Workload Characterization and Modeling of Agentic AI Systems in Realistic Applications

Anonymous ACL submission

Abstract

Agentic AI marks a shift from passive language models to autonomous systems capable of reasoning, tool use, and iterative perceive–think–act interactions. Despite rapid progress, their workload characteristics remain insufficiently understood. Our contributions are three-fold. First, we present a systematic taxonomy of Agentic AI architectures, with a focus on the ReAct paradigm and its applications in deep research, code generation, and GUI control. Second, we introduce a unified performance modeling framework that decomposes workloads into computation, memory, and communication, capturing multi-turn interaction dynamics. Third, we conduct a fine-grained empirical study across representative benchmarks (BrowseComp-ZH, SWE-Bench-Lite, and OSWorld), identifying distinct workload patterns and performance bottlenecks that challenge existing LLM serving optimizations.

1 Introduction

Artificial intelligence is evolving from passive language models to autonomous agentic systems. Traditional large language models (LLMs) act as prompt-driven predictors without initiative or goal-directed behavior Vaswani et al. (2017); Brown et al. (2020), whereas Agentic AI enables systems to interpret context, make decisions, and execute actions toward objectives Wu et al. (2024). This shift transforms LLMs into active agents capable of iterative reasoning and environment interaction, supporting complex multi-step workflows Yao et al. (2023b); Klang et al. (2025).

However, existing LLM serving systems (e.g., vLLM Kwon (2025), SGLang Zheng et al. (2024)) and benchmarks (e.g., HELM Liang et al. (2023), OpenLLM Myrzakhan et al. (2024)) are designed for *stateless*, *single-turn* workloads, optimizing independent request latency and throughput. This assumption breaks down for Agentic AI, whose workflows exhibit four key properties: (1) *stateful multi-turn interactions* requiring persistent KV cache Zhang et al. (2025); (2) *computation gaps* due to asynchronous tool execution Shi et al. (2025); (3) *non-linear execution graphs* with fork-join dependencies Bian et al. (2025); Yao et al. (2023a); and (4) *program-level dependencies* where global task completion defines optimization objectives Pagonas et al. (2025).

Recent agent-aware systems (e.g., Tempo Zhang et al. (2025), Tokencake Bian et al. (2025), Autellix Shi et al. (2025), Justitia Yang et al. (2025)) introduce agent-centric scheduling and memory management, but lack a unified understanding of workload characteristics. However, these prior works focus on algorithmic frameworks Hong et al. (2024); Qian et al. (2024); Park et al. (2023) or isolated optimizations Shen et al. (2025); Fang et al. (2025), without systematically characterizing resource bottlenecks or providing principled performance models Gao et al. (2025); Luo et al. (2025); Liao et al. (2025); Ma et al. (2024). Furthermore, no existing work offers unified theoretical modeling, leaving system designers without principled guidance for hardware provisioning and cross-scenario resource allocation in next-generation agent infrastructures.

In order to bridge this gap, we develop a unified theoretical framework that extends conventional LLM inference modeling to capture the dynamics of Agentic AI workflows. Our approach decomposes end-to-end latency into several interleaved components (section 3). Through rigorous analysis across three representative scenarios, we quantify distinctive resource utilization patterns (section 4). These insights bridge algorithmic design and system optimization for next-generation agent infrastructures. To our knowledge, this is the first work to provide comprehensive theoretical modeling and empirical analysis of production-grade Agentic AI systems across diverse real-world scenarios.

Our key contributions are:

1. **Unified Theoretical Framework**: A latency modeling framework decomposing Agentic AI workloads into computation, memory, and communication components, enabling systematic analysis of agent interactions.

2. **Scenario-Specific Analysis**: In-depth modeling of three representative scenarios (code generation, deep research, and GUI automation) to reveal distinct resource patterns.

3. **Architectural Implications**: Identification of core challenges under the ReAct paradigm and heterogeneous resource requirements across the agent lifecycle.

2 Background

2.1 Characteristic and Capabilities of Agentic AI

As shown in Figure 1, the Agentic AI operates via an iterative "Perceive–Think–Act" (ReAct) cycle, fundamentally differing from stateless LLM serving through multi-round interactions, tool invocations, and external agent environments.

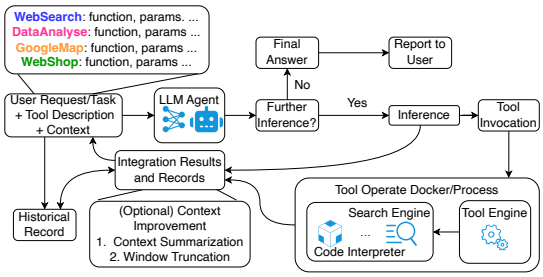


Figure 1: Agentic AI based on ReAct paradigm Yao et al. (2023b).

Multi-round Interaction Agentic AI maintains state across interactions through context management and memory mechanisms. Within each ReAct cycle, the agent: (1) *Perceives* environmental inputs; (2) *Reconstructs state* by integrating historical context with specific techniques; (3) *Reasons* using current perceptions and task history; (4) *Acts* via external tools or APIs; (5) *Iterates* as outcomes feed back into subsequent cycles. This stateful paradigm decomposes complex goals into sub-tasks, introducing distinctive computational patterns such as intermittent execution and asynchronous tool interactions Wei et al. (2025).

Tool Invocation Agent systems achieve embodiment through tool invocation, enabling the "Act" phase of the ReAct cycle. Agents leverage external tools to perform concrete operations and alter environmental states. For example, Web Sailor Li et al. (2025a) employs two tools: (1) `Search`: Queries Google Search and returns top-k results; (2) `Visit`: Fetches webpage content via Jina Reader and extracts task-relevant information.

Agent Environment The environment synthesizes historical context, current observations, and available actions into structured prompts for the LLM, enabling agents to effect tangible changes in the external world. The iterative ReAct cycle allows agents to refine strategies through continuous adaptation. By externalizing memory management and workflow orchestration, this architecture preserves the LLM's inference capabilities while conferring persistent identity and environment-aware execution Klang et al. (2025); Dery et al. (2026).

2.2 Algorithmic Advancements in Agentic AI

Recent work establishes a unified single-agent architecture comprising four interdependent modules: identity profiling, memory management, planning strategies, and action execution. Together, these modules form a recursive optimization cycle for systematic agent construction.

Identity Profiling This module defines operational boundaries through static expert-defined roles (e.g., AutoGen's dialog roles Wu et al. (2024), MetaGPT/ChatDev's SOP-based professionals Hong et al. (2024); Qian et al. (2024)) or dynamic parameterized initialization for social simulation (e.g., Generative Agents' personality-injected residents Park et al. (2023), RecAgent's user-data-driven profiles Wang et al. (2023b)).

Memory Management Memory spans short-term context windows for immediate execution (ReAct Yao et al. (2023b)) and long-term storage via skill libraries (Voyager's executable skills Wang et al. (2024a)), experience repositories (Reflexion's trial logs Shinn et al. (2023)), or hierarchical swapping systems (MemGPT's OS-inspired design Packer et al. (2023)). External knowledge integration evolves from basic RAG Lewis et al. (2020) to collaborative segmented processing (Chain of Agents Zhang et al. (2024)) and interleaved retrieval-reasoning (IRCoT Trivedi et al. (2023)).

Planning Strategies Task decomposition advances from single-path reasoning (Plan-and-Solve Wang et al. (2023a)) to multi-path explo-

ration with backtracking (Tree of Thoughts Yao et al. (2023a)). Feedback-driven refinement enables self-improvement through iterative correction without additional training data (Self-Refine Madaan et al. (2023)).

Action Execution This module converts textual outputs into environmental actions through tool utilization, enabling models to learn when and how to invoke these tools (Toolformer Schick et al. (2023)). Physical interaction employs the "code-as-action" paradigm, where agents control embodied entities through generated and executed programs (Voyager Wang et al. (2024a)).

2.3 System-level Optimization in Agentic Workflows

The transition from stateless LLM serving to stateful Agentic AI introduces structural system challenges requiring agent-aware optimizations.

Agent-aware Scheduling and Orchestration Modern schedulers treat agents as structured programs. Autellix Shi et al. (2025) employs OS-inspired process tables and multi-level feedback queues to mitigate blocking. Tempo Zhang et al. (2025) maximizes service gain density by backfilling tool-wait intervals. Justitia Yang et al. (2025) ensures worst-case latency guarantees via virtual time scheduling for multi-tenant fairness. Cortex Pagonas et al. (2025) enables stage-aware resource pooling while ALTO Santhanam et al. (2024) optimizes cross-component dataflow orchestration.

Memory-centric Optimization Extended contexts and branching reasoning intensify KV Cache pressure. Tokencake Bian et al. (2025) decouples logical and physical memory using predictive prefetching guided by agent step graphs. Halo Shen et al. (2025) merges common subgraphs in batch queries for offline workloads. Simulation-based planners derive globally optimal execution plans for multi-LLM applications Fang et al. (2025).

Infrastructure for Agent Training Specialized training systems address Agentic RL challenges. Agent Lightning Luo et al. (2025) decouples agent execution from RL training to support arbitrary architectures. AWorld Lu et al. (2025) accelerates data generation through large-scale parallel simulation. Token-level masking resolves credit assignment in multi-turn dialogs Wang et al. (2025a) and SLA-aware strategies dynamically adjust batching parameters for code generation tasks Thangarajah et al. (2025).

2.4 Real-world Applications

The profound impact of Agentic AI extends beyond theoreti-cal interest to deliver substantial practical benefits across numer-ous domains. By automating complex, multi-step workflows that require real-time contextual understanding and decision-making, these systems provide exceptional economic and technical value. An application overview is listed in Table 3. Collectively, these demonstrate Agentic AI's capacity to transform AI into a collaborative partner, enabling higher levels of automation for addressing increasingly complex challenges.

3 Modeling and Analysis of Representative Agentic AI Application Scenarios

Table 1: Frequently used notations

Name	Descriptions
τ	Parameter precision (=2 when BF16 is adopted)
N_L	# of layers
B	Batch size
t	Index of inference round
L_t	Input token length in round t
N_t	Output token length in round t
d_m	Hidden dimension size
h_q	# of query head
h_{kv}	# of key or value head
d_h	Head dimension size
d_{ffn}	FFN size
TP, PP, SP	Degree of parallelism in TP, PP, and SP paradigm
$v_{comp}, v_{comm}, v_{mem}$	FLOPs of GPU, bandwidth of internal network, bandwidth of HBM
$\xi_{comp}, \xi_{comm}, \xi_{mem}$	Utilization rate of computation, communication, and memory resources

As Agentic AI expands across domains, it increasingly enables the construction of dynamic and executable models that simulate real-world systems. This section presents detailed case studies focused on performance modeling and system workload analysis. Notably, since code generation, deep research, and GUI agents share a common LLM inference workflow, their unified performance model is presented in section 3.1, with scenario-specific distinctions elaborated in subsequent subsections.

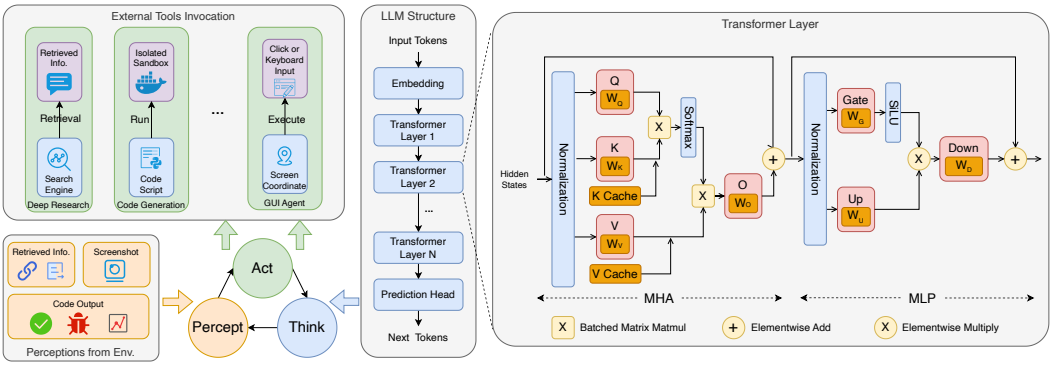


Figure 2: Overview of LLM-based Agent focusing on ReAct cyclic process.

3.1 Performance Modeling for ReAct Paradigm

While transformer-based LLM inference is well characterized Yuan et al. (2024), it captures only a subset of Agentic AI execution. Under the ReAct paradigm Yao et al. (2023b), agent execution interleaves LLM inference (*Think*), tool invocation (*Act*), and feedback integration (*Perceive*), introducing additional latency and intermittent execution beyond conventional serving. We therefore model agent performance as a composition of these phases. Focusing on the *Think* phase, and assuming modern architectures with GQA and SwiGLU, we decompose its latency into three components: computation, communication, and memory access. The dominant cost arises from matrix multiplications in attention and MLP layers, with prefill and decode complexities given in Equations 1 and 2. All notations are summarized in Table 1.

$$T^{\text{comp}}_{\text{prefill},t} = \frac{\tau N_L B}{v_{\text{comp}}\xi_{\text{comp}}}\Big(2L_t d_m^2\Big(1+\frac{h_{kv}}{h_q}\Big) + 2L_t^2 d_h h_q + 3L_t d_m d_{ffn}\Big) \tag{1}$$

$$T^{\text{comp}}_{\text{decode},t} = \frac{\tau N_L N_t}{v_{\text{comp}}\xi_{\text{comp}}}\Big(2d_m^2\Big(1+\frac{h_{kv}}{h_q}\Big) + (N_t - 1 + 2L_t)d_h h_q + 3d_m d_{ffn}\Big) \tag{2}$$

Memory bottlenecks differ between phases. During prefill, the primary cost stems from loading static model weights and initializing the KV cache, as formalized in Equation 3. In contrast, the decode phase primarily involves repeated KV cache accesses to compute attention scores, as quantified in Equation 4.

$$T^{\text{mem}}_{\text{prefill},t} = \frac{\tau N_L}{v_{\text{mem}}\xi_{\text{mem}}}\Big(2d_m^2 + 2d_m h_{kv} d_h + 3d_m d_{ffn} + 2L_t h_{kv} d_h\Big) \tag{3}$$

$$T^{\text{mem}}_{\text{decode},t} = \tau N_L \sum_{l=L_t}^{L_t+N_t-1} \frac{2d_h h_{kv}(l+1)}{v_{\text{mem}}\xi_{\text{mem}}} \tag{4}$$

Additionally, we incorporate classical model parallelism strategies, including tensor, pipeline, and sequence parallelism. As these paradigms exhibit overlapping patterns, we model their costs using an interleaved formulation (Equation 5), where δ and σ denote the effective overlap ratios among different communication components.

$$T_{\text{comm},t} = \frac{\tau}{v_{\text{net}}\xi_{\text{net}}}\Big(\delta \cdot PP(L_t+N_t)d_m + \sigma \cdot N_L(L_t+N_t)(h_q d_h + d_m) + (1-\delta-\sigma)\cdot\frac{2N_L L_t d_m}{SP}\Big) \tag{5}$$

Furthermore, the agent frequently invokes external tools to interact with the environment, resulting in probabilistic overhead due to the risks associated with tool misuse (Equation 6).

$$T_{\text{tool},t} = \sum_{T_i \in \mathcal{T}} t_{\alpha_t}(T_i) \times p_{\theta_C}(\alpha_t | T_i, x_{t-1}, H_{t-1}, q) \times p_{\theta_C}(T_i | x_{t-1}, H_{t-1}, q)] \tag{6}$$

Finally, the per-turn latency $T_{\text{infer},t}$ aggregates computation, memory access, internal communication, and tool invocation costs (Equation 7). For a complete task requiring R rounds, the total execution time accumulates across all rounds (Equation 8). Notably, R is not predetermined but emerges dynamically from the agent's planning efficacy and task complexity, introducing substantial variance in end-to-end performance.

$$T_{\text{infer},t} = T_{\text{compute},t} + T_{\text{memory},t} + T_{\text{comm},t} + T_{\text{tool},t} \tag{7}$$

$$T_{\text{infer}} = \sum_{t=1}^{R} T_{\text{infer},t} \tag{8}$$

3.2 Code Generation and Software Engineering

Workflow Analysis Code generation agents translate natural language specifications into executable programs via an environment-grounded ReAct loop, interleaving reasoning and sandboxed execution Wang et al. (2024c); Yang et al. (2024); Wang et al. (2024b). In a regression training task, the agent iteratively performs implementation, error diagnosis, and refinement, until producing a complete solution. This workflow exhibits three key characteristics: (1) sandbox initialization overhead before the first iteration; (2) tight coupling between code generation and environmental feedback, forming dependency chains; and (3) variable iteration depth driven by debugging complexity.

System Workload Analysis Code generation agents introduce a cold-start phase absent in standard inference: sandbox initialization (e.g., image pulling and container setup) incurs a one-time latency $T_{\text{cold_start}}$. Subsequent iterations follow the latency model in Equation 7, where tool latency $T_{\text{tool},t}$ is dominated by containerized execution rather than network calls. Two key characteristics arise: (1) *Rapid context growth*, as accumulated code and outputs require compression (e.g., summaries H_t) to mitigate quadratic attention costs; and (2) *Coupled memory footprint*, where each agent maintains both KV cache and sandbox state, yielding $M_{\text{total}} = N_{\text{agents}}(M_{\text{KV cache}} + M_{\text{sandbox}})$ and complicating multi-tenant resource allocation.

3.3 GUI Agents

Workflow Analysis GUI agents map natural language instructions to pixel-level actions for autonomous computer interaction. They process screenshots and generate coordinate-based actions via a multimodal ReAct loop: visual perception (ViT-based encoding), interface reasoning (grounding language to UI elements), action planning (e.g., `click`, `type`), and state verification via subsequent screenshots. OpenCUA Wang et al. (2025b), built on Qwen2.5-VL and evaluated on OSWorld Xie et al. (2024), exemplifies this paradigm. For example, to create a "Favorites" folder, the agent navigates menus, executes the creation action, and verifies the result through visual feedback. This workflow presents three challenges: (1) spatial reasoning over dynamic UI states; (2) precise coordinate mapping under resolution variance; and (3) visual feedback-driven error recovery.

System Workload Analysis GUI agent latency per round extends Equation 7 with an additional client-side processing term $T_{\text{client},t}$, presenting four key bottlenecks: (1) *Visual encoding*: for resolution $H \times W$, token count scales as $L_{\text{img}} = \frac{HW}{P_{\text{patch}}^2 K_{\text{pool}}}$, incurring quadratic prefill cost; (2) *Client-side processing*: transmitting K historical frames (e.g., $\sim$10 MB at 1920$\times$1080) makes $T_{\text{client},t}$ bandwidth-bound; (3) *KV cache growth*: visual tokens accumulate as $O(RL_{\text{img}})$, often dominating context and memory bandwidth; and (4) *Environment latency*: OS delays (100–500 ms) exceed LLM inference time, leading to GPU idle periods.

3.4 Advanced Information Retrieval and Analysis

Workflow Analysis Deep research agents follow a ReAct loop, interleaving reasoning (*Think*), tool invocation (*Act*), and feedback processing (*Perceive*) until completion. A representative system is WebSailor Huang et al. (2025); Li et al. (2025a), which integrates search and content extraction tools for autonomous web-based research. For complex retrieval tasks, the agent iteratively decomposes queries, issues and refines searches, and synthesizes evidence through multi-source verification.

System Workload Analysis Deep research workloads alternate between compute-intensive inference (*Think*) and latency-dominated tool execution (*Act*). Bottlenecks arise in three dimensions: (1) *Memory access*: prefill is limited by model weight loading, while decode becomes bandwidth-bound due to KV cache growth under effective bandwidth $v_{\text{mem}}\xi_{\text{mem}}$; (2) *Internal communication*: distributed inference incurs overhead from tensor, pipeline, and sequence parallelism, with latency scaling in d_m and L_t and inversely with $v_{\text{net}}\xi_{\text{net}}$; and (3) *External tool invocation*: tool calls introduce network-bound latency $T_{\text{tool},t}$ that dominates on-device computation, leading to execution gaps and GPU underutilization.

Table 4 in section A summarizes differences across code generation, deep research, and GUI agents, which share a common *Think* phase but diverge in perception and action patterns.

4 Experiments

In this section, we present the experimental setup, results, and key insights across three scenarios (code generation, GUI agents, and deep research). Due to space constraints, we report only the key insights for the deep research setting.

4.1 Code Generation

4.1.1 Experimental Setup

Validation Architecture & Metrics We employ the OpenHands framework Wang et al. (2024c) with CodeAct agents Wang et al. (2024b), evaluating the following metrics: (1) task success rate, (2) end-to-end latency, (3) fine-grained latency breakdown (computation, memory, communication, tool execution), and (4) token consumption and context growth dynamics.

Models & Datasets & Environment We evaluate four open-source models that span architectural and specialization diversity: (1) Qwen3-14B, a general-purpose large language model (LLM) with strong reasoning capabilities; (2) Qwen3-8B, a lightweight variant designed for efficiency comparison; (3) Qwen2.5-Coder-7B, a code-specialized model; and (4) Llama-3.1-8B-Instruct, an instruction-tuned model from the Llama family. For tasks, we sample 10 from SWE-Bench-Lite Jimenez et al. (2023), covering functional implementation, version compatibility fixes, and bug corrections to reflect realistic developer workflows.

All experiments run on homogeneous servers with 2× NVIDIA RTX A6000 GPUs interconnected via NVLink. Models are deployed via vLLM 0.4.3 with tensor parallelism (TP=2) and 0.8 memory utilization. The software stack comprises Ubuntu 22.04 LTS, CUDA 12.3, and PyTorch 2.8.0.

4.1.2 Experimental Results

Table 2: Task completion rate of different models in testset.

Model	Pass	Fail	Illegal Answer
Qwen3-14B	2	7	1
Qwen3-8B	2	4	4
Qwen2.5-Coder-7B	0	0	10
Llama3.1-8B-Instruct	0	2	8

Task Completion and Interaction Dynamics As shown in Table 2, Qwen3-14B achieves the highest pass rate (20%) with two fully correct implementations, while Qwen3-8B matches this count but generates more non-standard responses. The code-specialized Qwen2.5-Coder-7B produces par-

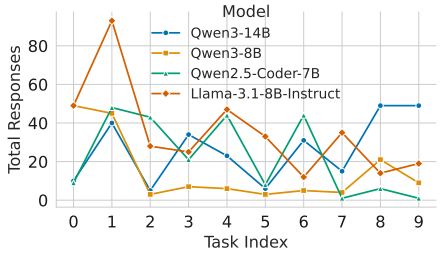
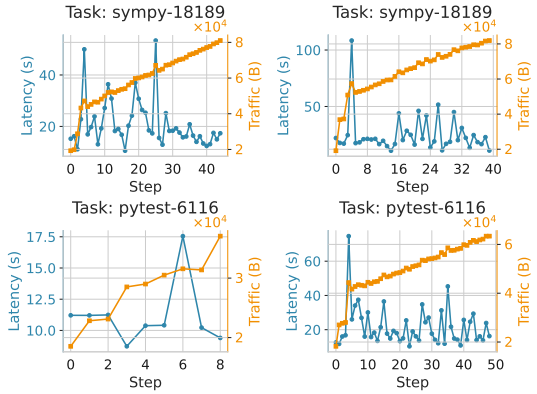
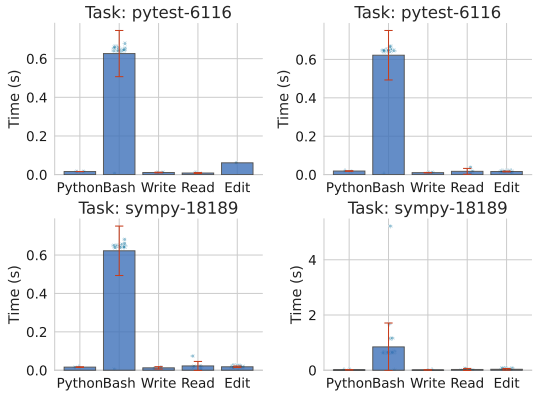


Figure 3: Total number of inference rounds when applying different models on the testset.



(a) Qwen3-8B **(b)** Qwen3-14B

Figure 4: Evolution of traffic & end-to-end latency in code generation tasks.



(a) Qwen3-8B **(b)** Qwen3-14B

Figure 5: Distribution of tool overhead in code generation tasks.

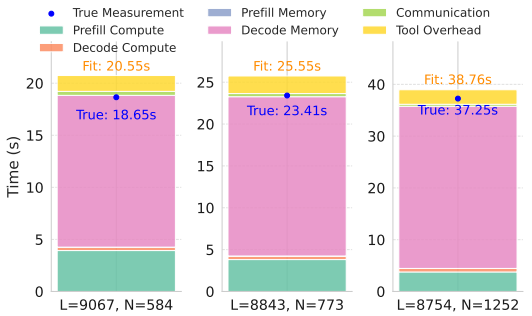


Figure 6: Fitting vs. true measurement in time breakdown (prompt length L, generate length N).

tially correct snippets yet fails to adhere to OpenHands conventions (e.g., tool invocation syntax), and Llama-3.1-8B-Instruct yields two standard-format responses containing logical errors.

Figure 3 reveals a quality–efficiency tradeoff: Qwen3-14B requires substantially more interaction rounds than Qwen3-8B for equivalent tasks (e.g., 38 vs. 14 rounds for task 6), investing additional computation in exhaustive reasoning and error diagnosis. This refinement strategy benefits complex bug fixes but proves inefficient for simpler tasks where shallow reasoning suffices.

Traffic Evolution Patterns Figure 4 characterizes distinct context growth signatures. Qwen3-14B exhibits rapid initial expansion during codebase explorations followed by sustained linear growth from ReAct cycle accumulation. In contrast, Qwen3-8B shows multi-phase traffic spikes aligned with discrete debugging stages (e.g., syntax error correction → logic error identification → edge case handling), indicating reliance on explicit phase transitions to advance problem-solving.

Tool Execution Profiling As quantified in Figure 5, Bash interpreter operations dominate tool execution time (>95%), primarily involving Linux commands (`cat`, `grep`) and package installation (`apt install`). Python execution and file I/O contribute negligible overhead. Critically, even dominant Bash operations consume <5% of the total ReAct cycle latency, confirming that LLM inference and context management, rather than tool execution, constitute the primary bottleneck.

Accuracy of Latency Modeling We evaluate the accuracy of our end-to-end latency model by comparing predicted and measured execution times across three tasks, as shown in Figure 6. The model achieves an average deviation of less than 10%. Remaining discrepancies are primarily due to dynamic system effects such as operator overlap and hardware-level optimizations, which are difficult to capture precisely in analytical modeling. Due to space constraints, we omit detailed results for the deep research and GUI scenarios; however, they exhibit consistent trends with the presented results.

Key Insight. Code generation workloads exhibit a fundamental asymmetry: tool execution governs semantic complexity but contributes minimally to computational latency. The primary bottleneck stems from context inflation during iterative debugging, where each refinement cycle appends complete code snippets and execution outputs, accelerating KV cache growth beyond linear rates. This motivates context compression techniques that preserve debugging history while mitigating $O(L_t^2)$ attention costs.

4.2 GUI Agent

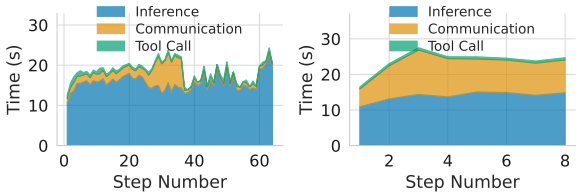
4.2.1 Experimental Setup

Validation Architecture & Metrics We evaluate OpenCUA Wang et al. (2025b) deployed on OSWorld Xie et al. (2024), measuring: (1) end-to-end latency decomposition (client processing, network, VLM inference, environment execution), (2) visual token consumption versus action complexity, (3) bandwidth utilization under varying context window sizes, and (4) action-type-specific computational signatures (`click`, `scroll`, `type`).

Models & Datasets & Environment We evaluate OpenCUA-7B Wang et al. (2025b), a Qwen2.5-VL variant fine-tuned for computer use, deployed on OSWorld Xie et al. (2024). The model processes 1920×1080 screenshots via ViT encoding (256 visual tokens per image), fused with textual instructions in the LLM backbone. We sample 10 tasks spanning four interface paradigms: office automation (LibreOffice), web interaction (Chrome), system management (file operations, VLC), and professional tools (GIMP, VS Code).

A client-server architecture isolates perception/execution (client) from reasoning (server). The server employs an NVIDIA L40 vGPU (24GB VRAM) running vLLM 0.4.2 with OpenAI-compatible API (Ubuntu 22.04, CUDA 12.8); the client executes Dockerized Ubuntu Desktop containers capturing screenshots via X11. Network latency averages 15ms under load.

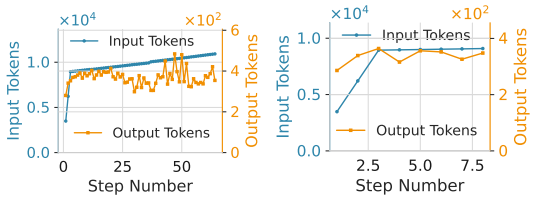
4.2.2 Experimental Results



(a) Task: chrome **(b)** Task: multi_apps

Figure 7: End-to-end latency breakdown of two tasks in GUI scenarios.

End-to-End Latency Breakdown Figure 7 reveals a deployment-dependent bottleneck shift: VLM inference dominates latency in single-application tasks (88% of total), while multi-



(a) Task: chrome **(b)** Task: multi_apps

Figure 8: Comparison between the length of output and input tokens in GUI tasks.

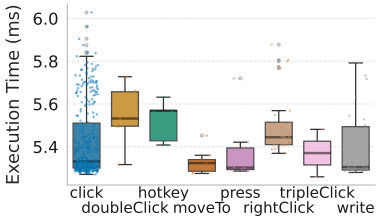


Figure 9: Distribution of tool execution overhead in GUI tasks.

application workflows incur substantial communication overhead (43% on average) from transmitting multiple high-resolution screenshots per cycle. Tool execution latency remains highly variable (0.3–2.1s) due to application-specific factors like page loads and UI rendering, hindering batched inference optimization.

Token Consumption & Tool-Use Distribution GUI agents exhibit extreme input-output asymmetry (Figure 8), with a 100:1 token ratio confirming a "fat client" bottleneck where latency is constrained by client-side image encoding and uplink bandwidth rather than server-side computation. Output tokens fluctuate significantly relative to steadily growing inputs, reflecting variable tool selection and context accumulation. Despite mouse manipulation primitives (`click`, `moveTo`, `rightClick`) comprising 90% of tool executions (Figure 9), they contribute <5% to total ReAct cycle latency, which is substantially less than VLM inference and image transmission.

Key Insight GUI agents encounter a fundamental bandwidth-computation asymmetry: the continuous transmission of high-resolution screenshots generates input token volumes that are orders of magnitude larger than the outputs. This "fat client" bottleneck, where uplink bandwidth dominates end-to-end latency, requires architectural co-design such as predictive caching of UI states to enable scalable real-world deployment.

4.3 Deep Research

Detailed experimental setups and results are illustrated in section A.1

Key Insight The dominant bottleneck shifts with deployment modality: cloud APIs suffer from inference/compute overhead, while local models are constrained by external tool latency and context expansion. Optimizing agent frameworks thus requires the co-design of lightweight tool interfaces and context compression mechanisms, especially for long-horizon tasks.

5 Limitations

Despite providing a unified performance model for ReAct-based agents, our analysis has several limitations.

Simplified Multi-Turn Modeling. Our model treats each ReAct cycle as independent, overlooking cross-turn dependencies such as context accumulation, adaptive reasoning strategies, and optimization effects across iterations. This simplification may underestimate latency in long-horizon tasks where context growth and memory access patterns evolve dynamically.

Single-Agent Assumption. We focus on single-agent execution, whereas production systems commonly involve multiple heterogeneous agents collaborating on shared tasks. Such settings introduce additional complexities, including inter-agent communication, resource contention (e.g., GPU memory), and scheduling dependencies, which are not captured in our current formulation.

Limited System-Level Validation. Our evaluation is primarily based on analytical modeling and controlled settings. Large-scale production environments with heterogeneous hardware, varying service-level objectives, and dynamic workloads may exhibit behaviors not fully reflected in our abstractions.

6 Conclusion

This paper presents a theoretical framework and workload analysis of Agentic AI systems, highlighting a shift from passive text generation to active, stateful reasoning-action cycles. This paradigm introduces distinct computational characteristics that require system designs beyond conventional LLM inference optimization.

Key challenges remain in improving resource efficiency for long-horizon tasks, ensuring reliability in complex environments, and addressing ethical risks of autonomous decision-making.

References

Zhuohang Bian, Feiyang Wu, Teng Ma, and Youwei Zhuo. 2025. Tokencake: A KV-cache-centric serving framework for LLM-based multi-agent applications. *arXiv preprint arXiv:2510.18586*.

Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D. White, and Philippe Schwaller. 2024. Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, 6(5):525–535.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Lucio M Dery, Zohar Yahav, Henry Prior, Qixuan Feng, Jiajun Shen, and Arthur Szlam. 2026. Latent space communication via kv cache alignment. *arXiv preprint arXiv:2601.06123*.

Jingzhi Fang, Yanyan Shen, Yue Wang, and Lei Chen. 2025. Improving the end-to-end efficiency of offline inference for multi-LLM applications based on sampling and simulation. *arXiv preprint arXiv:2503.16893*.

Wei Gao, Yuheng Zhao, Tianyuan Wu, Sheng Xiong, Wei Wang, Di An, Lunxi Cao, Diliyar Muhtar, Zhong Liu, Jiarui Zhang, Wenshuo Zhang, and Chuan Wu. 2025. Rollart: Scaling agentic rl training via disaggregated infrastructure. *arXiv preprint arXiv:2512.22560*.

Alireza Ghafarollahi and Markus J. Buehler. 2025. Automating alloy design and discovery with physics-aware multimodal multiagent AI. *Proceedings of the National Academy of Sciences (PNAS)*, 122(4):e2414074122.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, and 1 others. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations (ICLR)*.

Yuxuan Huang, Yihang Chen, Haozheng Zhang, Kang Li, Huichi Zhou, Meng Fang, Linyi Yang, Xiaoguang Li, Lifeng Shang, Songcen Xu, Jianye Hao, Kun Shao, and Jun Wang. 2025. Deep research agents: A systematic examination and roadmap. *Preprint*, arXiv:2506.18096.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

Eyal Klang, Mohammad Omar, Ganesh Raut, Reem Agbareia, Prem Timsina, Robert Freeman, Niall Gavin, Lauren Stump, Alexander W. Charney, Benjamin S. Glicksberg, and Girish N. Nadkarni. 2025. Orchestrated multi agents sustain accuracy under clinical-scale workloads compared to a single agent. *medRxiv*, 2025.08.22.25334049. Preprint. Available at: https://doi.org/10.1101/2025.08.22.25334049.

Patrick Tser Jern Kon, Jiachen Liu, Qiuyi Ding, Yiming Qiu, Zhenning Yang, Yibo Huang, Jayanth Srinivasa, Myungjin Lee, Mosharaf Chowdhury, and Ang Chen. 2025. Curie: Toward rigorous and automated scientific experimentation with AI agents. *arXiv preprint arXiv:2502.16069*.

Woosuk Kwon. 2025. *vLLM: An Efficient Inference Engine for Large Language Models*. Ph.D. thesis, UC Berkeley.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, Weizhou Shen, Junkai Zhang, Dingchu Zhang, Xixi Wu, Yong Jiang, Ming Yan, Pengjun Xie, Fei Huang, and Jingren Zhou. 2025a. Websailor: Navigating super-human reasoning for web agent. *Preprint*, arXiv:2507.02592.

Nian Li, Chen Gao, M. Li, Y. Li, and Q. Liao. 2024. EconAgent: Large language model-empowered agents for simulating macroeconomic activities. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025b. Search-o1: Agentic search-enhanced large reasoning models. *Preprint*, arXiv:2501.05366.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Alexander Cosgrove, Christopher D Manning, Christopher Re, Diana Acosta-Navas, Drew Arad Hudson, and 31 others. 2023. Holistic evaluation of language models. *Transactions on Machine Learning Research*. Featured Certification, Expert Certification.

Junwei Liao, Muning Wen, Jun Wang, and Weinan Zhang. 2025. Marft: Multi-agent reinforcement fine-tuning. *Preprint*, arXiv:2504.16129.

Siyuan Lu, Zechuan Wang, Hongxuan Zhang, Qintong Wu, Leilei Gan, Chenyi Zhuang, Jinjie Gu, and Tao Lin. 2025. AWorld: Orchestrating the training recipe for agentic AI. *arXiv preprint arXiv:2508.20404*.

Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Dongsheng Li, Luna K. Qiu, and Yuqing Yang. 2025. Agent lightning: Train ANY AI agents with reinforcement learning. *arXiv preprint arXiv:2508.03680*.

Hao Ma, Tianyi Hu, Zhiqiang Pu, Liu Boyin, Xiaolin Ai, Yanyan Liang, and Min Chen. 2024. Coevolving with the other you: Fine-tuning llm with sequential cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 37:15497–15525.

Aman Madaan, Niket Tandon, Prakhar Gupta, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Md Motaleb Hossen Manik and Ge Wang. 2026. Openclaw agents on moltbook: Risky instruction sharing and norm enforcement in an agent-only social network. *arXiv preprint arXiv:2602.02625*.

Aidar Myrzakhan, Sondos Mahmoud Bsharat, and Zhiqiang Shen. 2024. Open-llm-leaderboard: From multi-choice to open-style questions for llms evaluation, benchmark, and arena. *arXiv preprint arXiv:2406.07545*.

Charles Packer, Sarah Wooders, Kevin Lin, and 1 others. 2023. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*.

Nikos Pagonas, Yeounoh Chung, Kostis Kaffes, and Arvind Krishnamurthy. 2025. Cortex: Workflow-aware resource pooling and scheduling for agentic serving. *arXiv preprint arXiv:2510.14126*.

Joon Sung Park, Joseph C O'Brien, Carrie J Cai, and 1 others. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*.

Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. 2024. Graph retrieval-augmented generation: A survey. *Preprint*, arXiv:2408.08921.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, and 1 others. 2024. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*.

Keshav Santhanam, Deepti Raghavan, Muhammad Shahir Rahman, Thejas Venkatesh, Neha Kunjal, Pratiksha Thaker, Philip Levis, and Matei Zaharia. 2024. ALTO: An efficient network orchestrator for compound AI systems. In *Proceedings of the 4th Workshop on Machine Learning and Systems (EuroMLSys '24)*. ACM.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, and 1 others. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Junyi Shen, Noppanat Wadlom, and Yao Lu. 2025. Batch query processing and optimization for agentic workflows. *arXiv preprint arXiv:2509.02121*.

Xiaoxiang Shi, Colin Cai, Tianjun Zhang, Justin Wong, Yanping Huang, Zhifeng Chen, and Ion Stoica. 2025. Autellix: An efficient serving engine for LLM agents as general programs. *arXiv preprint arXiv:2502.13965*.

Noah Shinn, Federico Cassano, Beck Labash, and 1 others. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Kishanthan Thangarajah, Arthur Leung, Boyuan Chen, and Ahmed E. Hassan. 2025. SLA-awareness for AI-assisted coding. *arXiv preprint arXiv:2503.19876*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, and 1 others. 2024a. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research (TMLR)*.

Lei Wang, Wanyu Xu, Yihuai Lan, and 1 others. 2023a. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*.

Lei Wang, Jingsen Zhang, Xu Chen, Yankai Lin, and 1 others. 2023b. Recagent: A novel simulation paradigm for recommender systems. *arXiv preprint arXiv:2306.02552*.

Renxi Wang, Rifo Ahmad Genadi, Bilal El Bouardi, Yongxin Wang, Fajri Koto, Zhengzhong Liu, Timothy Baldwin, and Haoran Li. 2025a. AgentFly: Extensible and scalable reinforcement learning for LM agents. *arXiv preprint arXiv:2507.14897*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024b. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, and 2 others. 2024c. Openhands: An open platform for AI software developers as generalist agents. *arXiv preprint arXiv:2407.16741*. Formerly known as OpenDevin.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, and 1 others. 2025b. Opencua: Open foundations for computer-use agents. *arXiv preprint arXiv:2508.09123*.

Xinming Wei, Jiahao Zhang, Haoran Li, Jiayu Chen, Rui Qu, Maoliang Li, Xiang Chen, and Guojie Luo. 2025. Agent.xpu: Efficient scheduling of agentic LLM workloads on heterogeneous SoC. *arXiv preprint arXiv:2506.24045*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, and 1 others. 2024. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. In *First Conference on Language Modeling (COLM)*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, and 1 others. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*.

Mingyan Yang, Guanjie Wang, Manqi Luo, and Minyi Guo. 2025. Justitia: Fair and efficient scheduling for LLM applications. *arXiv preprint arXiv:2510.17015*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, and 1 others. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, and 1 others. 2023b. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Zhe Zhou, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, and 1 others. 2024. Llm inference unveiled: Survey and roofline model insights. *arXiv preprint arXiv:2402.16363*.

Wei Zhang, Zhiyu Wu, Yi Mu, and Fan Lai. 2025. Tempo: Application-aware LLM serving with mixed SLO requirements. *arXiv preprint arXiv:2504.20068*.

Yusen Zhang, Ruoxi Sun, Yanfei Chen, and 1 others. 2024. Chain of agents: Large language models collaborating on long-context tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, and 1 others. 2024. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583.

Peilin Zhou, Bruce Leon, Xiang Ying, Can Zhang, Yifan Shao, Qichen Ye, Dading Chong, Zhiling Jin, Chenxuan Xie, Meng Cao, and 1 others. 2025. Browsecomp-zh: Benchmarking web browsing ability of large language models in chinese. *arXiv preprint arXiv:2504.19314*.

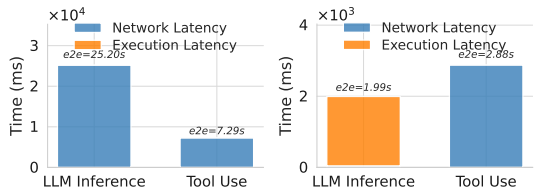
A Appendix

A.1 Deep Research Experiments

A.1.1 Experimental Setup

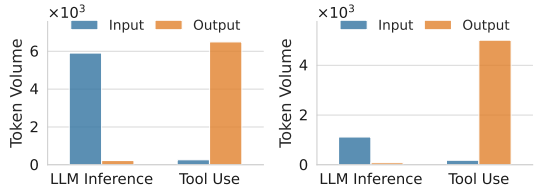
Validation Architecture & Metrics We adopt WebSailor Li et al. (2025a) with a primary focus on evaluating: (1) end-to-end latency, (2) token patterns of LLM inference and tool invocation, (3) distribution of rollout steps, and (4) traffic evolution of LLM inference and tool invocation.

Models & Datasets & Environment We evaluate three models spanning the API-to-local deployment spectrum: (1) GPT-4o-mini (remote API baseline); (2) Qwen2.5-7B deployed via vLLM with tensor parallelism (TP=4) on 4×RTX A6000 GPUs; and (3) Qwen2.5-14B with TP=8 on 8×RTX A6000 GPUs. Experiments use BrowseComp-ZH Zhou et al. (2025), a high-difficulty Chinese benchmark of 289 reverse-engineered multi-hop questions across 11 domains; we sample 50 for controlled evaluation. All local deployments run on homogeneous RTX A6000 GPUs interconnected via NVLink. The software stack comprises Ubuntu 22.04, CUDA 12.1, PyTorch ≥2.0, and vLLM 0.4.2. External APIs (Serper.dev, Jina.ai) maintain low-latency connectivity throughout experiments.



(a) GPT-4o-Mini **(b)** Qwen2.5-14B

Figure 10: End-to-end latency across different models in deep research tasks

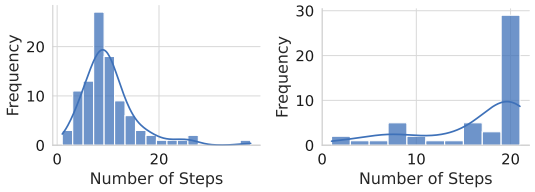


(a) GPT-4o-Mini **(b)** Qwen2.5-14B

Figure 11: Input and output token volumes in LLM inference and tool invocation in deep research tasks

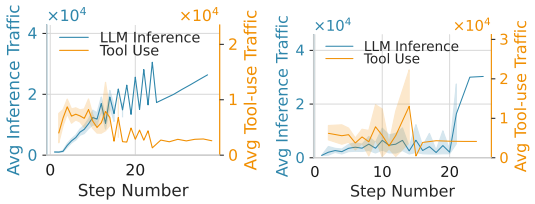
A.1.2 Experimental Results

Latency Characteristic & Token Consumption Figure 10 reveals a fundamental deployment-dependent bottleneck: for remote GPT-4o-mini, network latency (primarily HTTP handshakes) dominates both inference and tool invocation, whereas local Qwen deployments exhibit computation- and memory-bound inference with tool latency remaining network-constrained due to external search services.



(a) GPT-4o-Mini **(b)** Qwen2.5-14B

Figure 12: Distributions of rollout steps under different models in deep research tasks.



(a) GPT-4o-Mini **(b)** Qwen2.5-7B

Figure 13: Traffic evolution of LLM inference and tool invocation in deep research tasks.

This paradigm shift is further reflected in token-level patterns (Figure 11): GPT-4o-mini shows inference-dominated latency driven by API overhead, while self-hosted Qwen agents face tool-invocation latency exceeding inference time, highlighting that local deployments are constrained by external tooling and context management rather than raw compute capacity.

Regarding rollout efficiency, GPT-4o-mini typically converges within 5–15 steps (with a long tail to 38), whereas Qwen2.5-14B clusters near the 20-step maximum (Figure 12b), indicating frequent step-limit exhaustion and suggesting lower reasoning efficiency or premature convergence on complex multi-hop queries.

Network Traffic Patterns & Tool Overhead Dynamics Figure 13 reveals contrasting token consumption patterns across deployment modalities. GPT-4o-mini follows an exploration-exploitation trajectory: tool usage peaks early during information gathering and declines as the agent transitions to internal answer synthesis, while context length grows steadily throughout. In contrast, local Qwen models demonstrate delayed reasoning, with tool invocation increasing during intermediate steps, with LLM token consumption remaining moderate initially and surging sharply near task completion as evidence accumulation precedes concentrated inference.

Table 3: Agentic AI applications across diverse domains.

Application Domain	Core Objective	Flagship Frameworks	Key Contributions and Impact
Scientific Discovery	Automate end-to-end research workflows from hypothesis generation to experimental validation	ChemCrow Bran et al. (2024), AtomAgents Ghafarollahi and Buehler (2025), Curie Kon et al. (2025)	Enable autonomous scientific partnerships across chemistry, biology, and medicine; accelerate discovery cycles through closed-loop experimentation
Software Engineering	End-to-end software development including specification, implementation, testing, and debugging	MetaGPT Hong et al. (2024), ChatDev Qian et al. (2024), SWE-agent Yang et al. (2024)	Generate production-ready software from natural language specifications; transform developer productivity through iterative refinement and tool-augmented execution
Productivity & Automation	Automate complex digital workflows via seamless integration of browsers, APIs, and system tools	AutoGPT Peng et al. (2024), DeepResearch Li et al. (2025b), OpenClaw Manik and Wang (2026)	Enable universal task completion through autonomous tool orchestration; bridge semantic gaps between user intent and digital action execution
Social Science & Economics	Simulate emergent human behaviors and socioeconomic dynamics at scale	Generative Agents Park et al. (2023), EconAgent Li et al. (2024)	Establish agent-based modeling as a rigorous methodology for social science; reveal macro-level phenomena through micro-level interaction patterns
Games & Virtual Worlds	Develop adaptive agents capable of mastering interactive environments and portraying believable characters	Voyager Wang et al. (2024a), ReAct Yao et al. (2023b)	Achieve superhuman performance in open-world games; enable dynamic NPC behavior and assist procedural content generation through embodied reasoning

Table 4: Comparison of resource characteristics across different agent types.

Dimension	Deep Research	Code Generation	GUI Agents
Primary Latency	External API calls (network bandwidth)	Sandbox execution (local compute)	Visual transmission (memory bandwidth)
Context Composition	Query + webpage summaries	Code snippets + terminal outputs	Query + screenshot images
Cold Start Overhead	Negligible	Significant ($T_{\text{cold_start}}$: sandbox init.)	Moderate (T_{client}: client prep.)
Memory Pressure	Single-domain (KV cache)	Dual-domain (KV cache + container memory)	Dual-domain (KV cache + image storage)
Error Recovery	Low (query refinement)	High (sandbox re-initialization)	Moderate (image re-transmission)