Bar, Ondrej Bohdal, Atharv Singh Patlan, Shehzaad Dhuliawala, Caroline Geirhos, Julien Wist, Yuval Kansal, Bingsen Chen, Kutay Tire, Atak Talay Yücel, Brandon Christof, Veerupaksh Singla, Zijian Song, Sanxing Chen, Jiaxin Ge, Kaustubh Ponkshe, Isaac Park, Tianneng Shi, Martin Q. Ma, Joshua Mak, Sherwin Lai, Antoine Moulin, Zhuo Cheng, Zhanda Zhu, Ziyi Zhang, Vaidehi Patil, Ketan Jha, Qiutong Men, Jiaxuan Wu, Tianchi Zhang, Bruno Hebling Vieira, Alham Fikri Aji, Jae-Won Chung, Mohammed Mahfoud, Ha Thi Hoang, Marc Sperzel, Wei Hao,

Kristof Meding, Sihan Xu, Vassilis Kostakos, Davide Manini, Yueying Liu, Christopher Toukmaji, Jay Paek, Eunmi Yu, Arif Engin Demircali, Zhiyi Sun, Ivan Dewerpe, Hongsen Qin, Roman Pflugfelder, James Bailey, Johnathan Morris, Ville Heilala, Sybille Rosset, Zishun Yu, Peter E. Chen, Woongyeong Yeo, Eeshaan Jain, Ryan Yang, Sreekar Chigurupati, Julia Chernyavsky, Sai Prajwal Reddy, Subhashini Venugopalan, Hunar Batra, Core Francisco Park, Hieu Tran, Guilherme Maximiano, Genghan Zhang, Yizhuo Liang, Hu Shiyu, Rongwu Xu, Rui Pan, Siddharth Suresh, Ziqi Liu, Samaksh Gulati, Songyang Zhang, Peter Turchin, Christopher W. Bartlett, Christopher R. Scotese, Phuong M. Cao, Ben Wu, Jacek Karwowski, Davide Scaramuzza, Aakaash Nattanmai, Gordon McKellips, Anish Cheraku, Asim Suhail, Ethan Luo, Marvin Deng, Jason Luo, Ashley Zhang, Kavin Jindel, Jay Paek, Kasper Halevy, Allen Baranov, Michael Liu, Advaith Avadhanam, David Zhang, Vincent Cheng, Brad Ma, Evan Fu, Liam Do, Joshua Lass, Hubert Yang, Surya Sunkari, Vishruth Bharath, Violet Ai, James Leung, Rishit Agrawal, Alan Zhou, Kevin Chen, Tejas Kalpathi, Ziqi Xu, Gavin Wang, Tyler Xiao, Erik Maung, Sam Lee, Ryan Yang, Roy Yue, Ben Zhao, Julia Yoon, Sunny Sun, Aryan Singh, Ethan Luo, Clark Peng, Tyler Osbey, Taozhi Wang, Daryl Echeazu, Hubert Yang, Timothy Wu, Spandan Patel, Vidhi Kulkarni, Vijaykaarti Sundarapandiyan, Ashley Zhang, Andrew Le, Zafir Nasim, Srikar Yalam, Ritesh Kasamsetty, Soham Samal, Hubert Yang, David Sun, Nihar Shah, Abhijeet Saha, Alex Zhang, Leon Nguyen, Laasya Nagumalli, Kaixin Wang, Alan Zhou, Aidan Wu, Jason Luo, Anwith Telluri, Summer Yue, Alexandr Wang, and Dan Hendrycks. Humanity's last exam, 2025. URL https://arxiv.org/abs/2501.14249.

[16] Idan Shenfeld, Jyothish Pari, and Pulkit Agrawal. Rl's razor: Why online reinforcement learning forgets less, 2025. URL https://arxiv.org/abs/2509.04259.

[17] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, pp. 1279–1297. ACM, March 2025. doi: 10.1145/3689031.3696075. URL http://dx.doi.org/10.1145/3689031.3696075.

[18] Hongjin Su, Shizhe Diao, Ximing Lu, Mingjie Liu, Jiacheng Xu, Xin Dong, Yonggan Fu, Peter Belcak, Hanrong Ye, Hongxu Yin, Yi Dong, Evelina Bakhturina, Tao Yu, Yejin Choi, Jan Kautz, and Pavlo Molchanov. Toolorchestra: Elevating intelligence via efficient model and tool orchestration, 2025. URL https://arxiv.org/abs/2511.21689.

[19] Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference, 2025. URL https://arxiv.org/abs/2410.21465.

[20] 5 Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, Yifan An, Yilin Niu, Yuanhao Wen, Yushi Bai, Zhengxiao Du, Zihan Wang, Zilin Zhu, Bohan Zhang, Bosi Wen, Bowen Wu, Bowen Xu, Can Huang, Casey Zhao, Changpeng Cai, Chao Yu, Chen Li, Chendi Ge, Chenghua Huang, Chenhui Zhang, Chenxi Xu, Chenzheng Zhu, Chuang Li, Congfeng Yin, Daoyan Lin, Dayong Yang, Dazhi Jiang, Ding Ai, Erle Zhu, Fei Wang, Gengzheng Pan, Guo Wang, Hailong Sun, Haitao Li, Haiyang Li, Haiyi Hu, Hanyu Zhang, Hao Peng, Hao Tai, Haoke Zhang, Haoran Wang, Haoyu Yang, He Liu, He Zhao, Hongwei Liu, Hongxi Yan, Huan Liu, Huilong Chen, Ji Li, Jiajing Zhao, Jiamin Ren, Jian Jiao, Jiani Zhao, Jianyang Yan, Jiaqi Wang, Jiayi Gui, Jiayue Zhao, Jie Liu, Jijie Li, Jing Li, Jing Lu, Jingsen Wang, Jingwei Yuan, Jingxuan Li, Jingzhao Du, Jinhua Du, Jinxin Liu, Junkai Zhi, Junli Gao, Ke Wang, Lekang Yang, Liang Xu, Lin Fan, Lindong Wu, Lintao Ding, Lu Wang, Man Zhang, Minghao Li, Minghuan Xu, Mingming Zhao, Mingshu Zhai, Pengfan Du, Qian Dong, Shangde Lei, Shangqing Tu, Shangtong Yang, Shaoyou Lu, Shijie Li, Shuang Li, Shuang-Li, Shuxun Yang, Sibo Yi, Tianshu Yu, Wei Tian, Weihan Wang, Wenbo Yu, Weng Lam Tam, Wenjie Liang, Wentao Liu, Xiao Wang, Xiaohan Jia, Xiaotao Gu, Xiaoying Ling, Xin Wang, Xing Fan, Xingru Pan, Xinyuan Zhang, Xinze Zhang, Xiuqing Fu, Xunkai Zhang, Yabo Xu, Yandong Wu, Yida Lu, Yidong Wang, Yilin Zhou, Yiming Pan, Ying Zhang, Yingli Wang, Yingru Li, Yinpei Su, Yipeng Geng, Yitong Zhu, Yongkun Yang, Yuhang Li, Yuhao Wu, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yuxuan Zhang, Zezhen Liu, Zhen Yang, Zhengda Zhou, Zhongpei Qiao, Zhuoer Feng, Zhuorui Liu, Zichen Zhang, Zihan Wang, Zijun Yao, Zikang Wang, Ziqiang Liu, Ziwei

Chai, Zixuan Li, Zuodong Zhao, Wenguang Chen, Jidong Zhai, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025. URL https://arxiv.org/abs/2508.06471.

[21] vLLM Team. KV-aware Routing — vLLM Production Stack Documentation. https://docs.vllm.ai/projects/production-stack/en/vllm-stack-0.1.5/tutorials/kvaware.html, 2025. Accessed: 2026-01-15.

[22] Weixun Wang, XiaoXiao Xu, Wanhe An, Fangwen Dai, Wei Gao, Yancheng He, Ju Huang, Qiang Ji, Hanqi Jin, Xiaoyang Li, et al. Let it flow: Agentic crafting on rock and roll, building the rome model within an open agentic learning ecosystem. *arXiv preprint arXiv:2512.24873*, 2025.

[23] 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, 2025. URL https://arxiv.org/abs/2407.16741.

[24] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024. URL https://arxiv.org/abs/2404.07972.

[25] 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, 2025. URL https://arxiv.org/abs/2508.18572.

[26] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

[27] 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, 2024. URL https://arxiv.org/abs/2405.15793.

[28] Lingfan Yu, Jinkun Lin, and Jinyang Li. Stateful large language model serving with pensieve. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, pp. 144–158. ACM, March 2025. doi: 10.1145/3689031.3696086. URL http://dx.doi.org/10.1145/3689031.3696086.

[29] Yichao Yuan, Advait Iyer, Lin Ma, and Nishil Talati. Vortex: Overcoming memory capacity limitations in gpu-accelerated large-scale data analytics, 2025. URL https://arxiv.org/abs/2502.09541.

[30] Hanchen Zhang, Xiao Liu, Bowen Lv, Xueqiao Sun, Bohao Jing, Iat Long Iong, Zhenyu Hou, Zehan Qi, Hanyu Lai, Yifan Xu, et al. Agentrl: Scaling agentic reinforcement learning with a multi-turn, multi-task framework. *arXiv preprint arXiv:2510.04206*, 2025.

[31] Lei Zhang, Mouxiang Chen, Ruisheng Cao, Jiawei Chen, Fan Zhou, Yiheng Xu, Jiaxi Yang, Liang Chen, Changwei Luo, Kai Zhang, et al. Megaflow: Large-scale distributed orchestration system for the agentic era. *arXiv preprint arXiv:2601.07526*, 2026.

[32] Chujie Zheng, Kai Dang, Bowen Yu, Mingze Li, Huiqiang Jiang, Junrong Lin, Yuqiong Liu, Hao Lin, Chencan Wu, Feng Hu, An Yang, Jingren Zhou, and Junyang Lin. Stabilizing reinforcement learning with llms: Formulation and practices, 2025. URL https://arxiv.org/abs/2512.01374.

[33] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs, 2024. URL https://arxiv.org/abs/2312.07104.

[34] 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, 2024. URL https://arxiv.org/abs/2401.09670.

[35] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024. URL https://arxiv.org/abs/2307.13854.

A Extended Comparison with Prior Work

A.1 KV Cache Optimization

Multi-tiered KV cache management. To alleviate GPU memory pressure, systems such as Pensieve [28], Strata [25], and ShadowKV [19] leverage the hardware memory hierarchy, comprising GPU HBM, CPU DRAM, and NVMe SSD to manage KV states. These tiered caching mechanisms mitigate transient preemption by offloading inactive KV blocks to lower-tier storage and prefetching them back upon request resumption. However, the efficiency of these methods is fundamentally constrained by the inter-tier bandwidth and capacity of the host-memory and disk layers. In high-frequency agentic workflows, the overhead of frequent swap-in and swap-out cycles often negates the benefits of caching shown in Section 5.

Distributed Management of KV Caches. Distributed management of agentic states introduces significant complexity to KV cache eviction and preemption policies. While systems like BanaServe [6] and LMCache [12] enable KV-cache transfer across DP nodes, their performance in large-batch agentic serving and rollout is often constrained by limited interconnect bandwidth. The strong intra-program dependencies in agentic workflows necessitate frequent state transferring without program-level management, which can easily saturate the network during serving or rollout.

To bypass these bandwidth bottlenecks, standard frameworks like vLLM KV aware router [21] and SGLang Model Gateway [33] employ KV-aware routers that pin requests to specific nodes based on prefix locality or session ID. Similarly, Vortex [29] introduces session-aware prefetching to minimize cross-node latencies. However, these approaches lack the capability to dynamically migrate active program states between DP nodes. This absence of workload transfer leads to severe memory utilization imbalance across the cluster, shown in Figure 2a. Nodes hosting long-running agentic programs cannot offload states to idle peers, resulting in fragmented resource utilization and degraded aggregate throughput.

A.2 Extended experiment results on KV cache optimization

Experiments on KV cache offloading. We investigated KV cache offloading using LMcache [12] as a potential remedy for capacity constraints. While offloading theoritically extends effective memory space by utilizing CPU or disk storage, our implementation with vLLM+LMcache reveals a critical bottleneck: the PCIe bandwidth is insufficient to sustain the high-frequency context switching and large-volume data transfers inherent to agentic workloads. As demonstrated in Figure 7a, when serving the GLM-4.6 model [20] with the mini-SWEAgent framework [27], the latency penalty from frequent swap-in/swap-out operations negates the capacity benefits, resulting in severe throughput degradation under heavy agentic workloads.

Experiments on Prefill-Decode (PD) disaggregation. We also explored PD disaggregation [34], a standard optimization for chatbot serving by isolating the decoding phase from prefill interference. However, when applied to agentic workloads characterized by continuous context growth, we observe that PD disaggregation ***exacerbates*** thrashing. By partitioning the cluster into prefill-only and decode-only nodes, the effective HBM pool available for handling prefill is significantly smaller than that in a unified architecture. This memory fragmentation causes the system to hit capacity limits and trigger thrashing at much lower concurrency levels, as shown in Figure 7b. These results demonstrate that generic architectural optimizations cannot substitute for a program-centric scheduler that actively manages the working set.

A.3 Scaling up agentic workflows

Heterogeneous resource allocation and scheduling. To orchestrate multi-turn agent-environment interactions at scale, recent systems such as MegaFlow [31], RollArt [5, 22], AgentRL [30], and VerlTool [8] decouple model inference from environment execution. While these frameworks effectively scale environment concurrency via specialized services, they exhibit the inherent limitations of coarse-grained disaggregation. By treating the inference engine and tool executor as isolated black boxes, these systems lack unified resource management and are unable to coordinate KV cache lifecycles with the environment execution. Without fine-grained scheduling at program-level, disaggregation-based approaches waste KV cache reuse potential in agentic workloads, yielding sub-optimal throughput.

B System portability and interface abstraction.

B.1 Middleware architecture and unified interfaces.

ThunderAgent decouples agent control flow and the underlying inference backends via a program-level abstraction. The scheduler controls program state transitions based on the abstracted `ProgramState` (see Table 3a) together with the backend's cache capacity view (see Table 4). Meanwhile, each program binds only to the endpoint and does not depend on the concrete backend implementation.

B.2 Why program ID matters.

While the standard **session ID** serves as a routing label, **the program ID is used by our system to check the workflow metadata.** This visibility is critical: it allows the scheduler to distinguish valid tool-wait times from idle sessions, enabling smart preemption strategies that session-based baselines cannot support.

B.3 Low-overhead adoption of the ThunderAgent.

Figure 8 shows that adopting ThunderAgent **only requires** attaching `program_id` to requests (for both LLM inference and tool execution) and sending an explicit release signal with `program_id` when a program ends. The `program_id` tags each request with its own program instance for scheduling, while the release signal allows ThunderAgent to reclaim per-program resources after termination. **All other request fields and the OpenAI-style API surface remain unchanged.**

Field	**Type**	**Meaning**
ProgramState		
`status`	ProgramStatus	Current lifecycle state.
`backend_url`	str	Assigned backend endpoint.
`step_count`	int	Executed steps so far.
`total_tokens`	int	Total tokens over full history.

(a) ProgramState fields.

Status	**Meaning**
ProgramStatus	
`REASONING`	On-GPU inference.
`ACTING`	Off-GPU tool exec.
`PAUSED`	In global paused waiting set.
`STOPPED`	Released; resources reclaimed.

(b) ProgramStatus semantics.

Table 3: Program state and status definitions.

Field	**Type**	**Meaning**
BackendState		
`url`	str	Backend endpoint.
`healthy`	bool	Health flag for scheduling.
`cache_config`	Optional[CacheConfig]	Static cache configuration (fetched at startup).
`active_program_tokens`	int	Active token footprint on this backend.

Table 4: Key fields of BackendState.

C Tool execution time variability.

Practical agent tool calls are hard to characterize and often unpredictable. In some code-centric settings (e.g., serving SWE-Bench [9] with SWE-agent [27] or OpenHands [23]), agents primarily invoke local, lightweight tools, and tool latency is relatively stable with low variance. However, in broader and more realistic scenarios, e.g., serving HLE [15] with ToolOrchestra [18], the workload relies more heavily on remote-service tools (Table 5), making tool execution time volatile and difficult to predict. This volatility largely stems from factors external to the agent runtime, such as network jitter, backend load and queuing delays, and rate limiting, which can vary across requests and over time.

We empirically confirm this behavior in Figure 9. For remote-service tools (and some execution tools), the gap between the median and tail quantiles is large: p95 and p99 are substantially higher than the median, and the tail can extend to tens or even hundreds of seconds. This suggests that tool latency in these settings lacks a stable central tendency; instead, heavy-tailed behavior dominates, **making tool latency prediction intrinsically brittle in practice.**

Given the unpredictability of tool execution, underestimation wastes pinned cache capacity while still triggering premature KV eviction, causing thrashing upon resume. Overestimation, in contrast, may lead to unnecessary eviction of programs' KV that should have remained pinned. Even if tool runtimes were perfectly predictable, existing methods such as continuum [11] still decide whether to keep the KV cache pinned using a static, threshold-based rule. In contrast, ThunderAgent builds **a complete cost-modeling framework** and **dynamically trades off** $\text{Cost}_{\text{recompute}}$ and $\text{Cost}_{\text{caching}}$.

Tool bucket	Role	Primary variability source
`HLE-search`	Retrieve evidence	Remote service(Network latency/Rate limits)
`HLE-enhance-reasoning`	Model-as-a-tool call	Remote service
`HLE-answer`	Final generation	Local LLM inference
`SAB-execute_bash`	Shell execution	Sandbox and I/O
`SAB-execute_ipython_cell`	Python cell execution	Program runtime
`SAB-str_replace_editor`	File edit	Local filesystem
`SAB-task_tracker`	Task state tracking	Local filesystem

Table 5: Tool buckets.

D KV cache hit rate statistics and interpretation

In our cost decomposition Equation (3), throughput loss in agentic serving mainly comes from *non-productive* overheads: KV re-computation induced by thrashing and idle KV caching during external tool execution, i.e., $\text{Cost}_{\text{recompute}}$ and $\text{Cost}_{\text{caching}}$. When tool calls are short and predictable, the acting phase occupies KV for only a short time, so $\text{Cost}_{\text{caching}}$ is small; thus, avoiding thrashing dominates: a higher KV cache hit rate typically implies fewer re-prefills and higher throughput.

However, when tool execution times are highly variable (see Appendix C), a TTL-based scheduler can end up pinning the KV for long tool calls. While this can reduce $\text{Cost}_{\text{recompute}}$ and thus increase the KV cache hit rate, it simultaneously inflates $\text{Cost}_{\text{caching}}$ and reduces throughput. This helps explain why continuum [11] can underperform on tool-heavy workloads despite achieving a higher KV cache hit rate (Figs. 4, 5).

ThunderAgent adapts to these regimes by explicitly balancing caching and recomputation. ThunderAgent introduces a time-decay function $f(t)$ in Sec. 4.3 for acting programs to trade off $\text{Cost}_{\text{caching}}$ and $\text{Cost}_{\text{recompute}}$; we rigorously derive the optimal functional form of $f(t)$ in Appendix E.1. By progressively lowering the effective memory priority of long-idle acting programs, the scheduler evicts their KV caches to reduce idle caching cost while controlling recomputation, yielding better throughput in practice (Figs. 4).

E Extended Theoretical Analysis.

E.1 Proof of Time Decay Function for Periodic Thrashing Detection.

Hypothesis E.1 (Unpredictable Tool Execution Time)**.** *For acting programs, we hypothesize that the scheduler cannot reliably predict the tool return time for a given program (see Appendix C). Consequently, the decay function f should depend only on the elapsed acting time t in a time-homogeneous manner [14].*

Hypothesis E.2 (Boundary Conditions)**.** *We assume the time decay function $f : [0, \infty) \to (0, 1]$ satisfies*

$$f(0) = 1, \lim_{t \to \infty} f(t) = 0, \tag{12}$$

An intuitive interpretation of these boundary conditions is that, when the tool execution time is 0, corresponding to a multi-turn interaction without tool calls, all acting programs reduce to reasoning programs, and therefore $f(t) = 1$. Conversely, if the tool execution time is infinite, the agentic workflow collapses to single-turn generation, akin to standard chatbot serving, since requests never return for the next-turn interactions. In this regime, setting $f(t) = 0$ aligns the decay function with request-level scheduling policies.

Theorem E.1 (Admissible Time Decay Functions)**.** *Under Hypothesis E.1 and E.2, the admissible time decay function f for our capacity check function in Equation 7 must take one of the following forms: exponential in continuous time, $f(t) = e^{-\lambda t}$ with $\lambda > 0$, or geometric in discrete tick time, $f(k) = x^{-k}$ with $x > 1$.*

Proof. We prove this theorem by first formalizing the time-homogeneous property implied by Hypothesis E.1. Next, we inducing the admissible time decay functions f under the boundary conditions in Hypothesis E.2.

Formalization of unpredictable tool time. Let t denote the elapsed acting time, measured in wall-clock time (continuous time) or in periodic-monitor ticks (discrete time). Under Hypothesis E.1, the relative decay after waiting an additional duration Δ should not depend on the absolute elapsed time t, but only on the increment Δ. We formalize this as the existence of a function $\phi : [0, \infty) \to (0, 1]$ such that, for all $t, \Delta \geq 0$,

$$f(t + \Delta) = f(t)\, \phi(\Delta). \tag{13}$$

Semigroup equation. Setting $t = 0$ in Equation 13 and using the boundary condition $f(0) = 1$ (from Hypothesis E.2) yields $\phi(\Delta) = f(\Delta)$. Substituting back, we obtain the multiplicative semigroup equation

$$f(t + \Delta) = f(t)\, f(\Delta), \qquad \forall t, \Delta \geq 0. \tag{14}$$

Continuous-time case (exponential decay). We first consider the continuous-time case. Define $h(t) \triangleq \ln f(t)$. Applying the logarithms on both sides of Equation 14 yields the *Cauchy functional equation*

$$h(t + \Delta) = h(t) + h(\Delta). \tag{15}$$

Since $f(t) \in (0, 1]$, we have $h(t) \leq 0$ for all $t \geq 0$, which implies that h is bounded above on $[0, \infty)$. Under this boundedness condition, the Cauchy functional equation admits only linear solutions of the form $h(t) = ct$ for some $c \in \mathbb{R}$. Writing $\lambda \triangleq -c \geq 0$, we obtain

$$f(t) = e^{-\lambda t}. \tag{16}$$

Finally, the boundary condition $\lim_{t \to \infty} f(t) = 0$ (Hypothesis E.2) rules out $\lambda = 0$, and thus $\lambda > 0$.

Discrete-time case (geometric decay). We next consider the discrete-time setting, where elapsed acting time is measured in integer ticks $k \in \mathbb{Z}_{\geq 0}$. Equation 14 becomes

$$f(m + n) = f(m)\, f(n), \qquad \forall m, n \in \mathbb{Z}_{\geq 0}. \tag{17}$$

Setting $n = 1$ yields the recurrence $f(k) = f(k-1)f(1)$. Let $\gamma \triangleq f(1)$, we have $f(k) = f(1)^k \triangleq \gamma^k$. The boundary condition $\lim_{k \to \infty} f(k) = 0$ implies $0 < \gamma < 1$. Equivalently, we can parameterize

$$f(k) = x^{-k}, \qquad x \triangleq \gamma^{-1} > 1. \tag{18}$$

This completes the proof. □

E.2 Proof of recomputation STP cost

As defined in Section 4.2, the STP recomputation cost is given by:

$$\text{Cost}_{\text{recompute}} = \int_0^{t_{recompute}} c_i(t)\, dt \tag{19}$$

where $c_i(t)$ represents the instantaneous cost, which is proportional to the decoding step (i.e., $c_i(t) \propto t$). This proportionality arises because chunked prefill processes a constant number of KV pairs per iteration, resulting in a linear increase in accumulated computation over time. Consequently, evaluating the integral yields $\text{Cost}_{\text{recompute}} \propto t^2_{\text{recompute}}$. Given the relationship $t_{\text{recompute}} = c_i \times T_{\text{decode}}/\text{chunk}$, where both T_{decode} and the chunk size are constant, it follows that:

$$Cost_{\text{recompute}} \propto c_i^2$$

E.3 Proof of minimized recomputation STP cost

We provide a rigorous proof for the optimality of the Shortest-First Eviction policy using an **exchange argument**.

Problem Definition. We aim to select a subset of paused programs S to evict such that the total reclaimed memory satisfies $\sum_{i \in S} c_i \geq \Delta C$, while minimizing the total re-computation cost $J(S) = \sum_{i \in S} c_i^2$. Note that the cost function $f(x) = x^2$ is strictly convex and super-additive (i.e., $(a+b)^2 > a^2 + b^2$ for positive a, b).

Theorem. The optimal strategy to minimize $J(S)$ is to strictly select programs with the smallest context lengths c_i.

Proof. Suppose, for the sake of contradiction, that the optimal set S^* is *not* the set of the shortest programs. This implies there exists a ”long” program $p_{long} \in S^*$ and a ”short” program $p_{short} \notin S^*$ (available but not selected) such that $c_{short} < c_{long}$.

We can construct a new set S' by swapping or decomposing p_{long}. Since $c_{long} > c_{short}$, we can conceptualize p_{long} as being composed of a segment of length c_{short} and a residue $r = c_{long} - c_{short}$.

Replacing the selection of p_{long} with p_{short} (and theoretically the residue r) changes the cost. Consider the inequality derived from the convexity of the square function:

$$c_{long}^2 = (c_{short} + r)^2 = c_{short}^2 + r^2 + 2c_{short}r \tag{20}$$

Since $c_{short} > 0$ and $r > 0$, the cross-term $2c_{short}r > 0$. Therefore:

$$c_{short}^2 + r^2 < c_{long}^2 \tag{21}$$

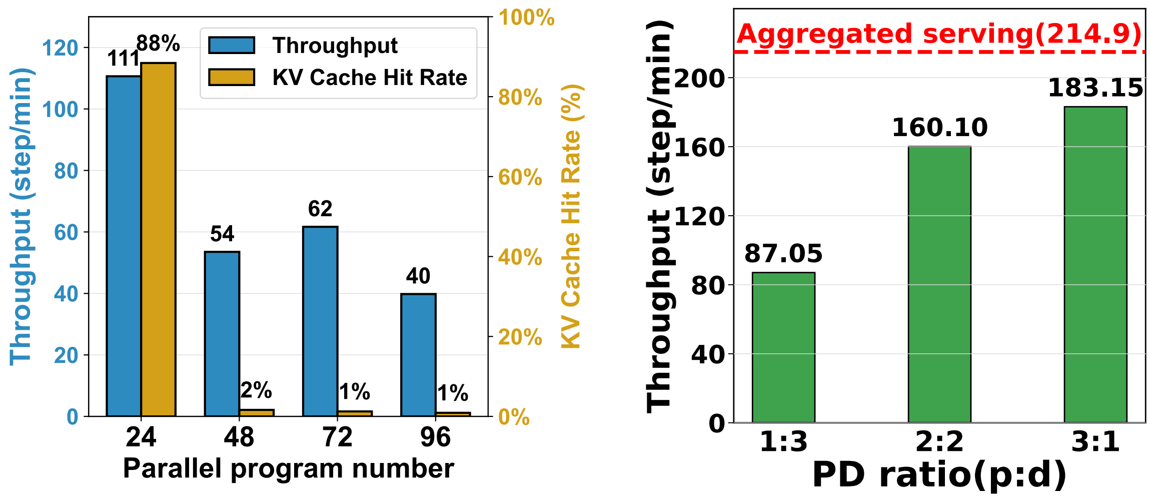
This inequality implies that breaking a large eviction target (c_{long}) into smaller components ($c_{short} + r$) strictly reduces the sum of squares. In the context of our scheduler, this means that if we are satisfying the memory constraint ΔC using a large program, we can strictly decrease the penalty by swapping it for available smaller programs (or a combination thereof) that sum to the same capacity.

By iteratively applying this exchange—replacing the largest selected programs with smaller unselected programs—we monotonically decrease the cost function $J(S)$. The cost reaches its global minimum only when no such exchange is possible, i.e., when S consists entirely of the programs with the smallest available context lengths.

Conclusion. The Shortest-First strategy is globally optimal because the super-linear cost of attention ($O(L^2)$) penalizes fragmentation less than aggregation.

F End-to-End Latency Analysis

Though we have stated in Section 1 that program-level latency(time used for whole workflow generation) is far more important than end-to-end per step latency for autonomous agents and agentic RL rollout. Here we compare ThunderAgent's average per-step latency with vLLM and Continuum. Figure 10 shows that ThunderAgent significantly outperforms vLLM and Continuum when applying GLM4.6 and Qwen3 235B with mini-SWEAgent and Openhands on a single H100 serving in either low or high parallel workflow number. **The reason is that it seems to improve end-to-end latency by switching acting programs. But it actually delays all the running programs' latency by triggering heavy KV-cache thrashing.**



(a) KV Cache Hit Rate with LMCache Offloading

(b) Throughput v.s. Prefill-Only/Decode-Only Node Ratio

Figure 7: Ablation study on KV cache offloading and Prefill-Decode (PD) disaggregation

Only inference backend (e.g., vLLM/SGLang)

```
# 1) LLM request
chat_completion(model_id, messages, extrabody)

# 2) Tool execution
run_tool(command, sandbox)

# 3) Program end
# (no explicit release)
```

With ThunderAgent

```
# 1) LLM request
extrabody["program_id"] = "PID"
chat_completion(model_id, messages, extrabody)

# 2) Tool execution
run_tool(command, sandbox, program_id="PID")

# 3) Program end (explicit release)
POST /programs/release
{ "program_id": "PID" }
```

Figure 8: **Only three changes are required to use the ThunderAgent.**

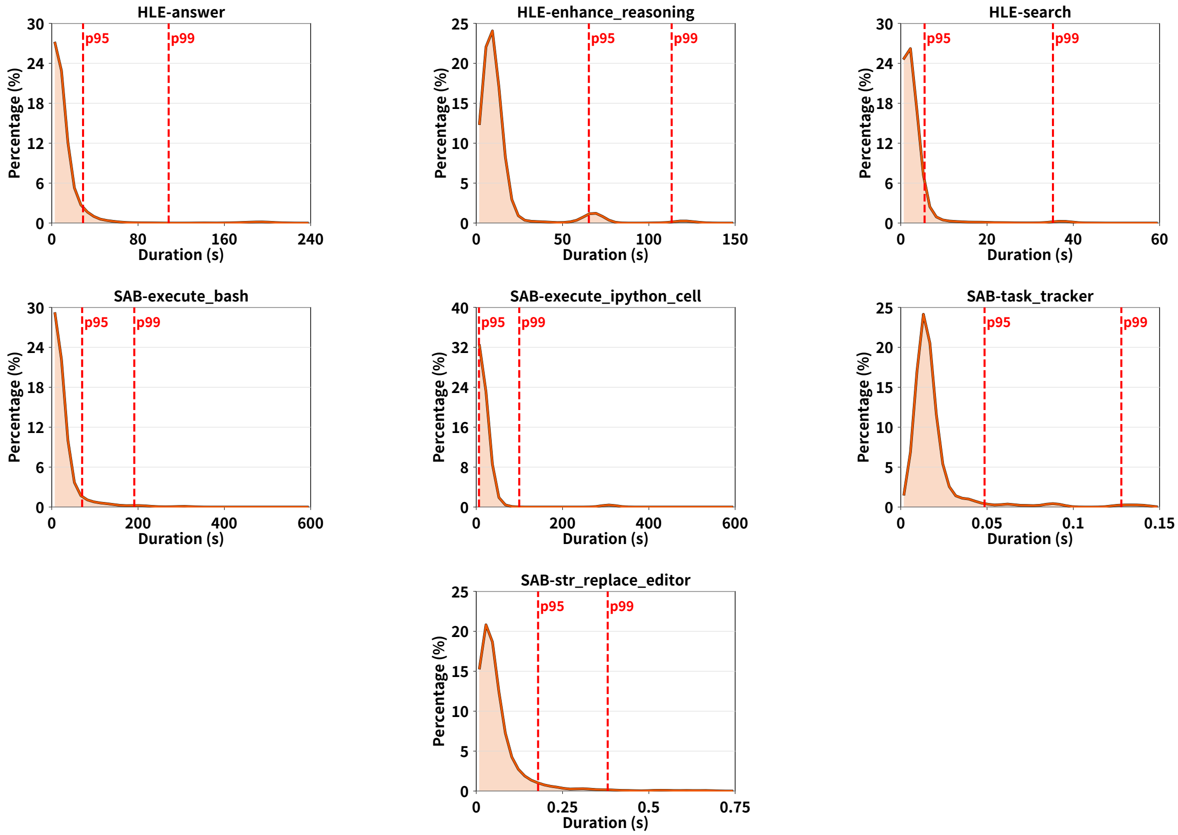


Figure 9: **Tool execution time distributions.**Tool execution time exhibits high variability and is difficult to predict.

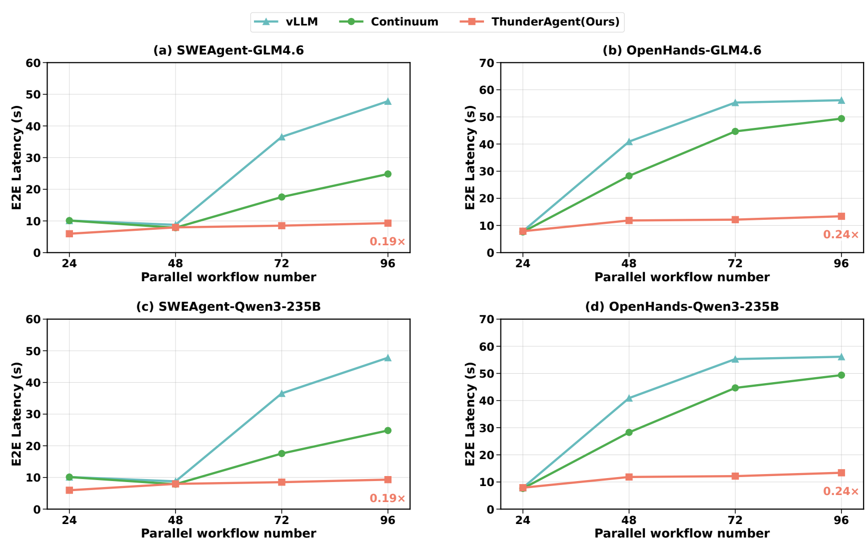


Figure 10: End-to-End latency comparision