

The Cost of Dynamic Reasoning: Demystifying AI Agents and Test-Time Scaling from an AI Infrastructure Perspective

Jiin Kim Byeongjun Shin Jinha Chung Minsoo Rhu

KAIST

{jiin.kim, byeongjun.shin, jinha.chung, mrhu}@kaist.ac.kr

Abstract—Large-language-model (LLM)–based AI agents have recently showcased impressive versatility by employing dynamic reasoning, an adaptive, multi-step process that coordinates with external tools. This shift from static, single-turn inference to agentic, multi-turn workflows broadens task generalization and behavioral flexibility, but it also introduces serious concerns about system-level cost, efficiency, and sustainability. This paper presents the first comprehensive system-level analysis of AI agents, quantifying their resource usage, latency behavior, energy consumption, and datacenter-wide power consumption demands across diverse agent designs and test-time scaling strategies. We further characterize how AI agent design choices, such as few-shot prompting, reflection depth, and parallel reasoning, impact accuracy-cost tradeoffs. Our findings reveal that while agents improve accuracy with increased compute, they suffer from rapidly diminishing returns, widening latency variance, and unsustainable infrastructure costs. Through detailed evaluation of representative agents, we highlight the profound computational demands introduced by AI agent workflows, uncovering a looming sustainability crisis. These results call for a paradigm shift in agent design toward compute-efficient reasoning, balancing performance with deployability under real-world constraints.

I. INTRODUCTION

Recent progress in large language models (LLMs) has shifted from scaling model size or pretraining data to improving inference-time behavior, a direction known as *test-time scaling* [48]. Test-time scaling is designed to enhance model performance by allocating additional computation during inference without modifying the model’s parameters. This includes techniques such as multi-sample decoding (e.g., Best-of-N [51], Self-Consistency [54]), step-by-step prompting (e.g., Chain-of-Thought [55], Least-to-Most Prompting [70], Self-Ask [38]), and structured search strategies (e.g., Tree-of-Thought [63], Graph-of-Thought [5]). These approaches promote more deliberate and interpretable reasoning within the LLM, enabling it not only to recognize patterns but also to derive conclusions, generate explanations, and solve tasks that require step-by-step logic.

Deploying these reasoning-enhanced LLMs, however, comes at an immense computational cost. Even in current **static reasoning** models which follow fixed input-output mappings without external tool interaction (Figure 1(a,b)), LLMs run on thousands of GPUs, whose power, cooling, and capital costs drive monthly expenses into the tens of millions of dollars [44]. A single ChatGPT query is estimated to consume about ten times the electricity of a typical web search [10] and requires a substantial amount of cooling water [50]. As a result, hyperscalers are investing at an unprecedented scale. Meta

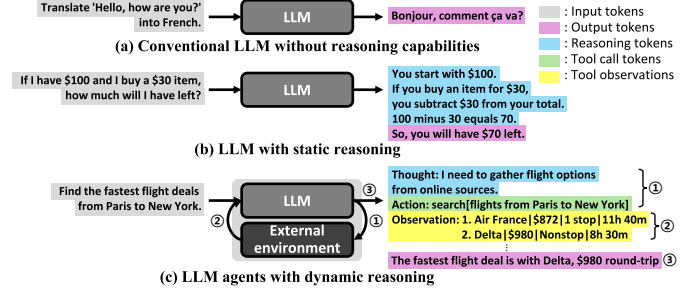


Fig. 1: Overview of test-time scaling. (a) Conventional LLMs map inputs directly to outputs in a single forward pass, with no explicit intermediate reasoning. (b) Reasoning-enhanced LLMs internally create intermediate steps—sampling alternative responses or extending token sequences—to deepen or diversify their thought process. (c) AI agents augment this reasoning by ① planning and invoking external tools, ② observing the outcomes and adapting their internal reasoning accordingly, and iteratively refining their decision-making until they ③ generate the final answer.

has committed over \$10 billion to AI infrastructure [20], and Microsoft, in collaboration with OpenAI, operates custom AI supercomputers that draw tens to hundreds of megawatts. As an example, xAI’s Colossus AI supercomputer alone employs approximately 100,000 Nvidia H100 GPUs, consuming 150 megawatts in total, with the GPUs alone accounting for around 70 megawatts [59]. For perspective, traditional hyperscale data centers typically draw between 10 and 100 megawatts [15], while advanced semiconductor fabs, such as those operated by Samsung and TSMC, consume several hundred megawatts each [4], [16]. Analysts forecast that total AI infrastructure spending will surpass \$1 trillion within this decade [11], raising serious concerns about power grid sustainability, environmental impact, and the economic viability of large-scale LLM deployment.

Given this landscape, the emergence of *AI agents* powered by LLMs with **dynamic reasoning** threatens to exacerbate these already formidable infrastructure pressures dramatically. Unlike static reasoning models, dynamic reasoning represents an advanced form of test-time scaling that significantly boosts capabilities through active interaction with external environments (Figure 1(c)). Specifically, AI agents continuously plan, invoke external tools, observe outcomes, and iteratively refine their reasoning, often performing dozens of inference calls to satisfy a single user request [47], [64], [68]. Without substantial system-level innovations, per-request computational costs could increase by orders of magnitude, making large-scale deployment of agents economically and environmentally

prohibitive. Industry leaders are already responding to these challenges: OpenAI’s upcoming Stargate cluster is projected to consume multiple “gigawatts” of power, with costs potentially reaching \$500 billion [32]. Yet, despite these developments, the computer architecture and systems research communities have largely focused on static LLMs, leaving the infrastructure implications of dynamic reasoning workloads underexplored.

To address this critical gap, this paper presents a rigorous, quantitative evaluation of the computational and infrastructural costs of dynamic reasoning. We systematically characterize resource utilization, latency implications, and energy demands inherent in the iterative execution patterns of AI agents. Our analysis highlights the critical system-level challenges faced when deploying AI agents and identifies opportunities for optimization through architectural improvements, enhanced inference algorithms, and intelligent resource allocation strategies. To the best of our knowledge, this work is the first to provide a system-level characterization of dynamic reasoning in AI agents, grounded in empirical analysis of end-to-end infrastructure behavior across diverse agentic workflows. A central contribution—and key objective—of our study is to quantitatively assess the AI infrastructure costs associated with dynamic reasoning deployments, and to inform and caution the broad research community about the urgent need for sustainable, efficient design principles to bridge the gap between advanced algorithmic capabilities and practical, scalable, and sustainable deployment.

II. BACKGROUND AND MOTIVATION

A. Definition of AI Agents

AI agents are inference-time frameworks that extend the capabilities of large language models by enabling multi-step reasoning, adaptive decision-making, and interaction with the external environment. Unlike conventional LLM applications that produce a single output from a static prompt, AI agents operate through iterative internal reasoning and external actions at inference time. At each iteration, the agent may generate an intermediate reasoning result, call an external tool, such as a search engine, calculator, or code interpreter, and incorporate the output into its subsequent decisions. This process allows the agent to refine its strategy, retrieve missing information, and revise its reasoning *dynamically* in response to evolving task demands. While this adaptivity enhances the agent’s ability to handle complex and open-ended problems, it also leads to variability in the number of model calls, tool usage patterns, and overall computational cost.

B. Core Components and Workflows of AI Agents

As illustrated in Figure 2, AI agents generally consist of four core components (*agent core*, *memory*, *plan*, and *tools*), and *AI agent workflows* interconnect these core components through iterative interactions. These workflows orchestrate how the components dynamically collaborate, enabling the agent’s adaptive behaviors.

The ❶ *agent core* is the central component responsible for *advanced reasoning*, powered by one or more LLMs

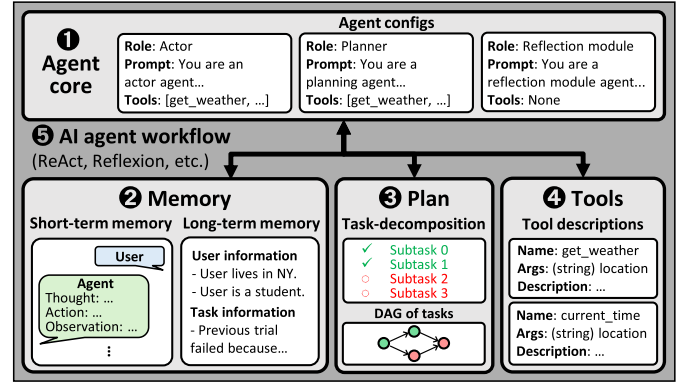


Fig. 2: Overview of AI agent structure.

configured in specific “roles”. These roles typically include an *actor*, which determines the agent’s next action; a *planner*, which decomposes high-level goals into subtasks; and a *reflection module*, which evaluates prior reasoning steps and tool interaction trajectories to guide future decisions.

This core reasoning capability of the AI agent is further supported by *memory*, *plan*, and *tools*. ❷ *Memory* plays a critical role in enabling the agent to maintain continuity across reasoning steps. It stores short-term interaction traces, such as prior LLM outputs and tool responses, as well as long-term knowledge, including user preferences or experience from past interactions. ❸ *Plan* organizes the agent’s objective into a sequence of subtasks or a directed acyclic graph (DAG) of interdependent actions. By maintaining an explicit plan, the agent can prioritize actions, track progress, and make forward-looking decisions that align with the overall task structure. ❹ *Tools* extend the agent’s capabilities beyond text generation by enabling interaction with external environments. At each step, the agent analyzes its current context, generates a structured command specifying the desired tool and input, executes the tool call, and incorporates the resulting output into its context. This output is then used to guide the next stage of reasoning.

Finally, ❺ *AI agent workflow* defines how an agent leverages interactions among the four core components iteratively to carry out reasoning and coordinate actions. AI agents implement their own distinct workflows, reflecting different coordination patterns among components. These workflows can be broadly decomposed into two phases: (1) *LLM inference phase*, where the agent performs internal reasoning tasks such as action generation, planning, or reflection; and (2) *tool use phase*, where the agent interacts with external environments using tools. These two phases alternate iteratively, forming the backbone of AI agentic systems.

C. Test-Time Scaling in AI Agents

Test-time scaling refers to methods that improve the reasoning performance of pretrained LLMs at inference time by increasing the amount of computation used for inference without modifying model parameters [48], [54], [55], [63], [70]. Representative techniques include Chain-of-Thought [55], which guides the model to produce intermediate reasoning steps through carefully crafted prompts, and Tree-of-Thought [63],

which expands the reasoning space by exploring multiple reasoning paths. Both approaches guide the model to perform step-by-step reasoning, effectively leveraging its internal reasoning capabilities without modifying the model parameters.

AI agents build upon this paradigm by implementing test-time reasoning not through prompt design alone, but through multi-step decision-making that integrates tool use and maintenance of intermediate reasoning state. Unlike conventional prompt-based methods that operate within a *static* input-output mapping, agents *dynamically* coordinate multiple model invocations and tool interactions, adapting their behavior based on intermediate outcomes. This form of *dynamic reasoning* enables agents to respond to new information, revise prior decisions, and handle real-time tasks involving external environments. As such, AI agents redefine test-time scaling by moving beyond conventional inference approaches that rely solely on the model’s internal reasoning abilities. Agentic workflows actively gather information, plan actions, and iteratively refine their reasoning over multiple iterations. This paradigm shift introduces new challenges in efficiency, latency, and resource management, highlighting the need for a system-level analysis of the behavior of AI agents.

D. Motivation

While AI agents have demonstrated promising capabilities through iterative reasoning and tool-augmented workflows, their growing complexity introduces profound challenges at the systems and infrastructure level. Unlike conventional single-turn LLM inference where computation is bounded to a single forward pass, agentic execution involves dynamically evolving control flows, multiple rounds of LLM inference, and external tool interactions. These behaviors incur significant compute overhead, amplify memory pressure, and introduce unpredictable latency and utilization patterns.

Despite these operational complexities, prior research on AI agents has largely focused on improving task success rates and qualitative reasoning behavior [47], [64], [68], with little attention paid to the underlying system and its deployment costs. Questions central to the deployment and scaling of such agents—such as GPU under-utilization due to agent’s adaptive control flow, memory bottlenecks from ever-growing contexts, tail-latency behavior under load, or energy implications of multi-step inference—remain largely unexamined. Consequently, existing architecture and systems optimizations for LLMs, which target static, single-pass workloads, may fall short in capturing or addressing the dynamic and iterative characteristics unique to AI agents.

This paper is motivated by the urgent need to fill this gap. To the best of our knowledge, this work is the first to present a rigorous, system-level characterization of AI agents, grounded in empirical measurement across diverse agent designs and tasks. We argue that without a principled understanding of the system-level implications of dynamic reasoning, the community risks building infrastructure optimized for yesterday’s workloads. A systems-oriented perspective is therefore critical to guide the design of sustainable, efficient, and scalable serv-

Agent	Reasoning	Tool Use	Reflection	Tree Search	Structured Planning
CoT [55]	O	X	X	X	X
ReAct [64]	O	O	X	X	X
Reflexion [47]	O	O	O	X	X
LATS [68]	O	O	O	O	X
LLMCompiler [18]	O	O	O	X	O

TABLE I: Comparison of AI agents.

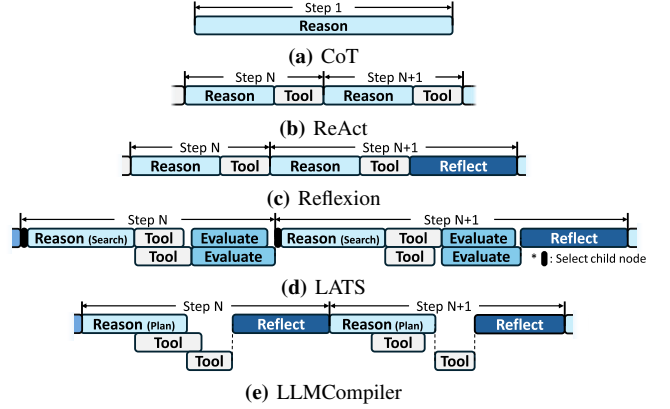


Fig. 3: Execution timeline of each AI agent.

ing infrastructures. Our study takes this first step by analyzing the computational and infrastructural costs of deploying AI agents in practice, providing actionable insights for future architecture and systems co-design.

III. METHODOLOGY

In this section, we describe the experimental setup used to characterize AI agents. We consider a representative set of AI agents and benchmarking workloads, covering diverse agent workflows and agentic task characteristics.

AI agent workflows. To analyze the system-level design space of AI agents, we investigate five representative agents: Chain-of-Thought (CoT) [55], ReAct [64], Reflexion [47], Language Agent Tree Search (LATS) [68], and LLMCompiler [18]. These agents were selected to cover a wide range of reasoning strategies, tool integrations, and planning mechanisms. Table I summarizes the presence or absence of five key capabilities across each agent.

- **Reasoning.** All agents considered in this study employ a reasoning mechanism. Among them, CoT operates purely through internal reasoning without the use of any external tool (Figure 3(a)). As a baseline for comparison, we also include language-only, CoT-style static reasoning approaches within the broader definition of AI agents, despite their lack of external interactions with tools.
- **Tool use.** Tool use differentiates purely language-based agents from those capable of interacting with the environment. This functionality enables agents to access real-time data or perform non-linguistic operations.
- **Reflection.** Reflection allows agents to evaluate past decisions and revise strategies accordingly. Reflective agents effectively manage *long-term memory* by abstracting past trajectories into reflections. While ReAct agents simply repeat reasoning and tool usage (Figure 3(b)), Reflexion,

Benchmark	Property	Description
HotpotQA [61]	Task	Multi-hop question answering
	Tool	Wikipedia APIs (search, lookup keywords)
	Agent	CoT, ReAct, Reflexion, LATS, LLMCompiler
WebShop [62]	Task	Online shopping
	Tool	Interactive web navigation (search, click)
	Agent	ReAct, Reflexion, LATS, LLMCompiler
MATH [14]	Task	Math problem solving
	Tool	Wolfram Alpha API, Python-based calculator
	Agent	CoT, ReAct, Reflexion, LATS
HumanEval [8]	Task	Programming
	Tool	Executing self-generated test code
	Agent	CoT, ReAct, Reflexion, LATS

TABLE II: Description of benchmarks.

the most fundamental reflective agent, enhances adaptability by periodically incorporating self-evaluation and refinement through reflection (Figure 3(c)).

- **Tree search.** LATS (Figure 3(d)) leverages Monte Carlo Tree Search (MCTS) [9] to simulate and expand multiple branches of reasoning and action, allowing the agent to evaluate different candidate paths before making a decision. By simulating multiple possible future paths, the agent can make more informed decisions and select optimal action sequences.
- **Structured planning.** LLMCompiler incorporates a structured multi-step planning and streaming for asynchronous task execution to minimize latency. During the planning phase, LLMCompiler analyzes task dependencies and constructs a directed acyclic graph (DAG) that organizes future tool calls into an execution plan. This enables multiple dependent actions to be generated within a single LLM invocation. As the plan is constructed, intermediate tool calls are streamed to the execution stage, allowing the scheduler to overlap planning and tool calls via asynchronous execution. Together, these features can help reduce repeated reasoning and lower end-to-end latency (Figure 3(e)).

In general, we utilized the official open-source implementations provided by the original authors of these agent workflows [17], [46], [65], [69]. Each AI agent is adapted to support our evaluation framework and benchmarks. For LATS, we further optimized its implementation to support concurrent LLM inference and parallel tool invocation because the original version [69] executes these operations sequentially, aggravating end-to-end latency.

Benchmarks. We select four popular benchmarks representative of various downstream agentic tasks, whose descriptions are summarized in Table II.

HotpotQA [61] is a question-answering benchmark that assesses the agent’s ability to accurately retrieve relevant evidence to answer multi-hop knowledge-intensive questions. We provide the Wikipedia APIs [56] as tools to solve these questions. WebShop [62] is a web-shopping benchmark where agents find the best-fit item that meets the given conditions. The agent is given web navigation tools to browse WebShop. MATH [14] is a benchmark suite of mathematics problems across various domains. Agents are equipped with access to the Wolfram Alpha API [57] for solving

complex equations, as well as a Python-based calculator for simple numerical computations. HumanEval [8] evaluates the programming capability of agents. In our setup, agents are equipped with a Python execution tool that allows them to validate the generated solutions by executing self-written test code. In addition to these agentic benchmarks, we utilize a *non-agentic* dataset, which is the ShareGPT dataset [45], to model conventional chatbot-like LLMs, characterized by single-turn LLM inference without iterative interactions with the external environments. ShareGPT contains a collection of real conversations between users and ChatGPT [29], capturing standard interactive dialogue scenarios.

It is worth pointing out that some “AI agent vs. benchmark” pairs are omitted if the agent is not suitable for solving the target task. For example, CoT is excluded from WebShop since it cannot interact with the shopping webpage. Similarly, LLMCompiler is omitted from MATH and HumanEval, as its DAG-style planning is not well-suited for problems that require sequential, step-by-step reasoning and tool usage.

LLM backend. We employed the OpenAI-compatible vLLM (version 0.6.6) server as the LLM serving infrastructure, integrated with PyTorch 2.6 and CUDA 12.8. We enabled *prefix caching* [19], which reduces redundant computation by reusing previously computed attention states (i.e., *Key-Value cache (KV cache)*) for shared input prefixes across LLM requests. As we explore in Section IV, prefix caching is particularly effective for agent workloads that involve long instructions, few-shot examples, and iterative reasoning tokens in the input prompt. Unless otherwise specified, we use Llama-3.1-8B-Instruct [25] as the default backend LLM. However, to discuss the impact of model size on cost and accuracy, we also use Llama-3.1-70B-Instruct [24] in Section V.

Hardware. Experiments were conducted on Google Cloud Platform (GCP). For the 8B model, we used the a2-highgpu-1g instance type with 12 vCPUs (6 physical cores), 85GB memory, and a single NVIDIA A100 40GB GPU. For the 70B model, we used the a2-highgpu-8g instance type with 96 vCPUs (48 physical cores), 680GB memory, and 8 NVIDIA A100 40GB GPUs.

IV. DEMYSTIFYING AI AGENTS

In this section, we demystify the execution characteristics of AI agents. Section IV-A first examines an agent’s single-request execution, analyzing how iterative LLM calls and interactions with external tools affect latency and resource usage. This is followed by a detailed exploration of the LLM inference and tool-calling characteristics of agents in Section IV-B. Lastly, Section IV-C shifts the focus to the serving environment of agentic systems where multiple requests are handled concurrently, identifying system-level bottlenecks and scalability issues that emerge during the deployment of agents.

A. Overall Workflow of AI Agents

An AI agent’s workflow alternates between two main stages: LLM inference and tool use (Section II-B). This process forms a dynamic feedback loop in which the LLM determines

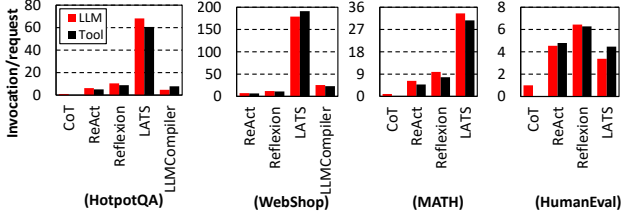


Fig. 4: Average number of LLM and tool invocations per request.

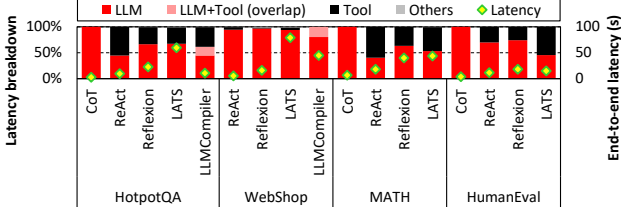


Fig. 5: Latency breakdown of agents (left axis, bar graph) and their end-to-end latency for processing a single request (right axis, diamond marker). The pink bars represent phases where LLM and tool execution latencies overlap, as observed in LLMCompiler, which asynchronously executes tools during plan generation.

the next action, invokes an external tool if necessary, and incorporates the resulting observation into the next inference step. This section analyzes the execution patterns of AI agent workflows, with a focus on their system-level characteristics.

Effect of LLM and tool calls on latency. Figure 4 shows the average number of LLM and tool invocations per request across benchmarks. While CoT performs only a single LLM inference per request, tool-augmented agentic systems—ReAct, Reflexion, LATS, and LLMCompiler—require significantly more LLM calls, averaging 9.2 times more than CoT. Among these, LATS exhibits the highest LLM invocation count, with an average of 71.0 LLM calls per request. This is primarily due to its use of tree search, which explores multiple reasoning branches (i.e., child nodes) by issuing separate LLM inferences for each one when expanding a tree node.

Figure 5 presents the end-to-end latency and the latency breakdown of each agent’s execution. While most agents exhibit a similar number of LLM and tool calls per request (Figure 4), the latency contribution from tool calls varies significantly depending on the workload. This discrepancy is primarily due to differences in the underlying tool execution latencies. For example, WebShop uses lightweight tools that interact with locally hosted webpages, resulting in tool latencies as low as 20 ms per call. In contrast, HotpotQA relies on the Wikipedia API, where individual calls take an average of 1.2 seconds. As a result, tool execution dominates the overall latency breakdown in this case. We discuss tool-calling characteristics further in Section IV-B.

On average, LLM inference and tool execution account for 69.4% and 30.2% of total latency, respectively. Both stages contribute significantly to overall latency, but they are difficult to overlap due to their sequential dependency. Specifically, the LLM output is needed to determine which tool to call and with what parameters. Conversely, the next LLM invocation typically relies on the observation returned

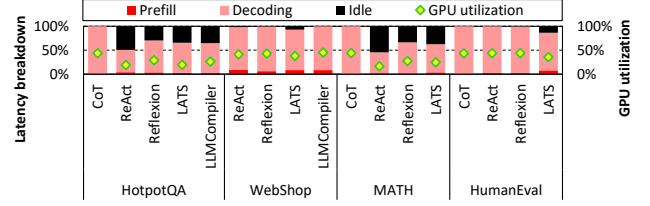


Fig. 6: Breakdown of GPU runtime by usage (left axis, bar graph) and the resulting average GPU utilization (right axis, diamond marker). GPU utilization is measured as the fraction of actively used GPU cores, using NVIDIA’s DCGM [27].

by the tool. Although LLMCompiler attempts to mitigate this dependency by streaming intermediate plans to the scheduler for asynchronous execution of tool calls (thus concurrently executing it with planning), the observed overlap accounts for only 18.2% of total latency.

Key takeaway #1: *The sequential dependency between LLM inference and tool execution creates a fundamental bottleneck in an AI agent’s execution latency, as these stages cannot be easily parallelized. To improve the end-to-end performance of AI agents, future systems should explore techniques that reduce serialization between LLM inference and tool execution. For example, asynchronous pipelines or speculative tool invocation could help overlap LLM inference with tool execution—a capability partially achieved with LLMCompiler. Optimizing either the LLM or the tool in isolation is insufficient; meaningful end-to-end latency reduction requires coordinated, system-level optimization.*

Agentic workflow’s effect on GPU compute utility. Figure 6 breaks down GPU runtime by usage and shows the resulting average GPU utilization while handling a single request. Compared to CoT, which performs a single LLM inference without any external interaction, tool-augmented AI agents exhibit significantly lower GPU utilization. This is primarily due to the tool execution phases, which introduce GPU idle periods. With the exception of HumanEval’s test generation tool—which uses the GPU to execute an LLM—the tools in our experiments generally run entirely on the local CPU or external systems (e.g., accessing the Wikipedia webpage). As a result, the GPU typically remains idle during tool-calling periods, leading to as much as 54.5% of the execution time being idle. When the GPU is executing the LLM, its activity can be further divided into the prefill and decode stages [28], [37], [67], which account for 4.7% and 74.1% of the GPU’s execution time, respectively. As noted in [1], [2], [6], [21], the decode stage is known to be memory-bound. Consequently, the large fraction of time spent in the decode stage further contributes to the underutilization of GPU resources.

Because the sequential dependency between LLM inference and tool calls limits parallel execution opportunities within a single request (i.e., intra-request parallelism), improving overall resource utilization requires leveraging *inter-request* parallelism. We explore this direction in Section IV-C, where we discuss the implications of serving AI agents over multiple queries with LLM request batching [19], [53], [66].

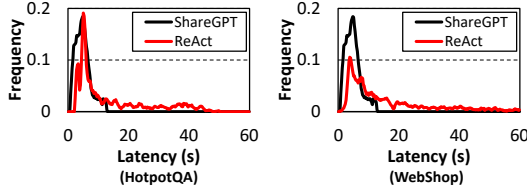


Fig. 7: Latency distribution when serving a non-agentic ShareGPT workload and a ReAct-based agent. Latency is measured while processing one request at a time, with prefix caching enabled.

Key takeaway #2: AI agents interleave GPU-bound LLM inference with CPU- or I/O-bound tool calls, resulting in non-trivial GPU idle time. This underutilization arises from the inherently sequential workflow of AI agents, which limits the potential for intra-request parallelism. To address this, agent-serving systems should adopt inter-request concurrency mechanisms to better utilize idle GPU cycles and improve overall throughput.

Latency distribution when serving AI agents. Figure 7 compares the end-to-end latency distributions of a conventional, non-agentic LLM service using ShareGPT and a ReAct-based agent system. The ShareGPT dataset represents a typical chatbot workload, where each response is generated by a single LLM inference. As shown, this results in a relatively low and consistent latency distribution, with most responses completing within 3 to 7 seconds. In contrast, the ReAct-based agent exhibits a much broader latency distribution with a heavier tail. This is due to its multi-step reasoning and reliance on external tool usage. Because the number of reasoning steps and tool calls varies across requests in AI agents, the associated computational demands also fluctuate. As a result, there is significant variance in latency across queries targeting AI agents.

Key takeaway #3: Unlike a single-pass LLM service with predictable latency, an agent’s iterative reasoning leads to highly variable, multi-step execution times, i.e., a heavy-tailed latency distribution per query. Systems must therefore be designed to handle this per-request variability. Techniques such as adaptive scheduling or elastic resource allocation can potentially help maintain efficiency and responsiveness, even as reasoning depth varies significantly across queries.

B. LLM Inference and Tool-Calling Characteristics

This section further analyzes the behavior of agentic systems by characterizing the properties of LLM inference and tool calls within the AI agent in greater detail.

Breakdown of input and output tokens in LLM inference. Figure 8 presents the token count distribution across different AI agents. *Instruction* tokens define the agent’s role and objective within the task, while *Few-shot* tokens provide in-context examples that guide the agent’s behavior. *User* tokens represent user queries. *LLM history* and *Tool history* tokens consist of accumulated outputs from previous LLM inferences and tool responses across iterations. *Output* tokens

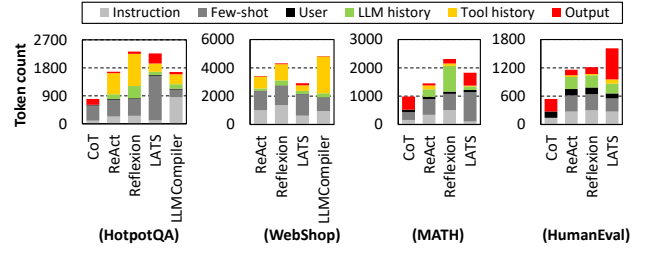


Fig. 8: Breakdown of input and output tokens in LLM inference. *Instruction* and *Few-shot* (light and dark gray) represent input tokens that are statically fixed as part of the initial prompt to the LLM. *User* (black) denotes input tokens provided by the user as part of the query. *Output* (red) refers to tokens generated by each LLM call. *LLM history* (green) and *Tool history* (yellow) represent tokens accumulated from previous LLM outputs and tool responses, respectively, which are then included as input tokens during the next LLM call.

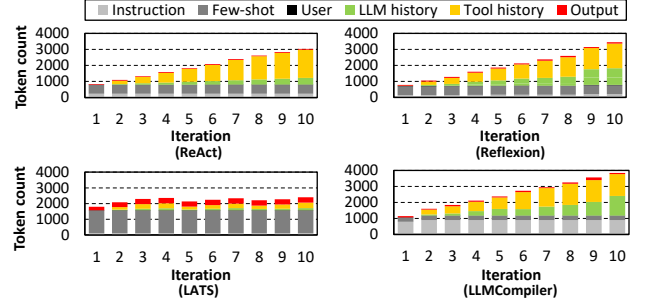


Fig. 9: Breakdown of token count per each AI agent’s iterative reasoning step for HotpotQA. While *Instruction* and *Few-shot* tokens remain constant, the accumulation of *LLM history* and *Tool history* tokens significantly increases the input context length over time.

are generated at each LLM inference step, while the remaining tokens collectively make up the input prompt.

Compared to CoT, other AI agents consume more input tokens but generate fewer output tokens per LLM call. This is because these agents decompose their reasoning into multiple steps, invoking the LLM several times and incorporating tool usage between LLM calls. As a result, the total output is spread across multiple LLM invocations. In contrast, CoT completes its reasoning in a single LLM call, producing a longer, continuous output in just a single step. Among the AI agents, LATS stands out for generating notably longer output sequences. This is because it produces multiple output samples from a single input to generate candidate expansions when growing a tree node.

Token usage patterns also vary depending on the task workload. In knowledge-intensive tasks such as HotpotQA and decision-making tasks like WebShop, tool calls often return large responses (e.g., the full content of a webpage) resulting in longer tool history tokens. In contrast, tasks that rely more heavily on internal reasoning, such as MATH and HumanEval, tend to produce longer LLM-generated outputs, leading to larger LLM history tokens.

Although the ratio of LLM and tool history tokens varies across workloads, most benchmarks exhibit substantial growth in input history over multiple iterations. An exception is LATS, which includes only the path from the root to the

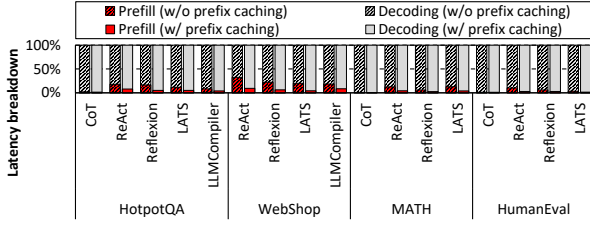


Fig. 10: Breakdown of LLM inference latency with and without prefix caching enabled.

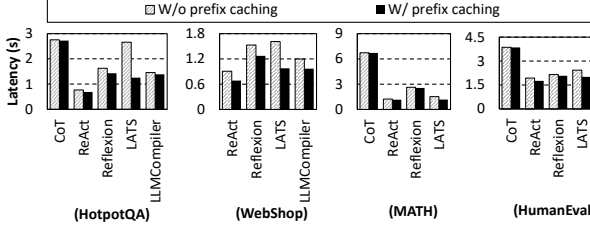


Fig. 11: LLM inference latency with and without prefix caching.

current node, rather than concatenating all prior interaction histories. As shown in Figure 9, initial inputs are typically around 1,000 tokens, but the input size increases to 3–4 \times as prior LLM outputs and tool responses are appended to the input context of subsequent LLM calls. Because histories accumulate sequentially, consecutive LLM calls share common prefixes in their input contexts. These long input contexts result in high KV cache usage per request and considerable prefix overlap across iterations. This behavior presents an opportunity to improve GPU compute and memory efficiency through *prefix caching* [19], as detailed below.

Key takeaway #4: *AI agents accumulate long input contexts over iterative reasoning steps by appending prior LLM outputs and tool responses. This behavior results in large KV cache footprints and increasing GPU memory usage. However, because many of these tokens are shared across iterations, system-level optimizations such as prefix caching are needed to reduce redundant computation and help alleviate memory overhead in multi-step reasoning workloads.*

Effect of prefix caching on AI agent’s compute efficiency.

Building on the token-level analysis above, we now turn to system-level characteristics, starting with GPU compute efficiency. AI agent workloads involve multiple iterative LLM calls, where a large portion of the input context is reused at each step. Prefix caching leverages this shared prefix to skip redundant computation during the prefill phase by reusing previously cached key-value (KV) pairs.

Figure 10 shows the proportion of prefill and decoding latency during LLM inference, with and without prefix caching. For CoT, LLM inference occurs only once per request, and the shared prefix across inferences is minimal. Moreover, CoT typically generates a relatively large number of output tokens, making decoding the dominant contributor to latency. In contrast, AI agents operate iteratively and accumulate long input contexts due to interaction histories. As a result, prefix caching reduces prefill latency by an average of 58.6%, demonstrating

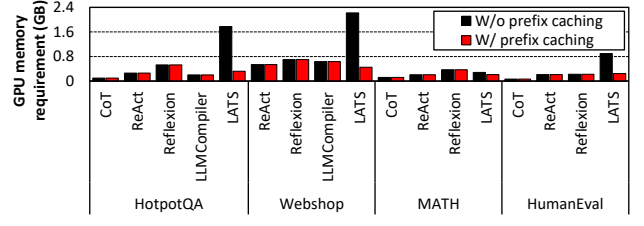


Fig. 12: Average GPU memory requirement for KV cache per AI agent request, with and without prefix caching.

its effectiveness in improving compute efficiency by avoiding redundant computations through prefix reuse.

As shown in Figure 11, the overall impact of prefix caching on end-to-end LLM inference latency varies significantly by workload type. In CoT-style prompting, the decoding phase dominates execution time—a characteristic common to virtually all static reasoning models (Figure 1(a,b))—leaving the prefill phase as only a small portion of the total. As a result, prefix caching offers limited latency reduction for CoT. In contrast, agentic workloads exhibit a non-negligible prefill phase (highlighted by the red bars in Figure 10) due to the accumulation of long input contexts over iterative steps. This makes them more amenable to performance gains from prefix caching, which eliminates redundant computation in the prefill stage and yields an average 15.7% reduction in end-to-end latency. While this per-request improvement may seem modest, the reduction in prefill time can significantly alleviate system-level bottlenecks. In token-level schedulers like vLLM, long prefill phases can delay the scheduling of concurrent requests. By shortening these phases, prefix caching can improve scheduling efficiency and increases overall system throughput. This effect is examined further in Section IV-C (Figure 15).

Key takeaway #5: *Prefix caching reduces redundant prefill computation across iterative LLM calls, improving per-call compute efficiency and shortening scheduling-critical prefill phases. However, because decoding dominates overall latency and is less amenable to caching, the net reduction in end-to-end inference time remains modest. Therefore, system-level optimizations such as speculative token generation or leveraging reduced prefill time to increase batching opportunities are essential complements to caching for improving overall performance.*

Effect of prefix caching on AI agent’s memory efficiency.

While the previous analysis demonstrated how prefix caching improves compute efficiency by bypassing prefill operations for shared prefix tokens, we now examine its impact on GPU memory requirements by measuring the average GPU memory required to store the KV cache (Figure 12). On average, tool-augmented AI agents consume 3.0 \times more memory per request than CoT, and up to 5.4 \times more in the worst case—even with prefix caching. This overhead arises from the iterative nature of agent workflows, where each LLM call appends intermediate reasoning steps and tool responses to the context, resulting in an increasingly large input for each LLM inference.

These results highlight the need for memory optimization

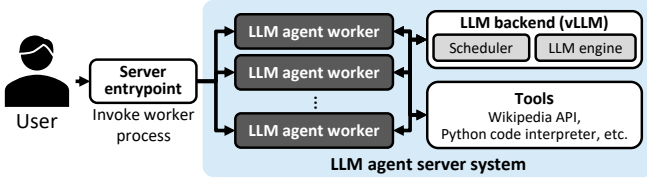


Fig. 13: High-level overview of our AI agent serving system.

in AI agent workloads, with prefix caching serving as a key technique for reducing GPU memory usage. In LATS, multiple LLM inferences are issued in parallel to evaluate several child nodes simultaneously during tree expansion. Without prefix caching, each of these parallel calls creates its own KV cache, resulting in significant memory overhead due to redundancy. With prefix caching, the shared prefix across these parallel calls can be reused, reducing memory requirements by an average of 64.8% in LATS. For other agents, where all LLM calls are invoked sequentially, prefix caching does not reduce memory usage *within a single request*, since the KV cache cannot be shared across LLM calls. However, in serving scenarios with concurrent requests, prefix caching can significantly improve memory efficiency by reusing the KV cache across requests. We further explore this serving-level memory efficiency in Section IV-C (Figure 16).

Key takeaway #6: *Iterative reasoning in tool-augmented AI agents substantially increases GPU memory usage due to the accumulation of long input contexts across multiple LLM calls. This highlights the need for memory-optimized system designs that enable context reuse through shared KV cache and potentially leverage specialized hardware or heterogeneous memory hierarchy for efficient context storage and retrieval (e.g., offloading all or parts of KV cache contexts to CPU memory or SSD).*

C. AI Agent Serving Characteristics

So far, our characterization has focused on the behavior of AI agents when servicing a single query for a specific task. In this section, we shift our attention to system-level properties of AI agent serving environments, analyzing scenarios where multiple requests are routed to the server and can be processed concurrently for high serving throughput. Unlike static reasoning models that process a user request with a single LLM inference step, AI agents perform multiple reasoning steps iteratively, introducing new challenges for efficient serving.

To examine the characteristics of AI agent serving, we implement an agent serving system, as illustrated in Figure 13. When a user sends a request to the agent server’s entry point, each worker processes the request according to the agent’s workflow. Depending on the current step of the task, a worker either sends a request to the LLM inference server or executes a tool. Tool execution may occur locally (e.g., code interpreters, custom functions) or involve external resources (e.g., web search, API calls). Each worker operates asynchronously, and LLM inference requests from multiple workers can be batched at the LLM backend (e.g., vLLM) for high-throughput

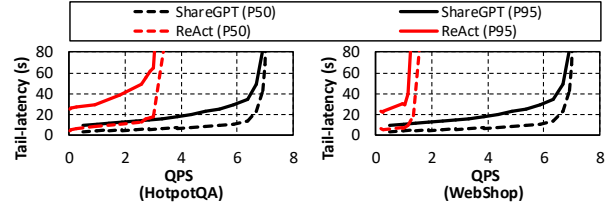


Fig. 14: 50th and 95th percentile latencies for chatbot (ShareGPT) and AI agent (ReAct) workloads as queries per second (QPS) rates increase, with prefix caching enabled.

processing using continuous batching [19], [66]. We adopt vLLM’s default first-come-first-served (FCFS) scheduler in the LLM inference backend. To simulate realistic traffic, input queries to the agent server are randomly sampled and issued to the server following a Poisson arrival distribution [26].

Importance of concurrent request scheduling. Before comparing AI agent serving against conventional static reasoning based chatbot (ShareGPT) serving scenarios, we first highlight the importance of concurrently servicing AI agent requests. When ReAct agents are executed *sequentially*, the average latency is 9.6 seconds for HotpotQA and 5.3 seconds for WebShop, limiting throughput to 0.10 and 0.19 QPS, respectively. With *concurrent execution*, throughput improves to 2.6 and 1.2 QPS for HotpotQA and WebShop, respectively, achieving 25 \times and 6.2 \times gains at the cost of a 2.1 \times increase in average latency. The greater throughput gain in HotpotQA arises from its longer tool latency, which causes the GPU to remain idle for extended periods. These idle intervals can be effectively utilized by executing other requests, enabling higher concurrency and improved throughput.

Comparison with conventional static reasoning LLM services. We now compare AI agent serving with a conventional LLM serving scenario, represented by the chatbot (ShareGPT) workload. ShareGPT, a typical single-turn LLM service, processes user queries in a single inference pass. Figure 14 shows the changes in end-to-end tail latencies for chatbot (ShareGPT) and AI agent (ReAct) workloads as input queries per second (QPS) to the server increase. The peak throughput is measured as the maximum sustainable QPS at the knee of the tail latency curve. As depicted, the peak throughput of ReAct is significantly lower than that of ShareGPT. While ShareGPT can sustain up to 6.4 QPS, ReAct supports only 2.6 QPS on HotpotQA and 1.2 QPS on WebShop. This limitation stems from ReAct’s multi-step reasoning, where each request involves multiple LLM calls and tool interactions, significantly increasing latency.

Another point worth noting is that ReAct’s 95th percentile latency increases much more sharply with load. For every 1 QPS increase from 50% to 100% of peak throughput, the 95th percentile latency rises by 17.8 seconds on HotpotQA and 6.1 seconds on WebShop, whereas ShareGPT’s latency increases by only 0.9 seconds. This steep increase in tail latency highlights AI agent’s high sensitivity to load and the difficulty of maintaining low tail latency under heavy traffic.

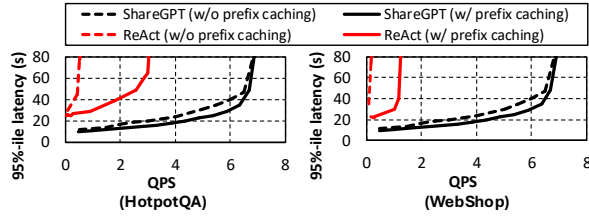


Fig. 15: 95th percentile latency for chatbot (ShareGPT) and AI agent (ReAct) workloads as QPS rates increase, with (solid line) and without (dashed line) prefix caching enabled.

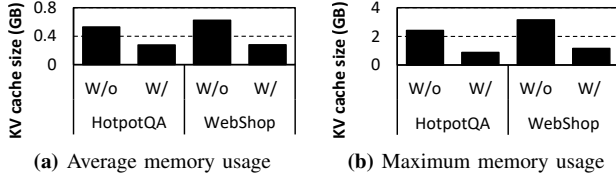


Fig. 16: (a) Average and (b) maximum memory used in allocating KV caches, with and without prefix caching enabled. Evaluation is conducted at 0.2 QPS (HotpotQA) and 0.1 QPS (WebShop) data points using ReAct.

Overall, these observations underscores the need for careful scheduling and resource management when serving AI agents, especially compared to conventional LLM services.

Key takeaway #7: *Compared to conventional LLM services, AI agent serving experiences lower throughput and significantly higher sensitivity to tail latency due to their dynamic execution patterns and iterative reasoning. This heightened sensitivity underscores the need for agent-specific scheduling and resource management strategies, such as agent-aware request dispatching or adaptive compute resource allocation per request, to maintain quality of service (QoS).*

Effect of prefix caching on AI agent serving throughput.

Prefix caching is an important system-level optimization that reduces redundant computation during the prefill phase of LLM inference by reusing previously computed key-value (KV) caches. While its impact on the latency of individual LLM calls is modest, it can substantially improve throughput and overall serving efficiency, particularly in AI agent serving.

Figure 15 compares the effect of prefix caching on chatbot (ShareGPT) and agentic (ReAct) workloads. ShareGPT shows only a modest $1.03\times$ throughput improvement, as it performs a single LLM call per request with minimal repetition. In contrast, ReAct benefits significantly, achieving an average $5.62\times$ increase in throughput. This is because agent workloads involve multiple LLM calls per request, amplifying the benefits of avoiding redundant prefill operations.

The performance gap is further explained by token-level batching systems such as vLLM. Without prefix caching, long prefill stages occupy the GPU and block decoding for other requests, leading to system-wide queuing delays. This bottleneck is particularly problematic for AI agents, where repeated LLM calls per request exacerbate inter-request contention. As a result, prefix caching plays a critical role in mitigating these interference effects and improving overall serving efficiency,

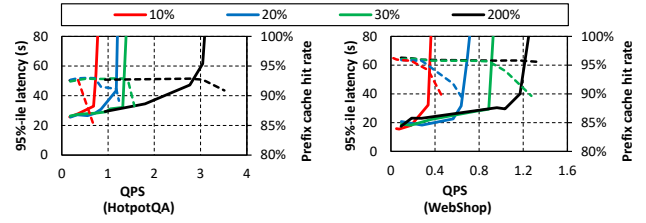


Fig. 17: 95th percentile latency (left axis, solid lines) and prefix cache hit rate (right axis, dashed lines) when the GPU memory reserved for KV cache allocation is changed. Legends denote the GPU memory reserved for KV cache allocation, the value of which denotes the reserved memory size relative to the LLM model weight size.

especially for agentic workloads.

Key takeaway #8: *Prefix caching significantly improves serving efficiency for AI agents by reducing prefill overhead and inter-request interference, resulting in much greater throughput gains than in conventional LLM workloads. Therefore, incorporating sophisticated caching mechanisms such as solutions that persist and reuse prefixes across queries is a promising direction for sustaining high throughput in complex, iterative reasoning tasks.*

Effect of prefix caching on AI agent’s memory usage.

We now investigate the impact of prefix caching on GPU memory efficiency in AI agent serving, focusing specifically on its effect on key-value (KV) cache size, one of the most significant contributors to memory usage in LLM inference. Figure 16 shows the GPU memory consumed in allocating KV caches, with and without prefix caching enabled, under identical QPS conditions. With prefix caching enabled, the average and maximum KV cache memory usage decrease by 51.7% and 63.5%, respectively, indicating improved memory efficiency. This reduction arises from the ability of prefix caching to reuse key-value pairs of shared prefix tokens across multiple LLM invocations across AI agent requests. Thus, prefix caching not only improves compute efficiency by eliminating redundant prefill operations but also reduces the KV cache memory footprint, enabling more efficient utilization of GPU memory during AI agent serving.

Next, we analyze how the effectiveness of prefix caching and the serving performance of AI agents are impacted by the GPU memory capacity available for KV cache allocation. Figure 17 illustrates how varying the amount of GPU memory reserved for the key-value (KV) cache affects serving throughput. As shown in the figure, reducing the KV cache allocation pool significantly limits throughput, as the system cannot accommodate many concurrent requests. Compared to the 200% configuration, the maximum sustainable throughput drops by 86.3% under the 10% configuration and by 73.6% under the 20% configuration. Because the opportunity to batch multiple requests is limited under these two configurations, both exhibit noticeable queuing delays, as request scheduling becomes increasingly serialized. Under the 30% configuration, the KV cache allocation pool is just large enough to enable batching and minimize queuing delays; however, concurrent requests still compete for limited KV cache. This results in

cache thrashing, which reduces the effectiveness of prefix reuse. As a result, the 30% configuration achieves 35% and 18% lower throughput for HotpotQA and WebShop, respectively, compared to the 200% configuration. This demonstrates that even moderate constraints in GPU memory can critically limit the serving performance of AI agents.

Key takeaway #9: *Prefix caching improves GPU memory efficiency in AI agent serving by reusing shared prefixes across requests. However, due to the long and growing contexts in agent workloads, limited KV cache capacity can lead to cache thrashing and sharply degrade throughput, even under moderate memory constraints. This highlights the need for more intelligent cache management strategies, such as locality-aware hierarchical caching or KV cache compression techniques, that helps prevent cache saturation and sustain performance under concurrent agent requests.*

V. DEMYSTIFYING TEST-TIME SCALING IN AI AGENTS

In the previous section, we characterized the system-level behavior of AI agents by analyzing their workflow patterns, GPU resource utilization, and serving characteristics. We now shift our focus to the diverse design space of AI agents and examine their test-time scaling behavior to understand the trade-offs between model accuracy and deployment cost. Section V-A first explores the cost-efficiency of various AI agent designs by quantifying how differences in design parameters affect both accuracy and cost-effectiveness. Section V-B then analyzes the test-time scaling behavior of AI agents by comparing multiple scaling strategies and evaluating their impact on accuracy and compute cost. To assess each configuration, we used a benchmark of 50 sample questions and measured the average accuracy and compute cost for each.

A. Analyzing Cost-Efficiency Across AI Agent Design Spaces

Deploying AI agents in practical settings requires careful configuration of agentic system parameters. These design choices significantly affect not only the agent’s task success rate but also the overall cost of operating such systems. In this section, we quantify how different parameter configurations in AI agents influence both accuracy and cost-efficiency.

Pareto analysis of accuracy and cost across AI agent designs. Figure 18 presents the trade-off between accuracy and cost across various AI agent configurations. Each point corresponds to a specific design variant, such as changes to the number of few-shot examples or maximum iteration limits.

Figure 18(a) shows the trade-off between accuracy and latency. ReAct demonstrates strong compute efficiency across all benchmarks, achieving moderate accuracy with consistently low latency. Reflexion builds on ReAct by introducing reflection steps guided by internal or external rewards. This approach yields modest accuracy improvements but significantly increases latency. LATS extends Reflexion with a tree-based reasoning approach that explores multiple candidate branches at each step. While this leads to higher accuracy, it also introduces substantial computational overhead due to the expansion of reasoning paths. LLMCompiler, with its

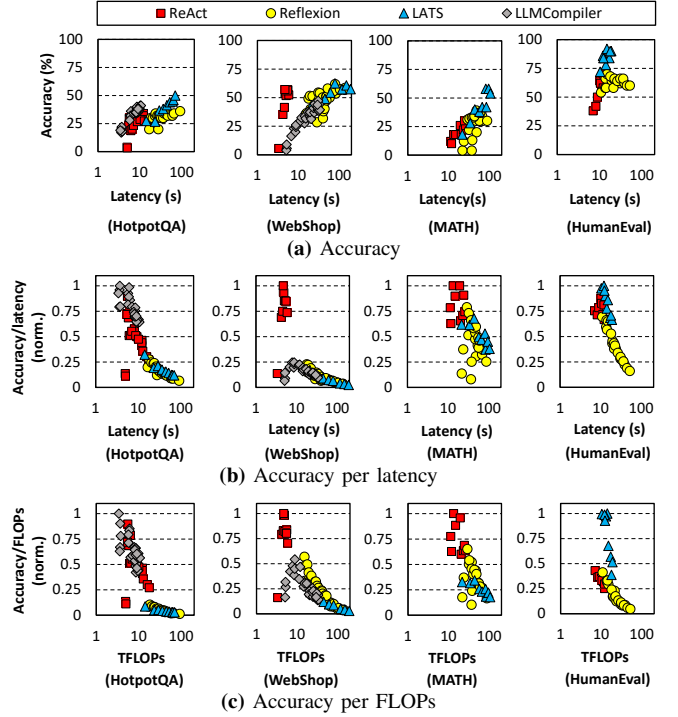


Fig. 18: Accuracy and cost-efficiency of AI agent design points. (a) Accuracy vs. end-to-end latency. (b) Accuracy per latency, and (c) Accuracy per FLOPs. (b) and (c) illustrates how efficiently each configuration translates compute cost into task performance.

planning-based architecture, outperforms ReAct on tasks like HotpotQA in both accuracy and cost-efficiency, thanks to its ability to generate and execute structured plans in parallel. However, in tasks such as WebShop—where tool usage involves high interdependencies (e.g., searching or clicking on a webpage)—its DAG-style planning results in unnecessary tool invocations, leading to lower efficiency than ReAct.

Figure 18(b) and (c) illustrate the cost-efficiency of various agent configurations. We define cost-efficiency as the ratio of accuracy to cost, where cost is measured either as end-to-end latency (Figure 18(b)) or estimated FLOPs (Figure 18(c)). This metric reflects how effectively each configuration translates compute resources into task accuracy. Across all agents and workloads, we observe a consistent pattern: *as computation cost increases, accuracy improves, but with diminishing returns*. This highlights the importance of careful system design. Instead of optimizing purely for accuracy, it is critical to identify configurations near the Pareto frontier, where the balance between accuracy and cost is most optimal.

Tuning Iteration and Prompting for Cost-Efficient Agent Behavior. To better understand the accuracy–cost trade-offs in AI agent design, we analyze how two key parameters in AI agent designs affect model performance: the maximum iteration budget and the number of few-shot examples.

Figure 19 shows how varying the iteration budget impacts average latency, 95th percentile latency, and accuracy. The iteration budget controls how many reasoning steps and tool invocations the agent is allowed per query. As this budget

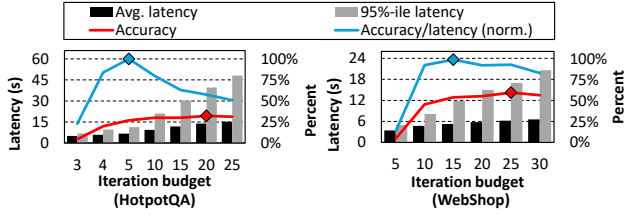


Fig. 19: Latency and accuracy trends under iteration budget constraints in ReAct. As the maximum iteration budget increases, both accuracy and average latency initially improve but eventually saturate, while 95th percentile latency continues to grow. Markers indicate the points of maximum accuracy (red diamond) and peak cost-efficiency (blue diamond), as measured by accuracy-to-latency ratio.

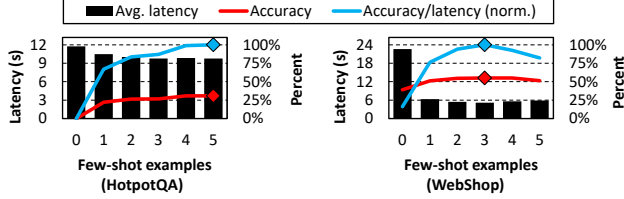


Fig. 20: End-to-end latency and accuracy trends with varying numbers of few-shot examples in ReAct. With more examples, accuracy initially improves while latency decreases, but both metrics eventually saturate. Markers indicate the configuration with the highest accuracy (red diamond) and the point of peak cost-efficiency (blue diamond), based on normalized accuracy-to-latency ratio.

increases, agents can perform deeper reasoning, which initially improves accuracy. However, both accuracy and average latency eventually saturate, while the 95th percentile latency continues to increase linearly. This rising tail latency is driven by a small set of outlier tasks that consume the full iteration budget. These outliers degrade cost-efficiency by contributing disproportionately to total compute usage without yielding substantial accuracy gains. The widening latency distribution also reduces predictability, which is especially problematic for latency-sensitive deployments. Therefore, iteration limits should be tuned not only for performance but also for latency consistency and operational stability.

Figure 20 shows how varying the number of few-shot examples in the prompt affects latency and accuracy. Initially, adding examples substantially improves accuracy, as agents gain better task understanding. However, beyond a certain point, the benefit diminishes—and in some cases, accuracy declines due to prompt length exceeding the model’s optimal processing range. Interestingly, average latency decreases as more examples are added. This counterintuitive result arises because good examples help agents solve tasks in fewer steps, offsetting the cost of longer prompts. Thus, while longer prompts marginally increase per-token processing time, the reduction in overall reasoning steps often leads to net latency savings. In summary, a small number of carefully chosen examples can improve both accuracy and efficiency, while excessive prompting may lead to diminishing returns.

To identify optimal configurations, we highlight the point at which the accuracy-to-latency ratio is maximized (denoted by blue markers in Figure 19 and Figure 20). This point represents

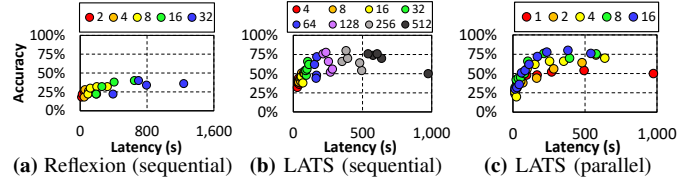


Fig. 21: Accuracy–latency trade-offs under sequential and parallel test-time scaling on HotpotQA. Legends denote the scaling level: maximum reflection steps in (a, b), and number of child nodes per expansion in (c).

the most cost-effective trade-off between model accuracy and response time. Such metrics provide a practical guideline for setting iteration budgets and few-shot prompting under latency or compute constraints.

Key takeaway #10: Design choices in AI agents, such as increasing iteration depth or adding few-shot examples, can improve accuracy, but often show diminishing returns or even regressions beyond a threshold. These improvements also come with significant latency costs. To ensure cost-effective operation, AI agent serving systems should aim to maximize accuracy per unit of compute by selecting configurations that balance quality and responsiveness under real-world constraints.

B. Test-Time Scaling of AI Agents

AI agents can dynamically scale their reasoning at test time by adjusting the number of reasoning steps based on task difficulty. This flexibility helps improve performance on complex problems, but it also introduces significant variation in computation cost. Designing systems that are both accurate and efficient requires a deeper understanding of how inference behavior evolves as compute usage increases.

Sequential vs. parallel reasoning at test time. We investigate the effect of two key forms of test-time scaling for AI agents: *sequential* and *parallel*. In *sequential scaling*, the agent gradually increases its reasoning steps over time, allowing for deeper introspection. This is typical of agents like Reflexion and LATS, where the number of reflection steps can be adjusted dynamically. In contrast, *parallel scaling* issues multiple reasoning branches simultaneously, commonly through parallel LLM calls, to explore diverse solution paths. LATS uses this approach by spawning multiple child nodes during each tree expansion step.

Figure 21(a) and (b) show the accuracy–latency trade-offs for Reflexion and LATS under sequential scaling. Both methods improve in accuracy with more reflection steps, but with diminishing returns. For example, in Reflexion, increasing latency from 16.9s to 25.6s yields a 4% accuracy gain. However, achieving the same model accuracy improvement from a later point (56.0s) requires a much larger increase in latency (269.5s), a $31\times$ higher cost for the same marginal gain.

On the other hand, parallel scaling exhibits a different trade-off. Figure 21(c) highlights the behavior under parallel scaling in LATS. Increasing the number of child nodes from 1 to 16

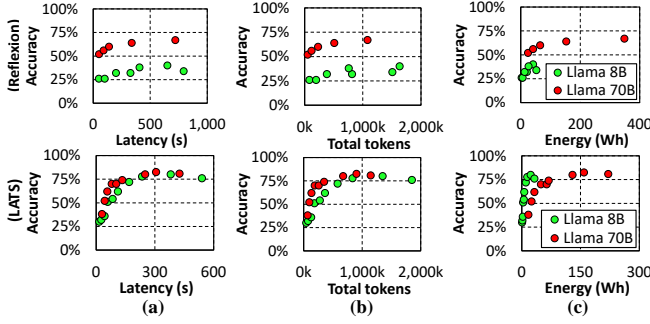


Fig. 22: Accuracy–cost trade-offs under test-time scaling across two model sizes (Llama-3.1-Instruct 8B and 70B) on HotpotQA. (a)–(c) compare Reflexion (top row) and LATS (bottom row) across latency, token usage, and energy consumption. While 70B achieves higher accuracy with fewer steps, the 8B model, especially when paired with parallel scaling, can approach 70B performance with lower energy cost. Each point corresponds to a different level of test-time reasoning.

improves accuracy by 14.4 percentage points while simultaneously *reducing* latency by 196.3s on average. This is because evaluating multiple reasoning paths in parallel helps the agent converge on high-quality answers more quickly. However, this comes at the cost of issuing more concurrent LLM requests, which increases memory pressure and may limit scalability in multi-tenant or resource-constrained environments.

These results suggest that AI agent configurations should align with system constraints such as latency budgets and available compute resources. Parallel scaling is effective for latency-sensitive workloads, as it allows the agent to explore multiple reasoning paths at once and reach better answers faster. However, it increases resource usage due to the large number of concurrent LLM calls. In contrast, sequential scaling is better suited for resource-constrained environments. This approach avoids concurrent LLM calls, lowering peak resource demand, but incurs higher latency from step-by-step reasoning.

Model size effects on test-time scaling. We further analyze how model size affects the accuracy–cost trade-offs under different test-time scaling strategies.

Figure 22(a) shows that both the 8B and 70B Llama-3.1-Instruct [24], [25] models eventually reach saturation in accuracy, but they differ in how quickly they reach this point. The 70B model achieves high accuracy with relatively low latency, whereas the 8B model requires much longer inference times to reach similar performance. This trend is echoed in Figure 22(b), which plots total token usage. The 8B model consumes significantly more tokens at high-accuracy settings, indicating it needs more reasoning iterations to match the 70B model’s performance. However, as shown in Figure 22(c), the 8B model is substantially more energy-efficient. While the 70B model relies on 8 A100 GPUs, the 8B model runs on just one, resulting in lower total energy consumption per request, even when requiring more reasoning steps to be involved.

Interestingly, the performance gap between models can be partially closed with effective scaling strategies. Reflexion (which uses sequential scaling) shows limited accuracy on the 8B model. But with LATS and parallel scaling, the 8B model

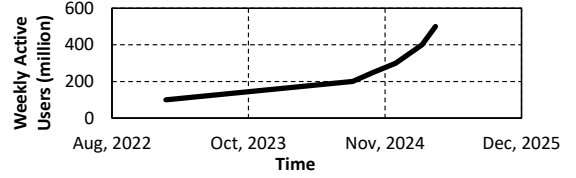


Fig. 23: Growth trend in weekly active users of OpenAI’s ChatGPT. The data shows a marked acceleration in adoption, with the user base surpassing 500 million by April 2025 [31], [35], [36], [39]–[41].

achieves near-70B performance by exploring multiple paths and selecting the best one. This shows that test-time strategy can play a compensatory role in low-resource settings.

Key takeaway #11: *AI agents scale their reasoning at test time either by deeper sequential thinking or by exploring multiple reasoning paths in parallel, and each strategy imposes distinct trade-offs in latency and resource usage. Sequential scaling favors lower peak resource usage but incurs longer runtime, while parallel scaling accelerates inference at the cost of increased compute and memory demand. System designers should match the appropriate scaling strategy to deployment constraints, potentially combining both for optimal cost-effectiveness.*

VI. AI INFRASTRUCTURE IMPLICATIONS OF AGENTIC TEST-TIME SCALING

To assess the system-level impact of agentic test-time scaling, we quantify GPU energy consumption and datacenter-wide power demands of AI agents. Following the methodology in Section V-B, we utilize Reflexion and LATS as representative AI agents employing sequential and parallel scaling, respectively (Table III).

Energy consumption (per query). Using Llama-3.1-Instruct 8B and 70B as backend LLMs, Reflexion consumes 41.53 Wh and 348.41 Wh per request, whereas LATS consumes 22.76 Wh and 158.48 Wh, respectively. By contrast, a conventional single-turn LLM inference based on ShareGPT requires only 0.32 Wh (8B) and 2.55 Wh (70B) per request. These figures correspond to a $62.1\times$ – $136.5\times$ increase in GPU energy per query under agent-based test-time scaling.

Based on recent estimates, ChatGPT serves roughly 500 million weekly active users (WAU) [35] (Figure 23), which translates to about 71.4 million daily active users (DAU). Assuming, conservatively, that each user submits only one agentic query per day, Reflexion’s daily GPU energy footprint would be roughly 2.97 GWh for the 8B model and 24.89 GWh for the 70B model. While our analysis does not consider LLM request batching [19], [66], our estimate is still conservative because (1) it reflects current DAU with only one query per user, even though AI adoption continues to accelerate often with much heavier per-user usage, and (2) it counts only GPU energy, omitting CPU, memory, networking, storage, and cooling overheads.

Even under these modest assumptions, the projected demand rivals the daily electricity consumption of Seattle and its surrounding area (24.8 GWh) [43]. As AI agents become

		Accuracy (%)	Latency (seconds)	Energy (Wh/query)	Power @ 71.4 Million Queries/day (Watt)	Power @ 13.7 Billion Queries/day (Watt)
8B	ShareGPT	–	4.23 (1×)	0.32 (1×)	1.0 M	182.7 M
	Reflexion	38	649.34 (153.7×)	41.53 (130.9×)	123.6 M	23.7 G
	LATS	80	380.90 (90.1×)	22.76 (71.7×)	67.7 M	13.0 G
70B	ShareGPT	–	6.40 (1×)	2.55 (1×)	7.6 M	1.5 G
	Reflexion	67	720.00 (112.6×)	348.41 (136.5×)	1.0 G	198.9 G
	LATS	82	305.67 (47.8×)	158.48 (62.1×)	471.5 M	90.5 G

TABLE III: Energy and power demands of handling an AI agent service request on HotpotQA. We report accuracy, latency, GPU energy consumption, and datacenter-wide power demand under current and future traffic scenarios (71.4 Million Queries/day and 13.7 Billion Queries/day) for two agentic workflows (Reflexion and LATS) using Llama-3.1-Instruct 8B and 70B models. ShareGPT serves as the baseline for conventional single-turn LLM inference. Numbers in parentheses denote the multiplicative increase relative to ShareGPT. Reflexion and LATS design points were selected based on the highest-accuracy configurations in Figure 22. The datacenter-wide power is computed by $P = (\text{Wh/query}) \times (\text{Queries/day}) / (24 \text{ hours})$.

increasingly embedded in everyday applications, their query volume could approach, or exceed, that of traditional search engines. For instance, Google Search processes over 13.7 billion queries per day [12], roughly $192\times$ the 71.4 million agentic queries assumed above. If this growth in user base and usage persists, AI-infrastructure demand could rise dramatically, potentially exceeding sustainable limits and underscoring the significant challenges posed by test-time scaling.

Datacenter-wide power demands. We now move on to estimating the datacenter-wide power requirements to sustain the aforementioned AI service demands, assuming today’s (ChatGPT’s current 71.4 million queries per day, assuming one query per user) and tomorrow’s (Google search’s 13.7 billion queries per day) AI traffic. The last two columns of Table III translate the per-query energy consumption numbers into datacenter-level power requirements. Under today’s 71.4 million DAU load, single-turn ShareGPT (70B) requires roughly 7.6 MW, well within the tens-of-megawatts envelope typical of modern datacenters [15], [59]. However, assuming similar traffic levels for AI agents, even the lighter 8B-based agents demand 67.7–123.6 MW, comparable to the power draw of a mid-sized U.S. city, while 70B-based agents approach 1 GW, nearly three orders of magnitude higher than the single-turn LLM baseline. Strikingly, this gigawatt-scale power requirement aligns with the announced budget for OpenAI’s multi-gigawatt Stargate facility [32], which is intended to support *future* AI model deployments. Yet, our analysis suggests that such infrastructure may already be necessary to support agentic systems under today’s traffic levels. Overall, our estimates indicate that even modest user traffic (on the order of tens of millions of queries per day) becomes gigawatt-scale once per-query energy exceeds ~ 100 Wh, a threshold representative of current agentic workloads.

If we were to scale the same per-query figures to Google’s 13.7 billion daily searches, the power numbers would raise single-turn ShareGPT (70B) to 1.5 GW and Reflexion (70B) to nearly 200 GW, far beyond any announced datacenter project and exceeding the power budgets of many national grids. To put this number into perspective, a 200 GW is almost half of the *entire* U.S. grid’s average load (which amounts to $4,178 \times 10^3 \text{ GWh} / (365 \times 24 \text{ hours}) = 476.9 \text{ GW}$ [52]), a scale usually discussed only for nation-wide decarbonization plans, not for a single industry or technology, one that fundamentally reshapes generation, transmission, and sustainability planning.

Sustainability challenges of agentic test-time scaling. Collectively, our findings show that AI agent performance does not scale proportionally with the associated compute, energy, and power costs. Once accuracy saturates, additional test-time scaling yields diminishing returns while imposing substantial system-level burdens. This cost inefficiency is not merely theoretical; it poses concrete constraints on real-world deployments. For instance, OpenAI’s recently introduced Deep Research system [34], designed for complex multi-step reasoning, can take up to 30 minutes per request [33]. To keep infrastructure costs manageable, OpenAI limits usage to 25 runs every 30 days for ChatGPT Plus users and 250 runs for Pro users [33]. These limits highlight the financial and computational challenges of sustaining AI systems that rely heavily on intensive test-time computation.

Based on these findings, we argue that building scalable and sustainable AI agents requires moving away from unconstrained test-time scaling. Instead, AI agents should be designed with compute-aware agentic workflows that deliver strong performance through efficient inference, rather than single-handedly relying on extended reasoning depth.

VII. RELATED WORK

AI agent workflows. Recent advances in LLM-based AI agents have introduced diverse workflows that combine language-based reasoning with external tool use. ReAct [64] interleaves reasoning and tool invocation through step-by-step decision making, while Reflexion [47] enhances agent behavior through self-evaluation and feedback. LATS [68] employs a tree-based search strategy to explore multiple reasoning paths in parallel, and LLMCompiler [18] plans tool usage in advance by generating action sequences. Building on these single-agent frameworks, multi-agent systems extend agent capabilities through collaborative task execution. CAMEL [22] implements role-based dialogue among agents, and AutoGen [58] formalizes multi-agent orchestration via structured roles, message passing, and execution control.

While these workflows substantially improve reasoning capabilities and behavioral flexibility, their system-level implications, such as resource usage, latency dynamics, and energy consumption, remain underexplored. To the best of our knowledge, this work is the first to address this gap through a comprehensive system-level analysis of representative AI

agent designs, offering new insights into the efficiency and scalability of agentic systems.

Structured agent interfaces for tool-augmented reasoning. In parallel with behavioral advancements in AI agents, recent efforts have focused on standardizing AI agent APIs and protocols to facilitate broader integration and deployment. OpenAI’s function-calling interface [30] defines a structured mechanism for API invocation, enabling agents to interact with tools in a verifiable and consistent manner. Anthropic’s Model-Context-Protocol (MCP) [3] further formalizes how agents manage context and interact with tools, enabling decoupled reasoning logic and modular deployment across platforms. Google’s Agent-to-Agent (A2A) protocol [13] complements these efforts by specifying a standard for multi-agent communication. A2A enables heterogeneous agents to exchange structured messages, delegate tasks, and coordinate behavior, supporting modular and scalable agentic systems.

Although these efforts focus on unifying the agent programming model, our work takes an orthogonal system-level perspective, uncovering the AI infrastructural challenges posed by agentic workloads under test-time scaling.

System-level optimization of AI agents. Several recent studies have proposed system-level techniques to improve the runtime performance of AI agents by restructuring their execution workflows and addressing latency bottlenecks. LLMCompiler [18], Alto [42], and Teola [49] reduce inference latency by enabling pipelined and parallel execution across reasoning steps. Autellix [23] introduces a queue-aware scheduling policy that minimizes wait time based on latency statistics. AI Metropolis [60] reduces coordination overhead in multi-agent systems via out-of-order execution, while Murakkab [7] improves resource isolation by managing agent workloads at the cloud scheduling layer.

While these works focus on optimizing specific components such as scheduling or execution flow, our study provides a broader characterization of system-level behaviors across diverse agent architectures, helping to understand the resource-efficiency trade-offs of AI agents at scale.

VIII. CONCLUSION

This paper provides the first system-level characterization of AI agents from an AI infrastructure perspective. While these LLM-based agents demonstrate powerful reasoning capabilities through dynamic, multi-step interactions, they also introduce substantial compute, memory, and energy overheads that are orders of magnitude higher than conventional single-turn LLM inference. Our analysis shows that common agent design patterns, such as iterative reflection or parallel reasoning, incur heavy latency penalties and infrastructure costs, especially when deployed at scale. Moreover, test-time scaling yields sharply diminishing returns in accuracy, challenging the cost-effectiveness of current agent implementations.

These findings underscore an urgent need to rethink agent architecture and workflow design. Rather than relying on brute-force test-time scaling, future agents should adopt compute-aware reasoning strategies that optimize accuracy

per unit cost. This includes smarter scheduling, caching, prompt engineering, and hybrid scaling approaches that adapt to deployment constraints. By exposing the hidden costs of agentic reasoning and offering actionable insights into their infrastructure impact, we hope this work informs future system and algorithm co-design for scalable and sustainable AI agents.

REFERENCES

- [1] A. Agrawal, N. Kedia, A. Panwar, J. Mohan, N. Kwatra, B. Gulavani, A. Tumanov, and R. Ramjee, “Taming Throughput-Latency tradeoff in LLM inference with Sarathi-Serve,” in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024.
- [2] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebron, and S. Sanghai, “GQA: Training generalized multi-query transformer models from multi-head checkpoints,” in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.
- [3] Anthropic, “Model Context Protocol (MCP),” 2024. [Online]. Available: <https://docs.anthropic.com/en/docs/agents-and-tools/mcp/>
- [4] P. A. Bao Tran, “Semiconductor Manufacturing Energy Consumption: How Green Is the Chip Industry?” 2025. [Online]. Available: <https://patentpc.com/blog/semiconductor-manufacturing-energy-consumption-how-green-is-the-chip-industry-latest-stats>
- [5] M. Besta, N. Blach, A. Kubicek, R. Gerstenberger, M. Podstawski, L. Gianinazzi, J. Gajda, T. Lehmann, H. Niewiadomski, P. Nyczzyk, and T. Hoefler, “Graph of thoughts: solving elaborate problems with large language models,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.
- [6] T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, and T. Dao, “Medusa: Simple llm inference acceleration framework with multiple decoding heads,” in *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.
- [7] G. I. Chaudhry, E. Choukse, Iñigo Goiri, R. Fonseca, A. Belay, and R. Bianchini, “Towards Resource-Efficient Compound AI Systems,” in *arxiv.org*, 2025.
- [8] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, “Evaluating Large Language Models Trained on Code,” in *arxiv.org*, 2021.
- [9] R. Coulom, “Efficient selectivity and backup operators in monte-carlo tree search,” in *International conference on computers and games*. Springer, 2006.
- [10] A. de Vries, “The growing energy footprint of artificial intelligence,” *Joule*, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2542435123003653>
- [11] Dell’Oro Group, “Data Center Capex to Surpass \$1 Trillion by 2029, According to Dell’Oro Group,” 2025. [Online]. Available: <https://www.delloro.com/news/data-center-capex-to-surpass-1-trillion-by-2029/>
- [12] Demandsage, “How Many Google Searches Per Day [2025 Data],” 2025. [Online]. Available: <https://www.demandsage.com/google-search-statistics/>
- [13] Google, “Announcing the Agent2Agent Protocol (A2A),” 2025. [Online]. Available: <https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interopability/>
- [14] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt, “Measuring Mathematical Problem Solving With the MATH Dataset,” in *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, 2021.
- [15] IBM, “What is a hyperscale data center?” 2024. [Online]. Available: <https://www.ibm.com/think/topics/hyperscale-data-center>
- [16] Industry Week, “The Success of US Chip Manufacturing Hinges on Our Electric Grid,” 2024. [Online]. Available: <https://www.industryweek.com/technology-and-iiot/energy/article/21284413/the-success-of-us-chip-manufacturing-hinges-on-our-electric-grid>

- [17] S. Kim, S. Moon, R. Tabrizi, N. Lee, M. Mahoney, K. Keutzer, and A. Gholami, "LLMCompiler: An LLM Compiler for Parallel Function Calling," 2023. [Online]. Available: <https://github.com/SqueezeAILab/LLMCompiler>
- [18] S. Kim, S. Moon, R. Tabrizi, N. Lee, M. W. Mahoney, K. Keutzer, and A. Gholami, "An LLM Compiler for Parallel Function Calling," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.
- [19] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient Memory Management for Large Language Model Serving with PagedAttention," in *Proceedings of the ACM Symposium on Operating System Principles (SOSP)*, 2023.
- [20] J. Landry, "Landry Announces Meta Selects North Louisiana as Site of \$10 Billion Artificial Intelligence Optimized Data Center," 2024. [Online]. Available: <https://gov.louisiana.gov/news/4697>
- [21] Y. Leviathan, M. Kalman, and Y. Matias, "Fast inference from transformers via speculative decoding," in *Proceedings of the 40th International Conference on Machine Learning*, 2023.
- [22] G. Li, H. A. A. K. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem, "CAMEL: Communicative agents for "mind" exploration of large language model society," in *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, 2023.
- [23] M. Luo, X. Shi, C. Cai, T. Zhang, J. Wong, Y. Wang, C. Wang, Y. Huang, Z. Chen, J. E. Gonzalez, and I. Stoica, "Autellix: An efficient serving engine for llm agents as general programs," 2025.
- [24] Meta, "Llama-3.1-70B-Instruct," 2025. [Online]. Available: <https://huggingface.co/meta-llama/Llama-3.1-70B-Instruct>
- [25] Meta, "Llama-3.1-8B-Instruct," 2025. [Online]. Available: <https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct>
- [26] MLPerf, "MLPerf Inference: Datacenter." [Online]. Available: <https://mlcommons.org/benchmarks/inference-datacenter/>
- [27] NVIDIA, "NVIDIA DCGM Documentation." [Online]. Available: <https://docs.nvidia.com/datacenter/dcgm/latest/index.html>
- [28] NVIDIA, "NVIDIA Dynamo Platform." [Online]. Available: <https://developer.nvidia.com/dynamo>
- [29] OpenAI, "Introducing ChatGPT," 2022. [Online]. Available: <https://openai.com/index/chatgpt/>
- [30] OpenAI, "OpenAI Function Calling," 2023. [Online]. Available: <https://platform.openai.com/docs/guides/function-calling?api-mode=chat>
- [31] OpenAI, "New funding to scale the benefits of AI," 2024. [Online]. Available: <https://openai.com/index/scale-the-benefits-of-ai/>
- [32] OpenAI, "Announcing The Stargate Project," 2025. [Online]. Available: <https://openai.com/index/announcing-the-stargate-project/>
- [33] OpenAI, "Deep Research FAQ," 2025. [Online]. Available: <https://help.openai.com/en/articles/10500283>
- [34] OpenAI, "Introducing Deep Research," 2025. [Online]. Available: <https://openai.com/index/introducing-deep-research/>
- [35] OpenAI, "New funding to build towards AGI," 2025. [Online]. Available: <https://openai.com/index/march-funding-updates/>
- [36] OpenAI Newsroom, "300M weekly active ChatGPT users," 2024. [Online]. Available: <https://x.com/OpenAINewsroom/status/1864373399218475440>
- [37] P. Patel, E. Choukse, C. Zhang, A. Shah, Í. Goiri, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative llm inference using phase splitting," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2024.
- [38] O. Press, M. Zhang, S. Min, L. Schmidt, N. A. Smith, and M. Lewis, "Measuring and narrowing the compositionality gap in language models," in *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.
- [39] Reuters, "ChatGPT sets record for fastest-growing user base - analyst note," 2023. [Online]. Available: <https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/>
- [40] Reuters, "OpenAI says ChatGPT's weekly users have grown to 200 million," 2024. [Online]. Available: <https://www.reuters.com/technology/artificial-intelligence/openai-says-chatgpts-weekly-users-have-grown-200-million-2024-08-29/>
- [41] Reuters, "OpenAI's weekly active users surpass 400 million," 2025. [Online]. Available: <https://www.reuters.com/technology/artificial-intelligence/openais-weekly-active-users-surpass-400-million-2025-02-20/>
- [42] K. Santhanam, D. Raghavan, M. S. Rahman, T. Venkatesh, N. Kunjal, P. Thaker, P. Levis, and M. Zaharia, "ALTO: An Efficient Network Orchestrator for Compound AI Systems," in *Proceedings of the 4th Workshop on Machine Learning and Systems*, 2024.
- [43] Seattle City Light, "Fingertip Facts," 2024. [Online]. Available: <https://www.seattle.gov/documents/Departments/CityLight/FingertipFacts.pdf>
- [44] SemiAnalysis, "The Inference Cost Of Search Disruption - Large Language Model Cost Analysis," 2023. [Online]. Available: <https://semianalysis.com/2023/02/09/the-inference-cost-of-search-disruption/>
- [45] ShareGPT Team, "Sharegpt," 2023. [Online]. Available: <https://sharegpt.com>
- [46] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language Agents with Verbal Reinforcement Learning," 2023. [Online]. Available: <https://github.com/noahshinn/reflexion>
- [47] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language Agents with Verbal Reinforcement Learning," in *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, 2023.
- [48] C. Snell, J. Lee, K. Xu, and A. Kumar, "Scaling llm test-time compute optimally can be more effective than scaling model parameters," in *arxiv.org*, 2024.
- [49] X. Tan, Y. Jiang, Y. Yang, and H. Xu, "Teola: Towards End-to-End Optimization of LLM-Based Applications," in *arxiv.org*, 2025.
- [50] The Washington Post, "A bottle of water per email: the hidden environmental costs of using AI chatbots," 2024. [Online]. Available: <https://www.washingtonpost.com/technology/2024/09/18/energy-ai-use-electricity-water-data-centers/>
- [51] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," in *arxiv.org*, 2023.
- [52] U.S. Energy Information Administration (EIA), "Electricity explained - Electricity generation, capacity, and sales in the United States," 2024. [Online]. Available: <https://www.eia.gov/energyexplained/electricity/electricity-in-the-us-generation-capacity-and-sales.php>
- [53] vLLM, "vLLM Documentation." [Online]. Available: <https://docs.vllm.ai/en/stable/>
- [54] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-Consistency Improves Chain of Thought Reasoning in Language Models," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.
- [55] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS '22. Red Hook, NY, USA: Curran Associates Inc., 2022.
- [56] Wikipedia contributors, "Wikipedia api," 2025. [Online]. Available: https://www.mediawiki.org/wiki/API:Main_page
- [57] Wolfram Alpha LLC, "Wolfram alpha api," 2025. [Online]. Available: <https://products.wolframalpha.com/api/>
- [58] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang, "Autogen: Enabling next-gen LLM applications via multi-agent conversations," in *First Conference on Language Modeling (CoLM)*, 2024.
- [59] xAI Colossus, "Colossus — xAI," 2025. [Online]. Available: <https://x.ai/colossus>
- [60] Z. Xie, H. Kang, Y. Sheng, T. Krishna, K. Fatahalian, and C. Kozyrakis, "AI Metropolis: Scaling Large Language Model-Based Multi-Agent Simulation with Out-of-Order Execution," in *arxiv.org*, 2024.
- [61] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. W. Cohen, R. Salakhutdinov, and C. D. Manning, "HotpotQA: A Dataset for Diverse, Explainable Multi-Gop Question Answering," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.
- [62] S. Yao, H. Chen, J. Yang, and K. Narasimhan, "WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents,"

- in *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, 2022.
- [63] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan, “Tree of Thoughts: Deliberate Problem Solving with Large Language Models,” in *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, 2023.
 - [64] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, “React: Synergizing Reasoning and Acting in Language Models,” in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.
 - [65] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, “ReAct: Synergizing Reasoning and Acting in Language Models,” 2023. [Online]. Available: <https://github.com/ysmyth/ReAct>
 - [66] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, “Orca: A Distributed Serving System for Transformer-Based Generative Models,” in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2022.
 - [67] Y. Zhong, S. Liu, J. Chen, J. Hu, Y. Zhu, X. Liu, X. Jin, and H. Zhang, “DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving,” in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024.
 - [68] A. Zhou, K. Yan, M. Shlapentokh-Rothman, H. Wang, and Y.-X. Wang, “Language Agent Tree Search Unifies Reasoning, Acting, and Planning in Language Models,” in *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.
 - [69] A. Zhou, K. Yan, M. Shlapentokh-Rothman, H. Wang, and Y.-X. Wang, “Official Repo of Language Agent Tree Search (LATS),” 2024. [Online]. Available: <https://github.com/lapisrocks/LanguageAgentTreeSearch>
 - [70] D. Zhou, N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. Le, and E. Chi, “Least-to-Most Prompting Enables Complex Reasoning in Large Language Models,” in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.