What Limits Agentic Systems Efficiency?

Anonymous Author(s)
Affiliation
Address
email

Abstract

Large Language Models (LLMs), such as OpenAI-o1 and DeepSeek-R1, have demonstrated strong reasoning capabilities. To further enhance this process, recent agentic systems, such as *Deep Research*, incorporate web interactions into LLM reasoning to mitigate uncertainties and reduce potential errors. However, existing research predominantly focuses on reasoning performance, often neglecting the efficiency of these systems. In this work, we present a comprehensive empirical study that identifies efficiency bottlenecks in web-interactive agentic systems. We decompose end-to-end latency into two primary components: LLM API latency and web environment latency. Our findings show that both components significantly contribute to the overall system latency. To improve latency, we propose `SpecCache`, a caching framework augmented with speculative execution to reduce web environment overhead. Extensive evaluations on two standard benchmarks demonstrate that our approach reduces web environment overhead by up to $3.51\times$, without compromising agentic system performance.

1 Introduction

Large Language Models (LLMs) have become a cornerstone of modern artificial intelligence, achieving outstanding performance across various downstream tasks. Their strengths in natural language understanding [52], and text generation [13, 64] have enabled significant breakthroughs across disciplines. To further enhance performance, recent advances in large-scale reinforcement learning (RL) have enabled large language models to demonstrate strong long-horizon reasoning abilities. As examples, OpenAI-o1 [37] and DeepSeek-R1 [21] leverage RL methods like PPO [42] and GRPO [43] to strengthen problem-solving capabilities, equipping them for complex reasoning tasks [22, 38].

Although reasoning models can generate step-by-step reasoning chains, their reasoning processes remain constrained by insufficient knowledge [25, 39]. To address this limitation, recent work has proposed agentic systems that combine web interaction with LLM-based reasoning to retrieve external knowledge and access up-to-date information [36]. Existing web-interactive agentic systems can be categorized into the following two types: (1) employing prompt engineering to inject external knowledge into LLMs for complex task completion [31, 56]; (2) leveraging reinforcement learning to integrate search capabilities into LLMs [11, 47, 26]. While existing web-interactive agentic systems primarily focus on improving the reasoning capabilities of LLMs for complex tasks [54, 56], they largely neglect *system efficiency*. The latency of agentic systems is critical for applications with low-latency service-level objectives (SLOs), as it directly affects service reliability and user satisfaction [53, 16].

To address this gap, we systematically benchmark the end-to-end latency of web-interactive agentic systems. We begin by sampling two queries from the *WebWalkerQA* [56] and *Frames* [28] benchmarks to evaluate the latency of a single iteration of the Reflexion-based agentic system [45].

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

As shown in Figure 1, both the LLM API and web environment contribute substantially to the latency of web-interactive agentic systems. Accordingly, we separately analyze the latency introduced by the LLM API and the web environment. To identify key contributors to LLM API latency, we analyze the impact of several factors, including geographic region (e.g., request origin locations), output token length, model type (e.g., reasoning vs. non-reasoning), and API deployment mode (e.g., serverless vs. dedicated) across 15 models from 5 providers: Anthropic, DeepSeek, Google, OpenAI, and Together AI (§2.1). To measure web environment latency, we use the *WebWalkerQA* benchmark [56], which centers on queries related to international organizations, conferences, and educational institutions, making it well-suited for assessing information retrieval performance (§2.2).

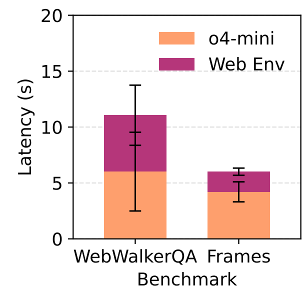


Figure 1: Average latency breakdown per iteration of a Reflexion-based agentic system [45] for sampled question answering.

Our empirical analysis reveals the following important observations: (1) High variability across 15 models and 5 providers is observed in LLM response latency. Latency for fixed-length requests may differ by up to $69.21\times$ based on the time they are issued (§2.1); (2) LLM response latency variance persists across dates and locations (§2.1); (3) Web environment latency can contribute as much as 45.6% to the overall latency of agentic systems (§2.2).

Motivated by the above observations, this paper focuses on reducing web environment latency as the primary strategy to improve the overall latency of agentic systems. This focus is grounded in the expectation that, as LLM deployment infrastructures rapidly advance, scheduling overheads and other system-level inefficiencies will continue to diminish, leading to lower LLM latency without compromising, and potentially even enhancing, model performance [2, 17, 35]. To reduce the web environment latency, we propose `SpecCache` (§3), a novel caching framework that uses speculative execution [30] to mitigate latency in web environments. Specifically, `SpecCache` implements a caching mechanism that stores LLM-generated actions along with a speculative execution path, which uses a draft model to predict the LLM's next action and proactively populate the action cache. Using a draft model unlocks a new dimension that allows environment interaction costs to be concealed by *overlapping* them with model reasoning. Furthermore, `SpecCache` is designed upon the ReAct [60] abstraction; therefore, `SpecCache` can be applied to not only web-interactive agentic systems but also other turn-based agentic systems that interact with external environments.

In summary, our key contributions are as follows:

- We present a comprehensive end-to-end latency analysis of web-interactive agentic systems, decomposing latency into LLM API and web environment components. Our findings show that both contribute significantly to overall latency.
- To reduce web environment overhead, we propose `SpecCache`, a caching framework that stores a small set of LLM-generated actions and corresponding results. In addition, `SpecCache` employs a model-driven strategy to enable the overlap of environment interaction costs with model reasoning.
- We conduct extensive experiments to demonstrate the effectiveness and efficiency of `SpecCache`. Compared with existing agentic systems on *WebWalkerQA* and *Frames*, `SpecCache` achieves up to a $3.51\times$ reduction in web environment overhead while maintaining performance. Our method does not change the results produced by the agentic system, as the caching framework operates on a separate path and does not interfere with the backbone LLM or the agentic system's reasoning path.

2 Latency Analysis

Given that both the LLM API and the web environment significantly contribute to the latency of web-interactive agentic systems (Figure 1), we analyze their impacts separately to gain deeper insight. We begin with a detailed examination of LLM API latency in §2.1, followed by an analysis of web environment performance in §2.2.

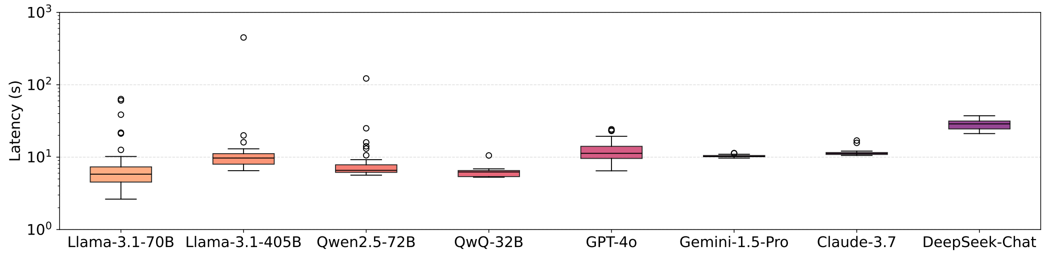


Figure 2: In this figure, we evaluate the end-to-end latency of API calls offered by five AI companies by querying the LLMs every hour. The evaluated models include: (i) Together AI: Llama-3.1-70B, Llama-3.1-405B, Qwen2.5-72B, QwQ-32B; (ii) OpenAI: GPT-4o; (iii) Google: Gemini-1.5-Pro; (iv) Anthropic: Claude-3.7-Sonnet. (v) DeepSeek: DeepSeek-Chat. This figure shows that LLM API response times exhibit high variance, including occasional outliers.

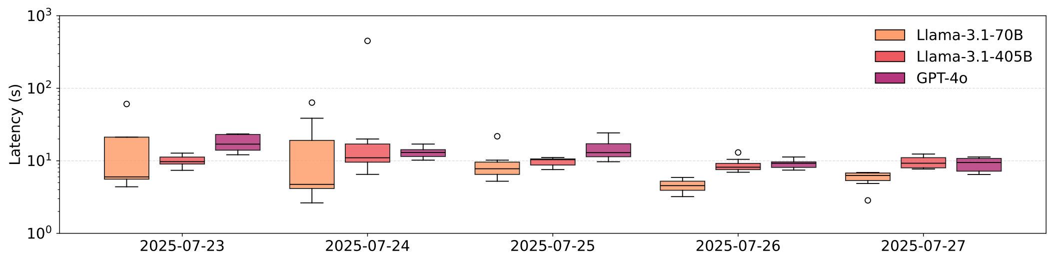


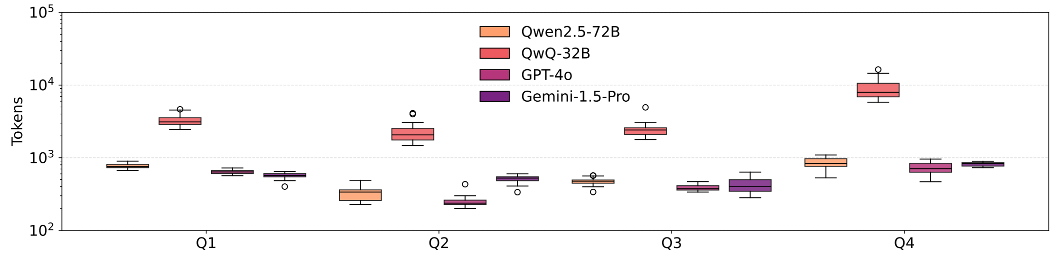
Figure 3: In this figure, we evaluate the end-to-end latency of API calls across different dates. Due to space constraints, we report results from three representative models: Llama-3.1-70B and Llama-3.1-405B provided by Together AI, and GPT-4o from OpenAI. This figure illustrates latency variance over time for various models, with fixed input prompts and a uniform output length.

2.1 LLM API

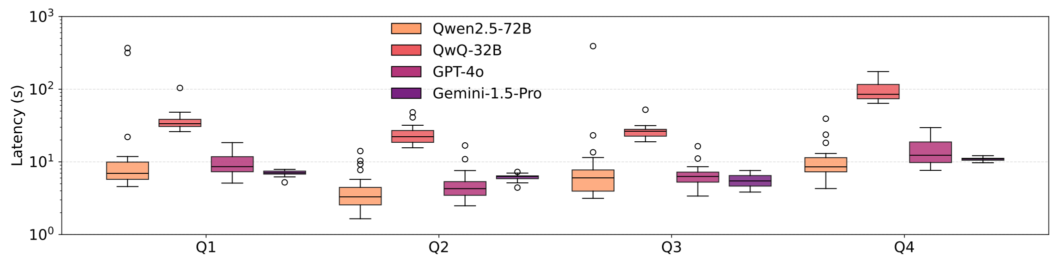
Although many popular models, such as LLaMA [50], Qwen [58], and DeepSeek [33], are open-weight, most agentic systems access LLMs through APIs in practice for two primary reasons. First, leading models, such as OpenAI's GPT-4o [24], Anthropic's Claude 3.5 [3], and Google's Gemini 2.5 [19], remain closed-source and are accessible only via proprietary APIs. Second, the substantial cost and technical complexity of deploying and operating LLMs at scale pose a major barrier for agentic system users [4, 27]. Therefore, in this section, we monitor API call latency over one week to evaluate its impact on agentic system performance.

Setup. Our experiments evaluate LLM API calls from the following providers and their respective models: (i) Anthropic [2]: Claude-3.7-Sonnet, (ii) DeepSeek [17]: DeepSeek-Chat, (iii) Google [20]: Gemini-1.5-Pro, (iv) OpenAI [35]: GPT-4o, and (v) Together AI [1]: Llama-3.1-70B, Llama-3.1-405B, Qwen2.5-72B, and QwQ-32B. Unless otherwise specified, all experiments use identical input questions (listed in Appendix A), generate up to 512 output tokens, and are conducted with top-p = 1 and temperature = 0. All experiments are conducted on a CloudLab [18] instance from Wisconsin. Experiments are conducted between July 23 and July 27, 2025.

High Variability in Latency. We begin with a five-day study evaluating the end-to-end latency of LLM APIs across providers, including Together AI [1], OpenAI [35], Google [20], and Anthropic [2]. Each provider is called once per hour for five consecutive days to collect measurement data. Figure 2 illustrates the considerable variance in API latency. For example, the response time for Llama-3.1-405B provided by Together AI [1] ranges from 6.50 seconds to 449.89 seconds. As shown in Figure 3, LLM API call latency exhibits variance over all five days, with fluctuations differing from day to day. High variance in LLM API call latency may arise from constrained GPU resources on the provider side, leading to queuing delays [44], or from performance noise in the cloud infrastructure hosting the model [15, 46]. Due to the variability in LLM API call latency, larger models can



(a) This figure shows the number of tokens generated for answering Q1–Q4 in Appendix B across four models: Qwen2.5-72B, QwQ-32B, GPT-4o, and Gemini-1.5-Pro. It highlights that QwQ-32B produces significantly more output tokens than the others when solving the sampled math problems.



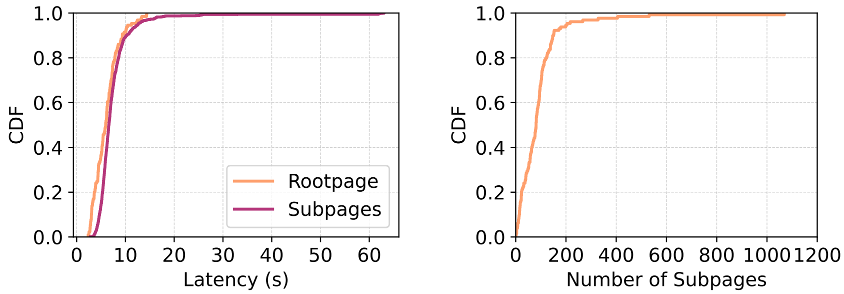
(b) This figure reports the latency for answering Q1–Q4 in Appendix B across four models: Qwen2.5-72B, QwQ-32B, GPT-4o, and Gemini-1.5-Pro. This indicates that QwQ-32B exhibits the highest end-to-end latency among the evaluated models.

Figure 4: This figure shows the LLM API latency and the number of generated tokens for answering Q1–Q4 from Appendix B. Following prior work [7], we set the temperature to 0.6 and top-p to 0.95 when solving the math problems. The results show that output token length significantly affects LLM API end-to-end latency.

occasionally exhibit lower latency than smaller ones. For example, on July 24, 2025, Llama-3.1-405B had lower latency than Llama-3.1-70B. While Gemini-1.5-Pro maintains low variability (3.71%, coefficient of variation), the pronounced variability in Llama-3.1-70B and GPT-4o (135.21% and 36.81%, respectively) poses challenges for consistent performance in latency-sensitive tasks, including language agentic systems [60, 45] and code generation [12] which rely on LLM APIs.

Output Tokens Affect Latency. Unlike the previous section which used fixed input and output token limits, we next randomly sample four questions from the MATH dataset [22] (listed in Appendix B) and allow the LLMs to generate unrestricted-length responses. As shown in Figure 4, although QwQ-32B is faster than Qwen2.5-72B per output token (Figure 2), it produces more output tokens due to the reasoning process, leading to higher overall latency. Notably, both QwQ-32B and Gemini-1.5-Pro are reasoning-oriented models. While Gemini-1.5-Pro is slower than QwQ-32B with fixed input and output tokens, it demonstrates greater efficiency on sampled math questions by generating fewer output tokens per answer. Therefore, learning to generate correct answers using fewer tokens is an important consideration for model training.

Due to limited space, we present more experimental results from varying model types, request locations, and number of output tokens in Appendix E. The following insights are derived from our extensive experiments: (1) With the fixed input question and output tokens, end-to-end API latency can vary by up to $69.21\times$, resulting in an unstable user experience; (2) Moreover, we observe variability in LLM API latency across different dates and three geographic regions. Specifically, the coefficient of variation in latency for Llama-3.1-70B API calls is 135.21% in Wisconsin, 42.61% in South Carolina, and 106.40% in Utah. (3) We observe that the number of output tokens significantly affects LLM API latency. Designing large language models that can solve tasks correctly with fewer output tokens presents a promising research direction.



(a) Latency Distribution (b) Subpages Distribution

Figure 5: The above two CDF figures illustrate the performance characteristics of the web environment from the *WebWalkerQA* benchmark [56]. (a) The distribution of latency when fetching root URLs and subpages, highlighting the initial overhead. (b) The distribution of the number of clickable subpages available from a root URL, showing a large action space.

2.2 Web Environment

In this section, we conduct a detailed analysis of the performance characteristics of external tool APIs and web crawlers. Our analysis reveals that these components introduce substantial overhead, potentially reducing deployment efficiency and diminishing user experience. Moreover, our analysis provides the empirical foundation for the caching and prefetching methodology introduced in §3.

Setup. To understand the performance trade-offs for web-interactive agentic systems operating in a real-world environment, we ground our analysis in a practical case study. While gym-like environments such as WebArena [67] are valuable for reproducibility, they abstract away the noise and performance variability inherent in deploying a live web-interactive language agentic system. Therefore, we utilize the *WebWalkerQA* benchmark [56], which requires agentic systems to perform multi-step reasoning and synthesize answers by exploring multiple pages across various real-world websites. The benchmark's focus on knowledge-intensive domains, such as international organizations, conferences, and educational institutions, makes it an ideal testbed for evaluating information retrieval performance under realistic conditions. We analyze the performance of a Reflexion-based agentic system [45] using QwQ-32B as the backbone reasoning model, following the setup in [56]. Due to resource constraints, we sample 30 tasks from distinct root domains for this case study.

Web Crawl Latency Limits System Performance. An agentic system's interaction cycle in a ReAct [60] or Reflexion-based agentic system [45] is composed of both reasoning (LLM inference) and action (web retrieval). We separately profiled the time spent on reasoning and on actions. Figure 5(a) shows the distribution of latencies for fetching and parsing the HTML of root URLs in our task sample (this includes various conference domains such as sigchi.org, international organization domains such as apec.org, and game producer websites such as rovio.com). As shown in Figure 5(a), the median latency is approximately 6 seconds, with a long tail extending to much higher values, accounting for as much as 45.6% of the total runtime of the agentic system. One potential solution is to use caching techniques [6] to reduce web crawl latency. However, as shown in Figure 5(b), the large and diverse space of subpages presents significant challenges for effective caching. Given these challenges, we introduce `SpecCache` in the next section as a solution for reducing web environment latency.

3 SpecCache

In this section, we will first outline the detailed challenges in designing a caching system aimed at reducing web environment overhead (§3.1). Next, we propose a caching framework that reduces the environment interaction cost by enabling parallelism between model inference and environment interaction, while preserving the original trajectory of the agentic system (§3.2). Finally, we provide a detailed discussion of our caching framework (§3.3).

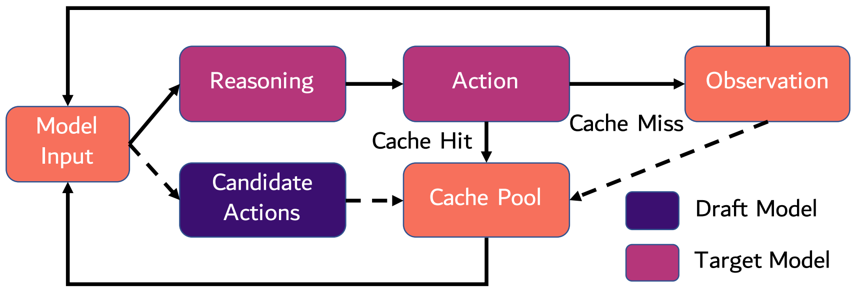


Figure 6: This figure shows the workflow of our `SpecCache` framework. In each iteration, the model input is fed to two independent and non-blocking threads, one Reflexion-based thread and one caching thread aimed at generating candidate actions. The caching thread updates the cache pool with its candidate actions. When the Reflexion-based thread selects an action, it first queries the cache pool. If the cache misses, it executes the action, retrieves the corresponding observation, and proceeds to the next iteration, updating the cache pool with the new action-observation pair.

3.1 Challenges

In standard LLM agentic system deployments, the agentic system waits for an environment response (e.g., a web page load) before invoking the LLM for the next reasoning step, and vice versa. To improve efficiency, our goal is to hide this environment interaction cost by *overlapping it* with model reasoning. A natural approach is to develop a caching mechanism that prefetches environment responses likely to be needed in future steps. However, designing an effective cache for language agentic systems is non-trivial due to the sheer size of the action space. For example, our analysis of the *WebWalkerQA* dataset [56] reveals that each of the 138 root pages contains a median of 81 clickable subpages (Figure 5(b)), representing possible next actions. This high branching factor makes it difficult to accurately anticipate which observations will be needed, presenting a key challenge for prefetching and caching strategies aimed at improving runtime efficiency. Naive strategies, such as uniform action sampling, would result in a near-zero cache hit rate.

3.2 Caching Framework

In this section, we propose a caching and prefetching framework that decouples and overlaps model reasoning and environment interaction, significantly reducing wall-clock latency without compromising task success rates. Our method is a caching system that employs a model-based action-observation cache.

Action-Observation Cache. The active-observation cache, following an LRU policy, is designed to store the outcomes of specific actions taken from a given state (e.g., a webpage). When the target LLM decides on an action, it first queries this cache. A cache hit signifies that the action has been previously executed, and the corresponding observation is immediately retrieved, bypassing the costly interaction with the environment. This on-demand caching of action-observation pairs is crucial for accelerating interactions within a session.

Model-Based Prefetching. To build the action-observation cache, we introduce a model-based prefetching scheme. This component of our framework moves beyond reactive caching to proactively explore and cache potential future states. Leveraging ideas from speculative execution [10, 30], we use a draft model, a smaller LLM running asynchronously with the primary reasoning LLM (the target model). The role of the draft model is to predict the future actions that the target model is likely to take from the current state.

The prefetching process unfolds as follows:

1. **Asynchronous Action Prediction:** While the target LLM performs reasoning, the draft model generates candidate actions (e.g., web crawls), which are executed in parallel.
2. **Asynchronous Caching:** The observations resulting from these speculative actions are stored in the action-observation cache.

When the target LLM eventually determines its next action, it first consults the cache. If the draft model's prediction is accurate, the observation is already present, and the agentic system can proceed instantaneously. This asynchronous prefetching effectively decouples the agentic system's reasoning from the environment's response time (dashed lines in Figure 6), enabling the design of more efficient agentic systems.

3.3 Discussion

Our speculative caching approach introduces new trade-offs that balance latency reduction with increased compute and environment load. The draft model introduces additional computation for running speculative rollouts asynchronously. We complete speculative actions even after the target model selects its next move. This preserves useful data in the cache for future steps. In cases where speculative actions are not used, the main agentic system flow is not interfered with.

The principles underpinning our caching and prefetching framework are not limited to web-interactive agentic systems. This methodology can be generalized to any turn-based agentic system that operates in an environment where the feedback loop constitutes a significant portion of the overall latency. By decoupling reasoning from interaction and proactively exploring the action space, our approach offers a robust and scalable solution for accelerating a wide range of language agentic system applications based on the ReAct-style [60] agentic workflows.

4 Experiments

In this section, we begin by detailing our experimental setup (§4.1). We then evaluate our framework's performance on web-based reasoning and retrieval tasks, analyzing its effectiveness in reducing end-to-end latency in real-world agentic system deployments (§4.2).

4.1 Setup

Agentic Systems. We evaluate a Reflexion-based agentic system using and o4-mini as backbone models. These models have demonstrated state-of-the-art performance on web exploration benchmarks, outperforming both open-sourced methods [48, 31, 32, 55] and proprietary agentic frameworks. For speculative execution, we employ GPT-4.1-mini as draft models.

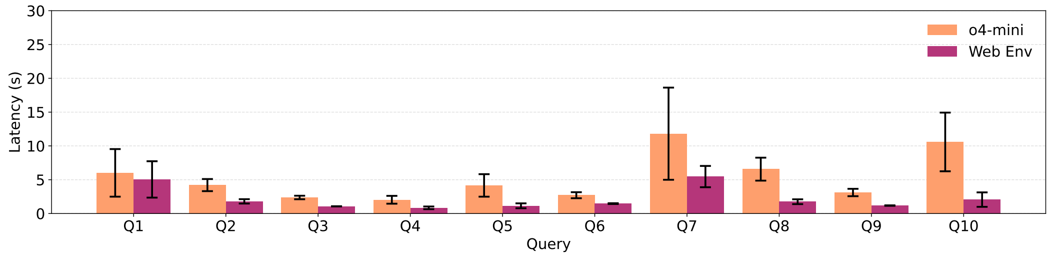
Benchmarks. Reasoning agentic systems capable of multi-hop, in-depth exploration of real-world web content remain a challenging research area, despite recent progress enabled by more powerful models [24, 2, 59]. As highlighted in [56], existing benchmarks such as GAIA [34], MMinA [65], and AssistantBench [62] primarily focus on breadth-wise reasoning, and do not sufficiently evaluate depth-wise web exploration capabilities.

We conduct experiments on two benchmarks designed to capture both multi-hop reasoning and in-depth web exploration: *WebWalkerQA* [56] and *Frames* [28].

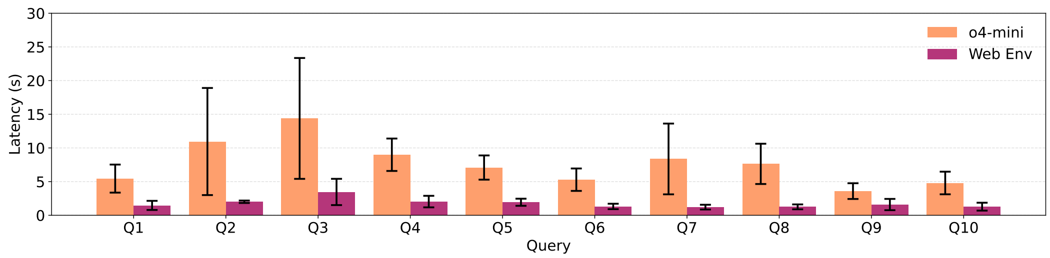
- *WebWalkerQA* evaluates an agentic system's ability to perform multi-hop web reasoning over a large set of websites. We sample a query from each distinct root URL for our evaluations, where the agentic system is provided with the root URL as a starting point.
- *Frames* is a benchmark consisting of factual questions that require synthesizing information from 2 to 15 Wikipedia pages. To emphasize the multi-hop setting, we select a subset of queries that require information from at least 5 distinct sources. The agentic system is provided with only a single Wikipedia page as the seed URL.

We cap the maximum number of iterations per task at 10, where each iteration consists of a reasoning step, an action step, and a critique step. Empirically, most tasks are completed within 5-6 iterations. Given budget constraints, we sample 10 questions from each benchmark to analyze the LLM and web environment overhead in agentic systems, as well as the acceleration achieved through `SpecCache`.

Metric. We measure LLM latency for each Reflexion-based iteration across multiple workloads, averaging results over five runs. Because our caching mechanism operates asynchronously in a separate thread, is non-blocking, and leaves the backbone LLM output unchanged, latency remains



(a) WebWalkerQA



(b) Frames

Figure 7: This figure presents the iteration-wise latency breakdown of the Reflexion-based agentic system when answering sampled questions from *WebWalkerQA* [56] and *Frames* [28]. We perform five runs for each sampled question. The sampled questions are listed in Appendix C.

effectively constant in our experiments, with any variance stemming solely from the model APIs. Averaging mitigates noise and isolates the effects of environmental bottlenecks and the overhead reductions achieved by our approach.

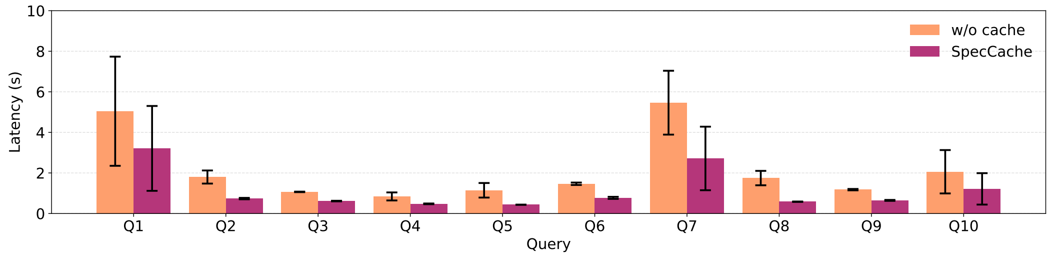
4.2 Experimental Results

Figure 7 presents the iteration-wise latency breakdown of o4-mini and the web environment for each question sampled from *WebWalkerQA* [56] and *Frames* [28]. From Figure 7, we observe high API latency variance for o4-mini, and the web environment constitutes a major component of the Reflexion-based agentic system, consistent with our empirical findings. We then evaluate the acceleration achieved by the proposed `SpecCache` in Figure 8. As shown in Figure 8, `SpecCache` achieves up to a $3.51\times$ reduction in web environment latency for answering sampled questions. These results reveal a new axis for accelerating agentic systems: allocating more compute to asynchronous assistant models allows environment overhead to be overlapped with LLM reasoning.

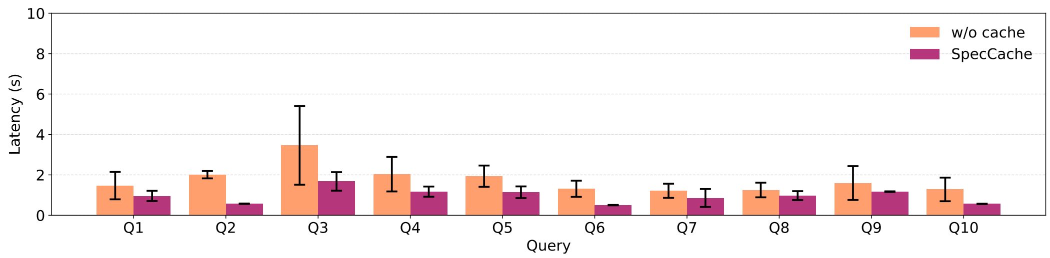
5 Related Work

Large Language Models. The Transformer architecture [51] has been successfully applied to a wide variety of tasks, including text classification [52, 41], text generation [64, 40], mathematical reasoning [14, 22], and code generation [5]. The development of GPT models [8] highlights how scaling up language models substantially improves their performance across a range of downstream tasks. Inspired by the success of GPT, several large-scale language models have been introduced, including LLaMA [50], Gemma [49], Qwen [58], and DeepSeek [33, 21].

Web-Interactive Agentic Systems and Benchmarks. Recent web-interactive agentic systems, including Search-o1 [31], ReSeaerch [11], Search-R1 [26], and WebDancer [55], enhance the reasoning capabilities of large language models by integrating web interaction into their decision-making. Concurrently, benchmarks such as GAIA [34], MMinA [65], AssistantBench [62], BrowseComp [54], and WebWalker [56] have been proposed to evaluate agentic system performance in real-world web environments.



(a) WebWalkerQA



(b) Frames

Figure 8: This figure shows the iteration-wise latency breakdown for the agentic systems accelerated by `SpecCache` when answering sampled questions from *WebWalkerQA* [56] and *Frames* [28]. We use o4-mini as the target model and GPT-4.1-mini as the draft model. We perform five runs for each sampled question. The sampled questions are listed in Appendix C.

LLM Inference. A substantial body of systems research has focused on accelerating LLM inference, leading to notable advances such as Orca [63], PagedAttention [29], RadixAttention [66], and FlashInfer [61]. These approaches target improved LLM inference performance via more efficient hardware usage. Another line of work is speculative decoding [10, 30, 57], which accelerates LLM inference by employing a lightweight draft model to generate candidate outputs that the larger target model later verifies. The concept of speculative decoding builds upon speculative execution [9, 23], an optimization widely employed in processors to perform tasks concurrently with verifying their correctness. In this work, we generalize speculative execution to agentic systems by predicting agentic system actions to reduce web environment overhead.

6 Limitations and Future Work

First, we primarily focus on reducing latency arising from the web environment. We leave the reduction of LLM API latency and its variance to future work. Secondly, to control for the high variance and varying outputs of LLM API calls, we fix the API responses when measuring the acceleration achieved by `SpecCache`. Measuring the end-to-end speedup of the agentic system is left for future work. Thirdly, we do not evaluate other types of agentic systems, such as code-based and tool-based agentic systems, leaving their analysis for future work. Lastly, we believe a deeper dive into the API traffic analysis would be of independent research interest, shedding light on how much request batching, query priority scheduling, and LLM execution contribute to the end-to-end latency and variance, respectively.

7 Conclusion

In this paper, we provide a comprehensive empirical analysis of web-interactive agentic systems. Our findings reveal that both the LLM API and the web environment significantly contribute to agentic system latency. To reduce agentic system latency, we propose `SpecCache`, a caching technique designed to mitigate web environment overhead. Extensive evaluations demonstrate that `SpecCache` reduces web environment latency by up to $3.51\times$.

References

[1] T. AI. Together ai api. `https://docs.together.ai/docs/introduction`, 2025. Accessed 2025.

[2] Anthropic. Anthropic api. `https://docs.anthropic.com/en/home`, 2025. Accessed 2025.

[3] Anthropic. Claude 3.5. `https://www.anthropic.com/news/claude-3-5-sonnet`, 2025. Accessed 2025.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[5] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[6] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[7] B. Brown, J. Juravsky, R. Ehrlich, R. Clark, Q. V. Le, C. Ré, and A. Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

[8] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[9] F. W. Burton. Speculative computation, parallelism, and functional programming. *IEEE Transactions on Computers*, 100(12):1190–1193, 2012.

[10] C. Chen, S. Borgeaud, G. Irving, J.-B. Lespiau, L. Sifre, and J. Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

[11] M. Chen, T. Li, H. Sun, Y. Zhou, C. Zhu, H. Wang, J. Z. Pan, W. Zhang, H. Chen, F. Yang, et al. Learning to reason with search for llms via reinforcement learning. *arXiv preprint arXiv:2503.19470*, 2025.

[12] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[13] C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

[14] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[15] D. De Sensi, T. De Matteis, K. Taranov, S. Di Girolamo, T. Rahn, and T. Hoefler. Noise in the clouds: Influence of network performance variability on application scalability. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(3):1–27, 2022.

[16] J. Dean and L. A. Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[17] DeepSeek. Deepseek api. `https://api-docs.deepseek.com/`, 2025. Accessed 2025.

[18] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, et al. The design and operation of {CloudLab}. In *2019 USENIX annual technical conference (USENIX ATC 19)*, pages 1–14, 2019.

[19] Google. Gemini 2.5. `https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/`, 2025. Accessed 2025.

[20] Google. Gemini api. `https://ai.google.dev/gemini-api/docs`, 2025. Accessed 2025.

[21] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[22] 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. *arXiv preprint arXiv:2103.03874*, 2021.

[23] J. L. Hennessy and D. A. Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[24] A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, A. Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[25] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung. Survey of hallucination in natural language generation. *ACM computing surveys*, 55(12):1–38, 2023.

[26] B. Jin, H. Zeng, Z. Yue, J. Yoon, S. Arik, D. Wang, H. Zamani, and J. Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

[27] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht. Occupy the cloud: Distributed computing for the 99%. In *Proceedings of the 2017 symposium on cloud computing*, pages 445–451, 2017.

[28] S. Krishna, K. Krishna, A. Mohananey, S. Schwarcz, A. Stambler, S. Upadhyay, and M. Faruqui. Fact, fetch, and reason: A unified evaluation of retrieval-augmented generation. *arXiv preprint arXiv:2409.12941*, 2024.

[29] 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 29th symposium on operating systems principles*, pages 611–626, 2023.

[30] Y. Leviathan, M. Kalman, and Y. Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

[31] X. Li, G. Dong, J. Jin, Y. Zhang, Y. Zhou, Y. Zhu, P. Zhang, and Z. Dou. Search-o1: Agentic search-enhanced large reasoning models. *arXiv preprint arXiv:2501.05366*, 2025.

[32] X. Li, J. Jin, G. Dong, H. Qian, Y. Zhu, Y. Wu, J.-R. Wen, and Z. Dou. Webthinker: Empowering large reasoning models with deep research capability. *arXiv preprint arXiv:2504.21776*, 2025.

[33] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[34] G. Mialon, C. Fourrier, T. Wolf, Y. LeCun, and T. Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023.

[35] OpenAI. Openai api. `https://platform.openai.com/docs/api-reference/introduction`, 2025. Accessed 2025.

[36] OpenAI. Openai deep research. `https://openai.com/index/introducing-deep-research/`, 2025. Accessed 2025.

[37] OpenAI. Openai-o1. `https://openai.com/o1/`, 2025. Accessed 2025.

[38] L. Phan, A. Gatti, Z. Han, N. Li, J. Hu, H. Zhang, C. B. C. Zhang, M. Shaaban, J. Ling, S. Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

[39] V. Rawte, A. Sheth, and A. Das. A survey of hallucination in large foundation models. *arXiv preprint arXiv:2309.05922*, 2023.

[40] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

[41] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich. Superglue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4938–4947, 2020.

[42] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[43] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. Li, Y. Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[44] Y. Sheng, S. Cao, D. Li, B. Zhu, Z. Li, D. Zhuo, J. E. Gonzalez, and I. Stoica. Fairness in serving large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 965–988, 2024.

[45] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

[46] P. Sinha, A. Guliani, R. Jain, B. Tran, M. D. Sinclair, and S. Venkataraman. Not all gpus are created equal: characterizing variability in large-scale, accelerator-rich systems. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 01–15. IEEE, 2022.

[47] H. Song, J. Jiang, Y. Min, J. Chen, Z. Chen, W. X. Zhao, L. Fang, and J.-R. Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *arXiv preprint arXiv:2503.05592*, 2025.

[48] S. Sun, H. Song, Y. Wang, R. Ren, J. Jiang, J. Zhang, F. Bai, J. Deng, W. X. Zhao, Z. Liu, et al. Simpledeepsearcher: Deep information seeking via web-powered reasoning trajectory synthesis. *arXiv preprint arXiv:2505.16834*, 2025.

[49] G. Team, T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, S. Pathak, L. Sifre, M. Rivière, M. S. Kale, J. Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[50] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[51] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[52] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[53] A. Wang, S. Venkataraman, S. Alspaugh, R. Katz, and I. Stoica. Cake: enabling high-level slos on shared storage systems. In *Proceedings of the Third ACM Symposium on Cloud Computing*, pages 1–14, 2012.

[54] J. Wei, Z. Sun, S. Papay, S. McKinney, J. Han, I. Fulford, H. W. Chung, A. T. Passos, W. Fedus, and A. Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025.

[55] J. Wu, B. Li, R. Fang, W. Yin, L. Zhang, Z. Tao, D. Zhang, Z. Xi, Y. Jiang, P. Xie, et al. Webdancer: Towards autonomous information seeking agency. *arXiv preprint arXiv:2505.22648*, 2025.

[56] J. Wu, W. Yin, Y. Jiang, Z. Wang, Z. Xi, R. Fang, L. Zhang, Y. He, D. Zhou, P. Xie, et al. Webwalker: Benchmarking llms in web traversal. *arXiv preprint arXiv:2501.07572*, 2025.

[57] M. Yan, S. Agarwal, and S. Venkataraman. Decoding speculative decoding. *arXiv preprint arXiv:2402.01528*, 2024.

[58] A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[59] A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Tang, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[60] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[61] Z. Ye, L. Chen, R. Lai, W. Lin, Y. Zhang, S. Wang, T. Chen, B. Kasikci, V. Grover, A. Krishnamurthy, et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. *arXiv preprint arXiv:2501.01005*, 2025.

[62] O. Yoran, S. J. Amouyal, C. Malaviya, B. Bogin, O. Press, and J. Berant. Assistantbench: Can web agents solve realistic and time-consuming tasks? *arXiv preprint arXiv:2407.15711*, 2024.

[63] 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 *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, 2022.

[64] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

[65] Z. Zhang, S. Tian, L. Chen, and Z. Liu. Mmina: Benchmarking multihop multimodal internet agents. *arXiv preprint arXiv:2404.09992*, 2024.

[66] L. Zheng, L. Yin, Z. Xie, C. L. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

[67] S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar, X. Cheng, T. Ou, Y. Bisk, D. Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

A Latency Test Question.

This section introduces the question used to measure LLM API latency, with a fixed input and a controlled number of output tokens.

QUESTION:
Tell a story about Blackberry. Make the story detailed, with rich descriptions, character development, and dialogue. Aim for a story that would take at least n tokens to tell.

Table 1: The question used to measure LLM API latency.

where n can be set to 64, 128, 256, 512, or 1024, depending on the number of output tokens.

B Examples of Math Questions

This section presents the sampled math questions used to measure LLM API latency.

QUESTION 1: Find the constant term in the expansion of $\left(10x^3 - \frac{1}{2x^2}\right)^5$
QUESTION 2: At what value of y is there a horizontal asymptote for the graph of the equation $y = \frac{4x^3+2x-4}{3x^3-2x^2+5x-1}$?
QUESTION 3: How many zeroes are at the end of $42!$ (42 factorial)? (Reminder: The number $n!$ is the product of the integers from 1 to n. For example, $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$.)
QUESTION 4: Suppose that $ABCD$ is a trapezoid in which $\overline{AD} \| \overline{BC}$. Given $\overline{AC} \perp \overline{CD}$, $\overline{AC}$ bisects angle $\angle BAD$, and $[ABCD] = 42$, then compute $[\triangle ACD]$.

Table 2: The sampled math questions used to measure LLM API latency.

C Sampled Questions

C.1 WebWalkerQA

In this section, we present the sampled WebWalkerQA Questions used to measure LLM and Web API latency in Table 3.

QUESTION 1:
During the 7th China International Import Expo (CIIE) in 2024, when will the National Exhibition and Convention Center (Shanghai) be closed for security measures, and who is permitted to access the venue during this period?

QUESTION 2:
Who were the recipients of the POMS Fellows Award in 2006 and the Wickham Skinner Award for Teaching Innovation in 2018?

QUESTION 3:
For ACL 2024, what is the deadline for students requiring financial assistance to apply for discounted virtual registration, and by what date will they be notified about the selection for D&I subsidies?

QUESTION 4:
What was the specific schedule for the social event held on the evening after the ACL 2023 best paper awards ceremony?

QUESTION 5:
When is the paper submission deadline for the ACL 2025 Industry Track, and what is the venue address for the conference?

QUESTION 6:
Where is the Web Conference 2024 Welcome Reception held and what is the nearest transportation method from the Resorts World Convention Centre?

QUESTION 7:
Which event has a higher total reward pool, the SHIBUYA Y3K event on October 2, 2024, or the upcoming The Smurfs: Gargamel's Castle experience?

QUESTION 8:
What is the official launch date of Junkworld on Apple Arcade, and what new feature was introduced in the January 2024 update?

QUESTION 9:
Find the first IGG recruitment contact email in Asia in alphabetical order.

QUESTION 10:
Who was the chair of the 12th APEC Tourism Ministerial Meeting held in Urubamba on June 9, 2024?

Table 3: The sampled WebWalkerQA Questions used to measure LLM and Web API latency.

C.2 Frames

In this section, we present the sampled Frames Questions used to measure LLM and Web API latency in Table 4.

QUESTION 1: I have an element in mind and would like you to identify the person it was named after. Here's a clue: The element's atomic number is 9 higher than that of an element discovered by the scientist who discovered Zirconium in the same year.
QUESTION 2: As of July 1, 2024, what is the parent company of the current record label of the singer of Edge of Seventeen?
QUESTION 3: According to the 1990 United States census, what was the total population of the cities in Oklahoma that had at least 100,000 residents according to the 2020 United States census?
QUESTION 4: The oldest extant football team in Italy plays in a stadium. The stadium is named after a person. Who was the emperor of China when that person was 5 years old?
QUESTION 5: Of the four main characters on Seinfeld, which actor is the oldest?
QUESTION 6: Which species from the genus mulona are both found in the same country?
QUESTION 7: I am moving to the G40 postcode area - what train stations are nearby, as of 2024?
QUESTION 8: Which player scored more than 15 goals in Eredevisie during the 21-22 season and had previously played for Auxerre?
QUESTION 9: Which MP standing as the leader of a major party in the 2019 United Kingdom General Election was also an MP for Henley?
QUESTION 10: What is the etymology of the name of the province to the east of the province in which Hazrati Sultan District is located?

Table 4: The sampled Frames Questions used to measure LLM and Web API latency.

D Prompts and Trajectory

D.1 Target LLM Prompts

We evaluate `SpecCache` on top of a Reflexion [45] agentic system. Table 5-7 outline the prompts used for each component of Reflexion.

TARGET MODEL ACTION PROMPT:

Digging through the buttons to find quality sources and the right information. You have access to the following tools:
<Tool Description>
Use the following format:
Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [<Tool Names>]
Action Input: the input to the action
Observation: the result of the action
Action: the action to take, should be one of [<Tool Names>]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can be repeated zero or more 20 times)
Notice:
- You must take action at every step. When you take action, you must use the tool with the correct format and output the action input.
- You can not say "I'm sorry, but I cannot assist with this request."!!! You must explore.
- When you have sufficient information to answer the query, provide your final answer in the format: "Final Answer: <your answer>"
- Action Input should be valid JSON.
- IF YOU DO NOT HAVE SUFFICIENT INFORMATION, CONTINUE EXPLORING BY TAKING ACTION.
- YOU MUST TAKE ACTION AT EVERY STEP UNLESS YOU ARE PRODUCING YOUR FINAL ANSWER. WHEN YOU TAKE ACTION, YOU MUST USE THE TOOL WITH THE CORRECT FORMAT AND OUTPUT THE ACTION INPUT. THEREFORE, YOU MUST OUTPUT AN ACTION AND AN ACTION INPUT.
- IF YOU ARE PRODUCING YOUR FINAL ANSWER, YOU MUST OUTPUT THE FINAL ANSWER IN THE FORMAT: "Final Answer: <your answer>"

Begin!
<Query>

Table 5: The prompt for the backbone LLM to take an action.

TARGET MODEL SELF-REFLECTION PROMPT:

You are an information extraction agent. Your task is to analyze the given observation and extract ANY information that could help answer the query, including:
- Direct facts and data
- Reasoning and conclusions made by the model
- Historical information that could be relevant
- Any insights that contribute to solving the query
- Background knowledge that supports the answer

Input:
- Query: "<Query>"
- Observation: "<Current Observation>"

Output (JSON):

{
"usefulness": true,
"information": "<Extracted Useful Information> using string format"
}
Or, if the observation contains NO potentially useful information at all:
{
"usefulness": false
}
Guidelines:
- Be generous in what you consider "useful information"
- Include reasoning, conclusions, and background knowledge
- If the observation contains ANY information that could contribute to solving the query, mark it as useful
- Only mark as false if the observation is completely irrelevant

Only respond with valid JSON.

Table 6: The prompt for the target model to perform self-reflection.

MAIN MODEL EVALUATOR PROMPT:

You are a query answering agent. Your task is to evaluate whether the accumulated useful information is sufficient to answer the current query with HIGH CONFIDENCE. If it is sufficient and you are very confident in the answer, return a JSON object with a "judge" value of true and an "answer" field with the answer. If the information is insufficient or you have doubts, return a JSON object with a "judge" value of false.
Input:
- Query: "<Query>"
- Accumulated Information: "<Accumulated Useful Information>"

Output (JSON):
{
"judge": true,
"answer": "<Generated Answer> using string format"
}
Or, if the information is insufficient or you have doubts:
{
"judge": false
}
Guidelines:
- Only mark as sufficient if you are VERY CONFIDENT in the answer
- If you have any doubts about facts, reasoning, or completeness, mark as insufficient
- Consider whether you need more information to verify your answer
- The answer should be clear, complete, and directly address the query
- When in doubt, prefer to continue exploring rather than give a potentially wrong answer

Only respond with valid JSON.

Table 7: The evaluator prompt for the target model to create internal feedback.

D.2 Draft Model Prompt

In this section, we present the prompt for the draft model to predict action in Table 8.

DRAFT MODEL ACTION PREDICTION PROMPT:

Digging through the buttons to find quality sources and the right information. You have access to the following tools:
<Tool Description>
Use the following format:
Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [<Tool Names>]
Action Input 1: {{"button": "About"}}
Action Input 2: {{"button": "Contact"}}
Action Input 3: {{"button": "Application"}}
Observation: the result of the action
Action: the action to take, should be one of [<Tool Names>]
Action Input 1: {{"button": "News"}}
Action Input 2: {{"button": "Info"}}
Action Input 3: {{"button": "Faculty"}}
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can be repeated zero or more 20 times)
Notice:
- You must take action at every step. When you take action, you must use the tool with the correct format and output 3 action inputs.
- You must always output three Action Input lines (Action Input 1, Action Input 2, Action Input 3) for each Action, unless there are fewer than three distinct valid inputs available.
- If there are fewer than three, output as many as are available.
- When you can not find the information you need, you should visit page of previous page recursively until you find the information you need.
- You can not say "I'm sorry, but I cannot assist with this request."!!! You must explore.
- If you do not have sufficient information, continue exploring.
- Action Input should be a valid JSON.
- Do not recommend navigation buttons such as "About Wikipedia", "Search", "Create account", "Log in", "View source", "Print/export", "Navigation".
- Focus on content-specific buttons that are likely to contain information relevant to your query, such as:
    - Names of people, places, events, or topics
    - Years, dates, or time periods
    - Specific categories or sections
    - Links to related articles or detailed pages
Begin!
<Query>

Table 8: The prompt for the draft model that performs speculative action prediction in SpecCache.

D.3 Trajectory Example

In this section, we show an example of a trajectory. We use {Action} to indicate a cache hit and {Action} to indicate cache exists and the content will not be retrieved.

Query:

When is the paper submission deadline for the ACL 2025 Industry Track, and what is the venue address for the conference?

Root URL:

`https://2025.aclweb.org/`

Draft LLM:

Thought: The "Call for Industry Track" button will likely provide details on submission deadlines for the Industry Track. For the venue address, "Participants Info" or similar buttons may give explicit venue details. Starting by exploring the "Call for Industry Track" for the specific deadline and the other buttons for venue information.

Action: visit_page

Action Input 1: {"button": "Call for Industry Track"}

Action Input 2: {"button": "Participants Info"}

Action Input 3: {"button": "Program"}

Target LLM:

Thought: Question: When is the paper submission deadline for the ACL 2025 Industry Track, and what is the venue address for the conference? Thought: To find the Industry Track deadline, click the {{Call for Industry Track}} button; to find the venue address, click {{Participants Info. }},

Action: visit_page,

Action Input: {"button": "Call for Industry Track"}

Observation: <Content retrieved from `https://2025.aclweb.org/calls/industry_track/`>

Draft LLM:

Thought: I have found the paper submission deadline for the Industry Track (21 March 2025). Next, I need to find the venue address. It is most likely in sections such as "Participants Info", "FAQ", or "Visa". I will explore these buttons to locate the venue address.

Action: visit_page

Action Input 1: {"button": "Participants Info"}

Action Input 2: {"button": "FAQ"}

Action Input 3: {"button": "Visa"}

Target LLM:

Thought: Question: When is the paper submission deadline for the ACL 2025 Industry Track, and what is the venue address for the conference? Thought: We already extracted the paper submission deadline (21 March 2025) from the Industry Track page. Next, we need to find the venue address by navigating to the {{ Participants Info }} section via the conference menu. Action: visit_page Action Input: {b̈utton: T̈oggle menu} Observation: The conference menu has appeared, showing entries including {{Participants Info.}},

Action: visit_page,

Action Input: {"button": "Participants Info"}

Observation: <Content retrieved from `https://2025.aclweb.org/venue/`>

Target LLM:

Final Answer: The paper submission deadline for the ACL 2025 Industry Track is 21 March 2025, and the conference will be held at Austria Center Vienna, Bruno-Kreisky-Platz 1, 1220 Wien, Austria.

E Additional LLM API Latency Measurement Results

In this section, we present further experimental results from our inference latency measurements. We follow the experimental setup mentioned in Section 2.1. For the regional ablation study, we use CloudLab [18] instances located in Madison, Clemson, and Utah. The results are shown in Figure 9. Figure 9 shows that LLM API latency exhibits variance regardless of the city from which requests are issued. Furthermore, we also measure the end-to-end latency of LLM API calls by fixing the input questions and varying the number of output tokens. The results are shown in Figure 10. As shown in Figure 10, LLM API latency tends to increase with the number of output tokens, although some variance is observed. Furthermore, as shown in Figure 11, dedicated mode[1] reduces the variance of API calls compared to serverless mode; however, it is also more expensive due to per-minute billing. Finally, we present additional end-to-end latency measurements of LLM API calls across a broader range of models, as shown in Figure 12. The results in Figure 12 support the conclusion that LLM API calls exhibit high variability.

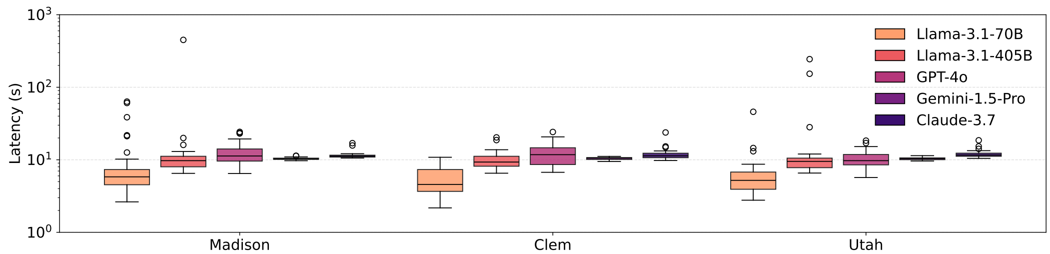


Figure 9: This figure shows the latency variance across regions for models including Llama-3.1-70B, Llama-3.1-405B, GPT-4o, Gemini-1.5-Pro, and Claude-3.7-Sonnet. All requests use the same input and a fixed number of output tokens. Latency is measured by sending requests from machines located in Wisconsin (Madison), South Carolina (Clemson), and Utah.

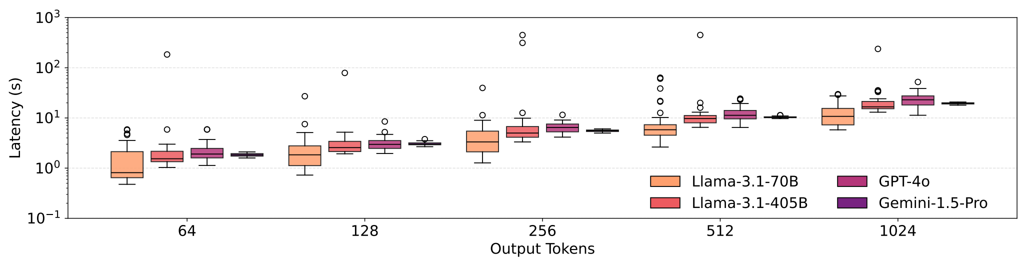
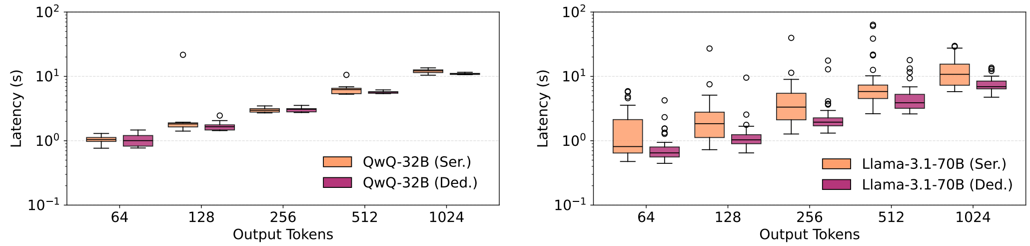


Figure 10: From July 23 to July 27, 2025, we evaluated the end-to-end latency of API calls provided by two AI companies by fixing the input sequence and varying the number of output tokens from 64 to 1024. The evaluated models include: (i) Together AI: Llama-3.1-70B, Llama-3.1-405B; (ii) OpenAI: GPT-4o; (iii) Google: Gemini-1.5-Pro. The figure illustrates an upward trend in LLM API latency with increasing output token count.

[1]With Together AI, we can create on-demand dedicated endpoints with the following advantages: (1) Consistent, predictable performance, unaffected by other users' load in our serverless environment; (2) No rate limits, with a high maximum load capacity; (3) More cost-effective under high utilization; (4) Access to a broader selection of models.



(a) QwQ-32B (b) Llama-3.1-70B

Figure 11: This figure illustrates the relationship between API call latency and the number of output tokens under various API deployment modes. QwQ-32B (Ser.) and Llama-3.1-70B (Ser.) denote API calls made in serverless mode, while QwQ-32B (Ded.) and Llama-3.1-70B (Ded.) refer to calls made in dedicated mode.

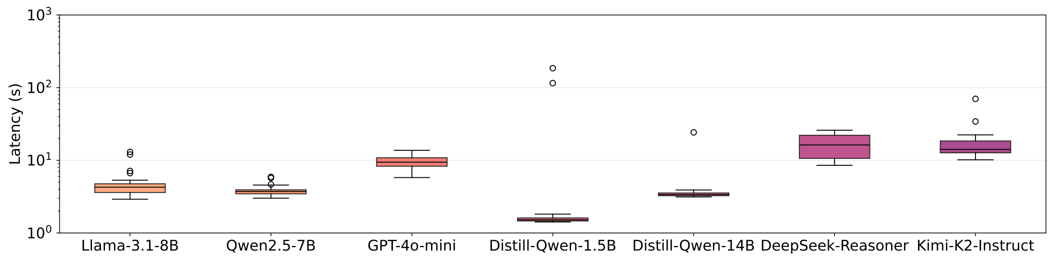


Figure 12: In this figure, we evaluate the end-to-end latency of API calls offered by three AI companies by querying the LLMs every hour. The evaluated models include: (i) Together AI: Llama-3.1-8B, Qwen2.5-7B, DeepSeek-R1-Distill-Qwen-1.5B, DeepSeek-R1-Distill-Qwen-14B, and Kimi-K2-Instruct; (ii) OpenAI: GPT-4o-mini; (iii) DeepSeek: DeepSeek-Reasoner. This figure shows that LLM API response times exhibit high variance, including occasional outliers.

F Model Version

Table 9 lists the versions of the models used in this paper.

Model Name	Version
GPT-4.1	GPT-4.1-2025-04-14
GPT-4.1-mini	GPT-4.1-mini-2025-04-14
o4-mini	o4-mini-2025-04-16
Claude-3.7	Claude-3-7-Sonnet-20250219
DeepSeek-V3	DeepSeek-V3-0324
DeepSeek-R1	DeepSeek-R1-0528

Table 9: This table presents the model versions used in this paper.