Cost-Efficient Serving of LLM Agents via Test-Time Plan Caching

Qizheng Zhang [1] **Michael Wornow** [1] **Kunle Olukotun** [1 2]

Abstract

LLM-based agentic applications have shown increasingly remarkable capabilities in complex workflows but incur substantial costs due to extensive planning and reasoning requirements. Existing LLM caching techniques (like context caching and semantic caching), primarily designed for serving chatbots, are insufficient for agentic applications where outputs depend on external data and environmental contexts. We propose agentic plan caching, a novel approach that extracts, stores, adapts, and reuses structured plan templates from planning stages of agentic applications across semantically similar tasks to reduce the cost of serving. Unlike traditional semantic caching, our system extracts plan templates from completed agent executions at test-time, employs keyword extraction to match new requests against cached plans, and utilizes lightweight models to adapt these templates to task-specific plans with contexts. Evaluation across multiple real-world agentic applications shows that our system can reduce costs by 46.62% on average while maintaining performance, offering a more efficient solution for serving LLM-based agents that complements existing LLM serving infrastructures.

1. Introduction

Agentic applications based on Large Language Models (LLMs) have shown early promise in replicating human performance on a broad range of workflows, from coding (Jain et al., 2024; Jimenez et al., 2023; Yang et al., 2024) to web navigation (Deng et al., 2023; He et al., 2024; Zhou et al., 2023) to open-ended research (gem; ope, b). Many of these LLM-based agents follow a **two-stage pipeline** that alternates between (Yao et al., 2023; Sarukkai et al., 2025): **(1) Plan** – reasoning about what to do next, and **(2) Act** – executing those plans. While effective, these agents incur significant costs due to the complexity of executed workflows (Kwa et al., 2025; Zhang et al., 2024b) and need to interact with external tools and environments (Patil et al., 2024). Specifically, the Plan stage is often implemented via test-time compute techniques (Brown et al., 2024; Snell et al., 2024) like chain-of-thought reasoning (Wei et al., 2022), which can require numerous LLM queries and access to expensive LLMs (*e.g.,* reasoning models). This results in substantial costs for executing agentic workflows via APIs (Chen et al., 2023; Narayan et al., 2025) or locally (Liu et al., 2024c).

To reduce LLM costs, methods have been developed to optimize responses to individual queries (Kwon et al., 2023; Zheng et al., 2024). In particular, *caching* has emerged as a popular approach, with two primary implementations: **Context caching** (*e.g.,* KV cache reuse and prompt caching (Gim et al., 2024; Yao et al., 2025; Yang et al., 2025)) stores internal model states to speed up subsequent generations, while **semantic caching** (Bang, 2023; Schroeder et al., 2025; ama, b) stores and reuses (input, output) pairs to accelerate the serving of queries that are similar to historical queries.

These caching techniques, however, have significant limitations when applied to Plan-Act agents. These agents often require making *data-dependent decisions*, *i.e.,* LLM outputs depend on external data or contextual information that varies between runs. For example, in data analysis applications, the same high-level query (*"summarize key statistics of this dataset"*) will result in similar high-level plans, but different specific details depending on the characteristics of the dataset provided. Similarly, in web or GUI navigation tasks, the same high-level query (*"delete the top comment"*) will require similar sequences of actions (*e.g., "click the menu button, scroll down"*), but the specifics may differ depending on screen size and window position (*e.g., "click coordinates (130, 493), scroll down 38 pixels"*). In such cases, conventional caching fails because it does not separate the core intent of the query from the dynamic context. Agents may benefit from local (*i.e.,* individual query-level) optimizations, but miss opportunities for global improvements that leverage patterns across the entire task execution.

---

[1] Department of Computer Science, Stanford University, Stanford, CA, United States [2] Department of Electrical Engineering, Stanford University, Stanford, CA, United States. Correspondence to: Qizheng Zhang <qizhengz@stanford.edu>.

ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models at the International Conference on Machine Learning (ICML) 2025, Vancouver, Canada. Copyright 2025 by the author(s).

To overcome these limitations, we propose **agentic plan caching**, which reduces the serving costs of LLM-based agents that follow the Plan-Act paradigm by adapting and reusing prior execution plans across semantically similar workflows. Our key insight is that the Plan stage, which incurs the majority of LLM compute cost, is often repeated (within or across workflows) despite yielding outputs that could be reused in future requests while maintaining performance. When an agent completes an execution of a workflow, we extract structured **plan templates** from the agent execution log. When a similar request arrives, we employ **keyword extraction** to identify the most important semantic target of the query, then match it against the cache to retrieve the most relevant plan template. Our approach differs from semantic caching by avoiding query-based cache lookups, which we found sub-optimal for agent applications. Whenever additional planning is required, we utilize a lightweight model to adapt the cached structured plan template into more detailed plans with task-specific contexts (*e.g.,* fiscal year and company name in financial data-intensive reasoning (Narayan et al., 2025)), rather than employing an expensive model.

Although several memory architectures have been proposed to help agents store and learn from past experiences (Sumers et al., 2023; Xu et al., 2025; Wang et al., 2024b; Packer et al., 2023), these efforts primarily focus on using such memories to improve the agent's accuracy on completing workflows (*e.g.,* with fewer hallucinations (ama, a) or with higher task success rate (Wang et al., 2024b)) rather than to reduce the cost of serving the agent. To our knowledge, the use of historical experiences to more efficiently serve LLM-based agents remains underexplored, particularly for applications where outputs depend on input data or environmental conditions external to the query itself.

We evaluate agentic plan caching on two applications and find that it **reduces costs by 46.62% (on average) while maintaining 96.67% of optimal accuracy**. The agentic plan caching we propose is compatible with existing LLM serving and agent frameworks, and can be used jointly with existing caching techniques as well.

In summary, we make the following contributions:

1. **Analysis of Caching Techniques for Serving LLMs:** We conduct a comprehensive analysis of existing caching techniques for LLM serving (context caching and semantic caching), and point out why they are insufficient for the era of agentic AI applications.
2. **Proposal of Agentic Plan Caching:** We propose the idea of agentic plan caching, which shifts the focus from query-level caching (suitable for chatbots) to task-level caching (targeting LLM-based agents). We design and implement a novel caching system that extracts, stores, adapts and reuses agent-generated plans at test-time.
3. **Evaluation of Caching Techniques:** We evaluate our agentic plan caching system on top of real-world agent architecture and two datasets/benchmarks, and find that our approach can reduce cost by 46.62% (on average), while maintaining 96.67% of optimal application performance.

2. Background and Motivation

2.1. Plan-Act Agents

The rise of large language models (LLMs) has driven the rapid expansion of agentic AI applications. Unlike single-model tasks like chatbots (Chiang et al., 2024), math (Hendrycks et al., 2021), or coding (Chen et al., 2021), these applications coordinate multiple models and queries to solve complex tasks, like data-intensive reasoning (Narayan et al., 2025), software engineering (Zhang et al., 2025; Wei et al., 2024), web navigation (Zhou et al., 2023), etc.

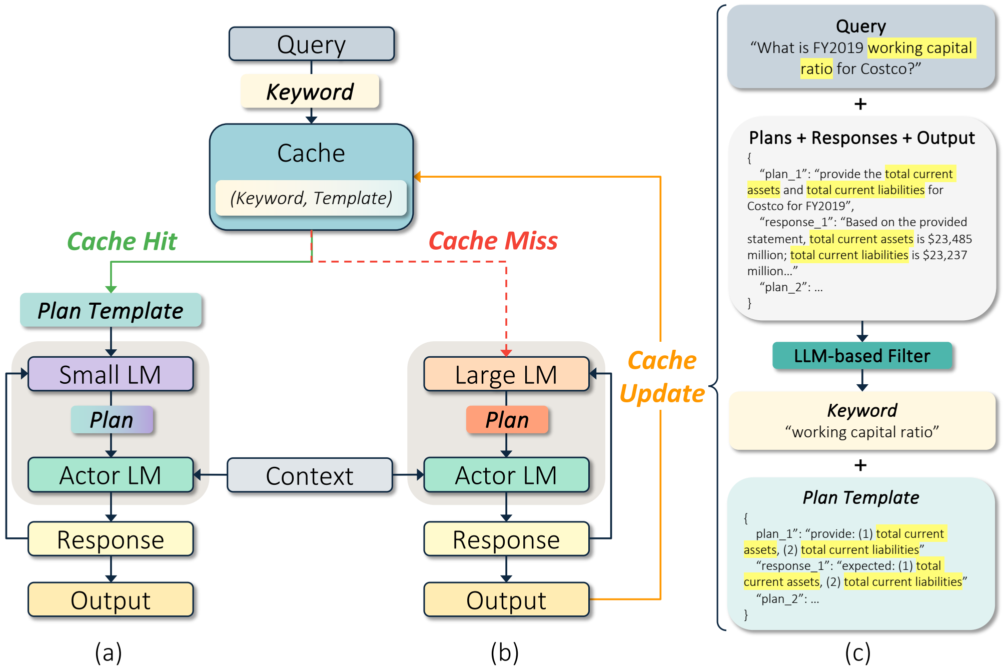
Many such agentic AI applications, like multi-agent systems (Wang et al., 2024a; Guo et al., 2024) and cloud-edge LLM systems (Zhang et al., 2024a; Narayan et al., 2025) , follow a two-stage pipeline loop (similar to the ReAct-agent loop (Yao et al., 2023)), as shown in Figure 3(a): (1) Plan and (2) Act. In the Plan stage, a planner LLM generates a strategy (*e.g.,* task decomposition, information retrieval) that guides subsequent actions of acting LLMs. In the Act stage, the actor LLM acts accordingly based on devised plans and external context or environment, and passes down the response to the planner LLM for the next step.

However, due to the use of multiple LLMs and queries, especially with advanced models like reasoning or multimodal LLMs, these agentic applications can incur significant costs (Jin et al., 2024b; Pan et al., 2025), particularly in terms of token ingestion/generation. Optimizing these costs is crucial for scaling agentic AI applications.

2.2. LLM Caching: Methods and Limitations

Caching is one of the most widely-adopted techniques for reducing the serving cost of LLM applications. The goal of caching is to eliminate redundant computation. Context caching (Gim et al., 2024; Yao et al., 2025; Yang et al., 2025), also known as KV cache reuse or prompt caching, involves storing and reusing the key-value pairs generated during the prefill phase of LLM inference. Semantic caching (Bang, 2023; Schroeder et al., 2025; ama, b), on the other hand, stores input-output pairs of previous LLM invocations. This relies on the fact that many prompts share similar underlying intents and thus expected outputs despite having different wording (Schroeder et al., 2025).

We find that existing caching techniques, primarily designed for serving **chatbots** (at query-level) instead of **agents** (at task-level), have three major limitations as described below.



"What is FY2019 working capital ratio for Costco?"

```
{
  "plan_1": "provide the total current assets and total current liabilities for Costco for FY2019",
  "response_1": "Based on the provided statement, total current assets is $23,485 million; total current liabilities is $23,237 million..."
  "plan_2": ...
}
```

"working capital ratio"

```
{
  plan_1": "provide: (1) total current assets, (2) total current liabilities"
  "response_1": "expected: (1) total current assets, (2) total current liabilities"
  "plan_2": ...
}
```

Figure 1. **Agentic Plan Caching Framework.** We show: (a) cache hit workflow, (b) cache miss workflow, and (c) plan template generation upon cache miss.

1) Model-Specific Constraints. Context caching relies on KV cache as the medium for storing and reusing knowledge (Liu et al., 2024b; Gim et al., 2024; Yang et al., 2025). These KV caches are inherently **model-dependent** and not easily transferable across different models (Wu et al., 2025; Liu et al., 2024a), since even identical text prompts produce model-specific KV caches. While this limitation is negligible for chatbots that consistently use a single model with the same system prompt, it becomes a problem for agentic AI applications that typically employ multiple LLMs across various processing stages.

2) Data-Dependent Outputs. Semantic caching stores input-output pairs from previous LLM calls, assuming outputs depend solely on input prompts (Bang, 2023; Schroeder et al., 2025). While this holds for chatbots, many agentic AI applications are **data-dependent**: Outputs depend not only on input queries but also on external data (*e.g.,* data-intensive reasoning (Narayan et al., 2025)) or dynamic environments (*e.g.,* web or GUI agents (Zhou et al., 2023; Wornow et al., 2024b;a)). This dependency complicates the reuse of cached responses even when input prompts are semantically similar.

3) Limited Adaptability. Both context and semantic caching lack flexibility for handling slight variations in input. Context caching requires exact text matches. Semantic caching, while more accommodating, does not capture the transformation process from prompt to response. This could hinder adaptation to similar queries with minor differences (*e.g.,* numeric values or variable names in mathematical reasoning (Cobbe et al., 2021), coding tasks (Jain et al., 2024)), a common challenge in agentic AI.

3. The Agentic Plan Caching Framework

We provide an end-to-end overview of the agentic plan caching framework in Figure 1. The process begins with a cost-effective language model (*e.g.,* GPT-4o-mini) extracting a keyword that captures the higher-level intent of the input task query (*e.g., "compute the average of all numbers listed in an external document" →"mean calculation"*). This keyword is then used to search the plan cache, which stores (keyword, plan template) pairs, potentially resulting in a cache hit or miss.

For a cache hit (Figure 1(a)), a small planner LM ("Small LM") adapts the retrieved plan template for the current execution by incorporating context-specific information (*e.g.,* user information, environment variables). For a cache miss (Figure 1(b)), a large planner LM ("Large LM") generates a new plan from the input task query. The adapted or generated plan, along with the task context (*e.g.,* external data or web/GUI environment), is then passed to the "Actor LM", which produces a response. The response is evaluated by the "Planner LM" to determine if further iterations are needed. If the task is complete, the final output is generated, concluding the agent's execution.

In the case of a cache miss, once the agent successfully completes execution with correct outputs, the system generates a

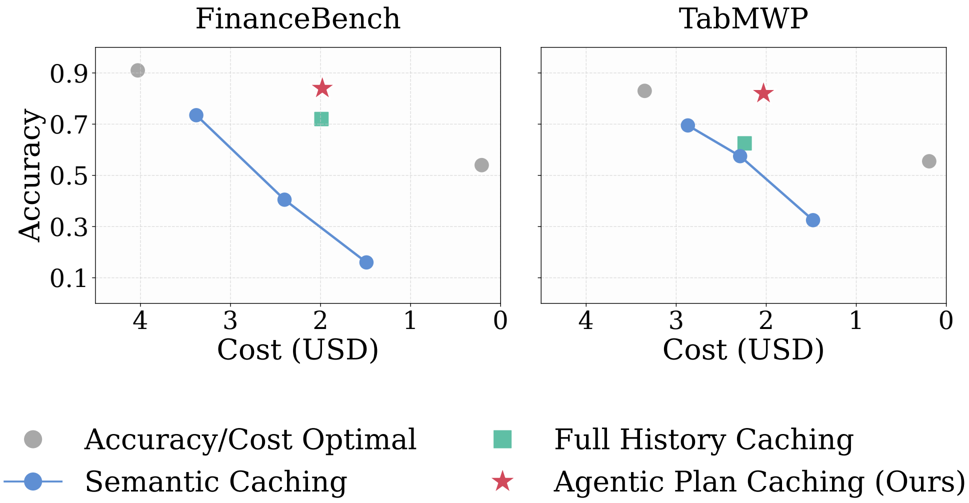


Figure 2. **Main Results across Four Baselines and Agentic Plan Caching.**

plan template that can be reused in future invocations of the agent through the following two-step process: (1) A rule-based filter extracts critical information from the execution log while discarding irrelevant details, such as verbose reasoning steps; (2) A lightweight LLM-based filter removes context-specific elements (*e.g.,* entity names, numeric values), producing a generalized template ("Plan Template") and relevant keywords for caching (Figure 1(c)).

In the Appendix, we provide additional algorithmic details and discuss the motivation behind design choices.

4. Results

We evaluate our agentic plan caching framework across two datasets focused on data-intensive reasoning, covering long-context financial data (FinanceBench (Islam et al., 2023)) as well as tabular mathematical data (Tabular Math Word Problems (Lu et al., 2022)).

Our key findings are:

- **Reduced Cost:** Agentic plan caching reduces LLM serving costs by an average of 46.62% (§4.2).
- **High Accuracy:** We maintain 96.67% of application-level performance compared to the accuracy-optimal baseline (§4.2).
- **Low Overhead:** On average, keyword extraction and cache generation account for only 1.04% of the overall cost of running each benchmark (§D.3).

4.1. Experiment Setup

Our agentic plan caching system is built on the Minion architecture (Figure 3(a)) from the Minions project (Narayan et al., 2025), a sequential Plan-Act LLM framework that can be readily generalized. The Minion architecture is composed of a large (cloud-hosted) planner LM for reasoning and task decomposition and a smaller (locally hosted) actor LM with access to additional context for plan execution. Given a task query, the planner LM and the actor LM collaborate iteratively (as in Figure 3(a)) to produce a final output. We set the maximum number of iterations to be 10.

We evaluate our system on two data-intensive reasoning benchmarks: FinanceBench (Islam et al., 2023) (financial and long-context data reasoning) and Tabular Math Word Problems (Lu et al., 2022) (tabular and mathematical data reasoning). Implementation details and dataset specifics are provided in the Appendix.

Evaluation Metrics We assess application-level performance using GPT-4o as the evaluation model, as LLM-based evaluation is more effective than exact matches or F1 scores for numeric evaluation and long-form responses (Chen et al., 2025; Goldie et al., 2025; Zheng et al., 2023). Cost is calculated based on input/output tokens and the latest API pricing from commercial LLM providers (OpenAI API (ope, a) and TogetherAI API (tog)). Additional details on evaluation models, prompts, and API pricing are included in the Appendix.

Language Models For the main results (§4.2), we use GPT-4o (gpt, a) as the planner LM and LLaMa-3.2-8B (lla) as both the small planner LM and actor LM. For keyword extraction and cache generation, we use GPT-4o-mini (gpt, b). To demonstrate broader applicability, we include a sensitivity analysis with a wider range of models in the Appendix.

Baselines We evaluate our system against the following baselines:

- **Accuracy-Optimal**: No caching is applied. The large planner LM is always used to establish the best achievable application performance.

- **Cost-Optimal:** No caching is applied. The small planner LM is consistently used to assess the lowest possible cost.
- **Semantic Caching**: We implement a query-level semantic caching method based on previous work (Bang, 2023; Schroeder et al., 2025). Following the approach of GPTCache (Bang, 2023)[1], we cache and reuse responses to individual queries, determining cache hits based on query-level similarity.
- **Full-History Caching** (discussed in §B.2): Inspired by knowledge caching in retrieval-augmented generation (Yao et al., 2025; Jin et al., 2024a), this baseline caches the complete agent execution log, including inputs and outputs of all LLM agent components. Cache hits are determined by keyword-level similarity. Upon a hit, the cached execution log is used as an in-context example for the small planner LM to generate new plans.

4.2. Results Analysis

Main Results As shown in Figure 2, agentic plan caching reduces cost by 46.62% on average while maintaining 96.67% of application-level performance compared to the accuracy-optimal baseline. We note that:

- **Semantic Caching:** Despite cost savings at lower similarity thresholds, semantic caching suffers from a high rate of false-positive cache hits, leading to substantial performance degradation. Additional case studies of false-positive hits are provided in the Appendix.
- **Full-History Caching:** While full-history caching preserves past plans and actions that might help plan generation for similar tasks, it underperforms agentic plan caching in accuracy (72.00% vs. 85.50% in FinanceBench) and incurs higher costs ($1.99 vs. $1.86). This is due to the small planner LM's difficulty in processing lengthy and unfiltered histories, emphasizing the necessity of our LLM-based filter to extract concise, reusable plan templates.

5. Conclusion

In this work, we introduce agentic plan caching, which shifts the focus from query-level caching (suitable for chatbots) to task-level caching (targeting LLM-based agents). By implementing agentic plan caching and evaluating it on two real-world agent workloads, we demonstrate that our approach reduces LLM serving costs by 46.62% on average while maintaining 96.67% of optimal application performance. Furthermore, the overhead introduced by plan caching remains minimal, accounting for only 1.04% (on average) of the total serving cost.

References

Reducing hallucinations in LLM agents with a verified semantic cache using Amazon Bedrock Knowledge Bases. `https://aws.amazon.com/blogs/machine-learning/reducing-hallucinations-in-llm-agents-with-a-verified-semantic-cache-using-amazon-bedrock-knowledge-bases/#:~:text=The%20semantic%20cache%20significantly%20reduces,handle%20unique%20questions%20when%20necessary.`, a.

Build a read-through semantic cache with Amazon OpenSearch Serverless and Amazon Bedrock. `https://aws.amazon.com/blogs/machine-learning/build-a-read-through-semantic-cache-with-amazon-opensearch-serverless-and-amazon-bedrock/`, b.

Gemini Deep Research . `https://gemini.google/overview/deep-research/?hl=en/`.

GPT-4o System Card . `https://openai.com/index/gpt-4o-system-card/`, a.

GPT-4o mini: advancing cost-efficient intelligence . `https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/`, b.

Llama 3.2: Revolutionizing edge AI and vision with open, customizable models . `https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/`.

OpenAI API Platform . `https://openai.com/api/`, a.

OpenAI deep research. `https://openai.com/index/deep-research/`, b.

api.together.ai . `https://api.together.xyz/`.

Bang, F. Gptcache: An open-source semantic cache for llm applications enabling faster answers and cost savings. In *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, pp. 212–218, 2023.

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

---

[1] We do not use the official GPTCache release as it (1) lacks support for post-GPT-4 OpenAI models and (2) relies on a deprecated version of the OpenAI API.

Chen, D., Yu, Q., Wang, P., Zhang, W., Tang, B., Xiong, F., Li, X., Yang, M., and Li, Z. xverify: Efficient answer verifier for reasoning model evaluations. *arXiv preprint arXiv:2504.10481*, 2025.

Chen, L., Zaharia, M., and Zou, J. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

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

Chiang, W.-L., Zheng, L., Sheng, Y., Angelopoulos, A. N., Li, T., Li, D., Zhu, B., Zhang, H., Jordan, M., Gonzalez, J. E., et al. Chatbot arena: An open platform for evaluating llms by human preference. In *Forty-first International Conference on Machine Learning*, 2024.

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

Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., Sun, H., and Su, Y. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023.

Gim, I., Chen, G., Lee, S.-s., Sarda, N., Khandelwal, A., and Zhong, L. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338, 2024.

Goldie, A., Mirhoseini, A., Zhou, H., Cai, I., and Manning, C. D. Synthetic data generation & multi-step rl for reasoning & tool use. *arXiv preprint arXiv:2504.04736*, 2025.

Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N. V., Wiest, O., and Zhang, X. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*, 2024.

He, H., Yao, W., Ma, K., Yu, W., Dai, Y., Zhang, H., Lan, Z., and Yu, D. Webvoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*, 2024.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Islam, P., Kannappan, A., Kiela, D., Qian, R., Scherrer, N., and Vidgen, B. Financebench: A new benchmark for financial question answering. *arXiv preprint arXiv:2311.11944*, 2023.

Jain, N., Han, K., Gu, A., Li, W.-D., Yan, F., Zhang, T., Wang, S., Solar-Lezama, A., Sen, K., and Stoica, I. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Ji, S., Li, G., Li, C., and Feng, J. Efficient interactive fuzzy keyword search. In *Proceedings of the 18th international conference on World wide web*, pp. 371–380, 2009.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

Jin, C., Zhang, Z., Jiang, X., Liu, F., Liu, X., Liu, X., and Jin, X. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *arXiv preprint arXiv:2404.12457*, 2024a.

Jin, Y., Li, J., Liu, Y., Gu, T., Wu, K., Jiang, Z., He, M., Zhao, B., Tan, X., Gan, Z., et al. Efficient multimodal large language models: A survey. *arXiv preprint arXiv:2405.10739*, 2024b.

Kwa, T., West, B., Becker, J., Deng, A., Garcia, K., Hasin, M., Jawhar, S., Kinniment, M., Rush, N., Von Arx, S., et al. Measuring ai ability to complete long tasks. *arXiv preprint arXiv:2503.14499*, 2025.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Liu, Y., Huang, Y., Yao, J., Gu, Z., Du, K., Li, H., Cheng, Y., Jiang, J., Lu, S., Musuvathi, M., et al. Droidspeak: Kv cache sharing for cross-llm communication and multi-llm serving. *arXiv preprint arXiv:2411.02820*, 2024a.

Liu, Y., Li, H., Cheng, Y., Ray, S., Huang, Y., Zhang, Q., Du, K., Yao, J., Lu, S., Ananthanarayanan, G., et al. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pp. 38–56, 2024b.

Liu, Z., Zhao, C., Iandola, F., Lai, C., Tian, Y., Fedorov, I., Xiong, Y., Chang, E., Shi, Y., Krishnamoorthi, R., et al. Mobilellm: Optimizing sub-billion parameter language models for on-device use cases. In *Forty-first International Conference on Machine Learning*, 2024c.

Lu, P., Qiu, L., Chang, K.-W., Wu, Y. N., Zhu, S.-C., Rajpurohit, T., Clark, P., and Kalyan, A. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. *arXiv preprint arXiv:2209.14610*, 2022.

Narayan, A., Biderman, D., Eyuboglu, S., May, A., Linderman, S., Zou, J., and Re, C. Minions: Cost-efficient collaboration between on-device and cloud language models. *arXiv preprint arXiv:2502.15964*, 2025.

Packer, C., Fang, V., Patil, S., Lin, K., Wooders, S., and Gonzalez, J. Memgpt: Towards llms as operating systems. 2023.

Pan, R., Dai, Y., Zhang, Z., Oliaro, G., Jia, Z., and Netravali, R. Specreason: Fast and accurate inference-time compute via speculative reasoning. *arXiv preprint arXiv:2504.07891*, 2025.

Patil, S. G., Zhang, T., Wang, X., and Gonzalez, J. E. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37: 126544–126565, 2024.

Sarukkai, V., Xie, Z., and Fatahalian, K. Self-generated in-context examples improve llm agents for sequential decision-making tasks. *arXiv preprint arXiv:2505.00234*, 2025.

Schroeder, L. G., Liu, S., Cuadron, A., Zhao, M., Krusche, S., Kemper, A., Zaharia, M., and Gonzalez, J. E. Adaptive semantic prompt caching with vectorq. *arXiv preprint arXiv:2502.03771*, 2025.

Snell, C., Lee, J., Xu, K., and Kumar, A. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Sumers, T., Yao, S., Narasimhan, K., and Griffiths, T. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2023.

Wang, J., Wang, J., Athiwaratkun, B., Zhang, C., and Zou, J. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692*, 2024a.

Wang, Z. Z., Mao, J., Fried, D., and Neubig, G. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024b.

Wei, A., Nie, A., Teixeira, T. S., Yadav, R., Lee, W., Wang, K., and Aiken, A. Improving parallel program performance through dsl-driven code generation with llm optimizers. *arXiv preprint arXiv:2410.15625*, 2024.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Wornow, M., Narayan, A., Opsahl-Ong, K., McIntyre, Q., Shah, N. H., and Re, C. Automating the enterprise with foundation models. *arXiv preprint arXiv:2405.03710*, 2024a.

Wornow, M., Narayan, A., Viggiano, B., Khare, I., Verma, T., Thompson, T., Hernandez, M., Sundar, S., Trujillo, C., Chawla, K., et al. Wonderbread: A benchmark for evaluating multimodal foundation models on business process management tasks. *Advances in Neural Information Processing Systems*, 37:115963–116021, 2024b.

Wu, G., Zhang, Z., Zhang, Y., Wang, W., Niu, J., Wu, Y., and Zhang, Y. I know what you asked: Prompt leakage via kv-cache sharing in multi-tenant llm serving. In *Proceedings of the 2025 Network and Distributed System Security (NDSS) Symposium. San Diego, CA, USA*, 2025.

Xu, W., Liang, Z., Mei, K., Gao, H., Tan, J., and Zhang, Y. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

Yang, J., Jimenez, C., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., and Press, O. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Yang, J., Hou, B., Wei, W., Bao, Y., and Chang, S. Kvlink: Accelerating large language models via efficient kv cache reuse. *arXiv preprint arXiv:2502.16002*, 2025.

Yao, J., Li, H., Liu, Y., Ray, S., Cheng, Y., Zhang, Q., Du, K., Lu, S., and Jiang, J. Cacheblend: Fast large language model serving for rag with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 94–109, 2025.

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

Zhang, G., Liang, W., Hsu, O., and Olukotun, K. Adaptive self-improvement llm agentic system for ml library development. *arXiv preprint arXiv:2502.02534*, 2025.

Zhang, Q., Imran, A., Bardhi, E., Swamy, T., Zhang, N., Shahbaz, M., and Olukotun, K. Caravan: Practical Online Learning of In-Network ML Models with Labeling Agents. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 325–345, 2024a.

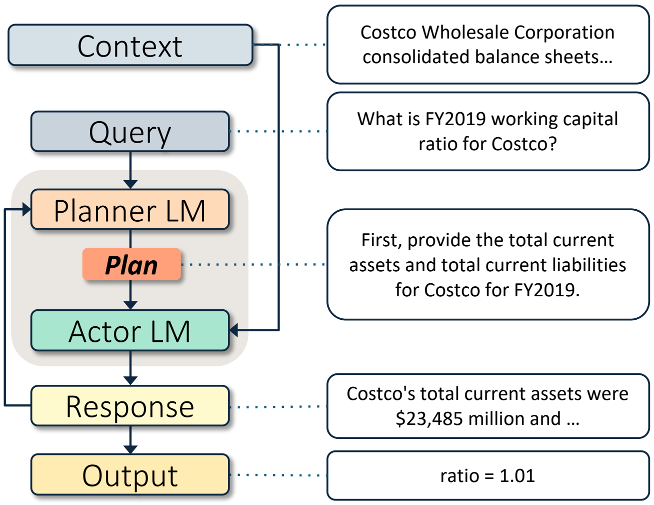
Zhang, Y., Sun, R., Chen, Y., Pfister, T., Zhang, R., and Arik, S. Chain of agents: Large language models collaborating on long-context tasks. *Advances in Neural Information Processing Systems*, 37:132208–132237, 2024b.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36: 46595–46623, 2023.

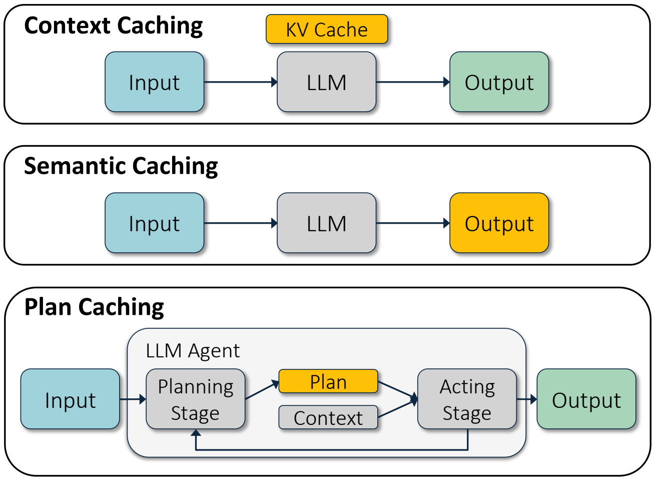
Zheng, L., Yin, L., Xie, Z., Sun, C. L., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., et al. Sglang: Efficient execution of structured language model programs. *Advances in Neural Information Processing Systems*, 37:62557–62583, 2024.

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

A. Extended Discussion of Background and Motivation


Context
Costco Wholesale Corporation consolidated balance sheets...
Query
What is FY2019 working capital ratio for Costco?
Planner LM
Plan
First, provide the total current assets and total current liabilities for Costco for FY2019.
Actor LM
Response
Costco's total current assets were $23,485 million and ...
Output
ratio = 1.01

(a) Plan-Act LLM application



(b) Comparison of different LLM caching techniques

Figure 3. **Plan-Act LLM Applications and Caching Techniques.** (a) A typical Plan-Act agent pipeline loop and (b) a comparison of LLM caching methods, with cached components highlighted in yellow.

B. Extended Discussion of Framework Design

B.1. Algorithms for Framework Design

In this section, we provide more algorithmic details of the agentic plan caching framework. To start with, the end-to-end workflow is provided in Algorithm 1. The cases of cache hit / miss are demonstrated in Algorithm 2 and Algorithm 3.

Algorithm 1 Agentic Plan Caching: End-to-End Framework

```
Require: Query $q$, Context $ctx$, Cache $C$
Ensure: Output $o$, Updated Cache $C'$
$keyword \leftarrow$ ExtractKeyword($q$)                    ▷ Extract keyword using a small LM
if $keyword \in C$ then                                      ▷ Cache hit (Figure 1(a))
    $o, C \leftarrow$ HandleCacheHit($q, ctx, C[keyword], C$)     ▷ Algorithm 2
else                                                         ▷ Cache miss (Figure 1(b))
    $o, C' \leftarrow$ HandleCacheMiss($q, ctx, keyword, C$)      ▷ Algorithm 3
end if
return $o, C'$                                ▷ Return response and possibly updated cache
```

Algorithm 2 Cache Hit

```
Require: Query $q$, Context $ctx$, Plan Template $template$, Plan Cache $C$
Ensure: Output $o$, Cache $C$
$responses \leftarrow \emptyset$                             ▷ Initialize actor LM response to be empty
$plan, o \leftarrow$ LightLM($q, template, responses$)       ▷ Adapt the retrieved template to be a task-specific plan using a lightweight model
Assert: $o$ is None
while $o$ is None do
    $response \leftarrow$ ActorLM($q, ctx, plan$)            ▷ Execute the plan based on context
    $responses \leftarrow responses \cup response$
    $plan, o \leftarrow$ LightLM($q, template, responses$)   ▷ Generate the final output or a new adapted plan
end while
return $o, C$
```

Algorithm 3 Cache Miss

```
Require: Query $q$, Context $ctx$, Plan Template $template$, Plan Cache $C$
Ensure: Output $o$, Updated Cache $C'$
$log \leftarrow \emptyset$                                   ▷ Initialize the execution log to be empty
$responses \leftarrow \emptyset$                             ▷ Initialize actor LM response to be empty
$plan, o \leftarrow$ PlannerLM($q, responses$)               ▷ Generate initial plan with full model
Assert: $o$ is None
while $o$ is None do
    $response \leftarrow$ ActorLM($q, ctx, plan$)            ▷ Execute the plan based on context
    $responses \leftarrow responses \cup response$
    $log \leftarrow log \cup \{(plan, ctx, response)\}$      ▷ Update the log
    $plan, o \leftarrow$ PlannerLM($q, responses$)           ▷ Generate the final output or a new plan
end while
$log \leftarrow log \cup \{o\}$                              ▷ Update the log
$template \leftarrow$ GenerateTemplate($log, keyword$)       ▷ Create reusable plan template based on execution log
$C' \leftarrow C$
$C'[keyword] \leftarrow template$                            ▷ Store template in cache
return $o, C'$
```

B.2. Design Choices

Why Keyword Extraction? A common method for identifying similar queries in a cache is to assess semantic or textual similarity, as seen in frameworks like GPTCache (Bang, 2023) which use embeddings for similarity searches. However, we find that *query-based similarity matching, despite its popularity, is insufficient for detecting cache hits/misses for agentic plan caching*. This is because it might overemphasize context-specific details (*e.g.,* names of individuals or companies) rather than the broader intent of queries, which makes it difficult to establish an effective similarity threshold (Schroeder et al., 2025). This often results in a high number of false positives (irrelevant cache hits) or false negatives (missed reuse opportunities). In contrast, extracting keywords that reflect the higher-level intent of queries provides a more reliable indicator of whether two queries would result in similar agentic plans, as illustrated in Figure 4.

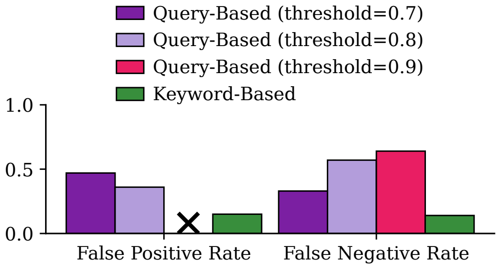


Figure 4. **Query-Based v. Keyword-Based Cache Search.** Keyword-based cache search achieves lower levels of false positive and false negative rates than query-based similarity cache search across different thresholds. This suggests that semantic similarity of queries alone may not effectively capture shared task intents and reusable plans.

Exact Matching v. Fuzzy Matching Our system uses exact matches (between keywords) to minimize false positives. While fuzzy search (Ji et al., 2009) (identifying cache hits based on similar but not identical keywords) could handle approximate key similarities and is feasible to integrate, we opted against it and leave it for future exploration for two main reasons: (1) Determining fuzzy matches based on semantic or textual similarity of keywords would reintroduce challenges faced by semantic caching, such as setting effective similarity thresholds, and (2) although lightweight LMs could potentially enable fuzzy matching, cache lookups must remain fast and cost-effective, particularly in low-hit-rate scenarios.

Caching Plan Template v. Caching Full Execution History One naive approach to reuse historical experience is to cache and reuse past agent execution logs (containing all inputs and outputs from planner and actor LMs) as in-context learning examples for the small planner LM. However, in our experiments (§4.2), we find that small planner LMs, usually based on small language models (*e.g.,* we use LLaMa-3.2-8B), struggle to handle long-context and unfiltered agent execution logs even when containing reusable plan information. This motivates us to filter agent execution logs into high-quality plan templates, and re-adapt them so that small planner LMs can better take advantage of their information.

C. Extended Description of Experiment Setup

C.1. Platform

The prototype of our agentic plan caching framework, which we use to run our experiments, is implemented on a Runpod server with dual-socket Intel Xeon Gold 6342 CPUs (96 vCPUs, 2.80GHz base clock, 3.5GHz max turbo) and 512MB total L1, 60MB L2, and 72MB L3 cache. The server supports AVX-512 and runs in a 2×48-core NUMA configuration. For memory, the server is equipped with 503GB of system RAM and no swap space.

C.2. LLM API Usage and Pricing

All language model inferences in our prototype are performed via third-party APIs. While it is feasible to run inference locally when model weights are available, we use API access to quantify cost in dollar terms for this study. If metrics such as latency or throughput were preferred, running all inferences locally would help eliminate variability introduced by external services, especially when they are hosted remotely. We use the Python APIs for OpenAI (v1.74.0), Together AI (v1.5.8), and Anthropic (v0.49.0). For all experiments, we set `temperature` to 0 (if supported) and `max_tokens` to 4096. Table 1 lists the per-token pricing of all models used in our experiments at the time of evaluation.

Model Name	API Provider	$ / Million Input Tokens	$ / Million Output Tokens
GPT-4o (`gpt-4o`)	OpenAI	2.50	10.00
GPT-4o-mini (`gpt-4o-mini`)	OpenAI	0.15	0.60
Claude 3.5 Sonnet (`claude-3-5-sonnet-20240620`)	Anthropic	3.00	15.00
Llama-3.1-8B (`Meta-Llama-3.1-8B-Instruct-Turbo`)	Together AI	0.18	0.18
Llama-3.2-3B (`Llama-3.2-3B-Instruct-Turbo`)	Together AI	0.06	0.06
Qwen-2.5-7B (`Qwen2.5-7B-Instruct-Turbo`)	Together AI	0.30	0.30

Table 1. **LLM API pricing used in our experiments.**

C.3. Datasets

FinanceBench. We use an augmented version of the FinanceBench test split from HuggingFace[2]. Following the Minions project, we filter for numerical reasoning questions and randomly sample 200 questions for evaluation. Each question requires long-context financial reasoning and is paired with a company-specific document essential for answering. The planner LM does not have access to the financial document, while the actor LM does.

TabMVP. We sample 200 questions from the test split of the TabMVP dataset provided by the authors[3]. Each question involves numeric reasoning and is paired with a required table; the question cannot be answered without the associated tabular data. The planner LM does not have access to the tabular data, while the actor LM does.

C.4. Prompts

C.4.1. AGENT PROMPTS

We use the same prompts from the Minion protocol in the Minions project (Narayan et al., 2025).

C.4.2. LLM-AS-A-JUDGE PROMPT

As discussed in the results section (§4), standard metrics like exact match or F1 score are often inadequate for evaluating numeric or long-form responses. For LLM-as-a-judge evaluation, we provide the prompt used to assess answer correctness. We closely follow the FinanceBench dataset's original evaluation criteria and define rules for acceptable numeric deviations according to what the FinanceBench dataset paper proposes, specifying what qualifies as a correct answer. These rules are applied consistently across both FinanceBench and TabMWP evaluations.

---

[2] `https://huggingface.co/datasets/virattt/financebench`

[3] `https://github.com/lupantech/PromptPG/blob/main/data/tabmwp/problems_test1k.json`

Correctness Evaluation Prompt: You are a judge that grades numeric answers to data-intensive reasoning problems.
This is the question: `{task}`.
This is the reference answer: `{gt_answer}`.
This is the answer given by a language model: `{response}`.
Please grade it. Requirements:
(1) Please allow minor deviations, such as
(i) giving the answer in billions when the unit was given in the question as millions.
(ii) giving the answer in percentage when the ground truth answer is floating point.
Please also allow small rounding errors or small numerical errors.
(2) Incorrect answers vary, from calculations that are off by small margins to several orders of magnitude, and from making up legal information to giving the wrong direction for an effect (e.g. reporting negative growth when it is actually positive).
(3) Just answer ’1’ for correct answers, or ’0’ for incorrect answers.

C.4.3. KEYWORD EXTRACTION PROMPT

Keyword Extraction Prompt: Can you help me summarize what is the ’task’ or ’keyword’ describing the higher-level goal or intent of this query? Please answer only with the task / keyword, which must be independent from problem-specific details.
`{query}`

C.4.4. CACHE GENERATION PROMPT

Cache Generation Prompt: You will see a filtered JSON trace that shows the complete workflow of how a planner language model solves a complex task by collaborating with an actor language model. Clean up the element of each item in the workflow, so that we can reuse this trace as a reference template (independent from problem-specific variables like company name or fiscal year) when we meet similar tasks later.
Requirements:
(1) the first element in each "workflow" item can only be "message", "output", or "answer",
(2) the task and the workflow should not contain problem-specific details or numbers, and
(3) return the result in JSON format that can be parsed by Python’s json.loads().
IMPORTANT: The workflow must maintain the sequence of message->loop(output->message/answer) to ensure proper functioning. Always start with a "message" and end with an "answer".
JSON trace: `{trace}`

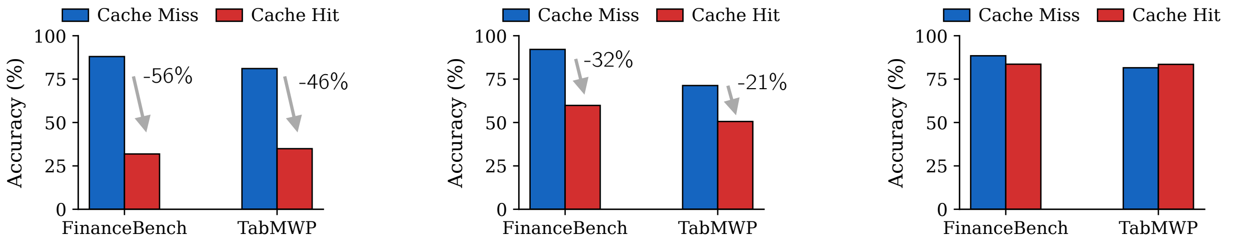
C.4.5. CACHE ADAPTATION PROMPT

Cache Adaptation Prompt: You are an intelligent language model that works with another model to solve complex tasks, like data-intensive reasoning questions.
Please construct a follow-up action plan (in the form of a message) based on the task and the reference template.
Reference task: `{cached_task}`
Reference follow-up action plan (as a message): `{next_item_in_cached_template}`
Your task is to adapt the reference follow-up message to the current context, maintaining the same inquiry structure but customizing it for the specific details of the current question and model output. Make sure the message asks for information not contained in past messages.
Format your response as a JSON object with a "reasoning" field set to "N/A" and a "message" field containing your action plan message.
Current task: `{task}`
Past action plans (as messages): `{past_messages}`
Past actor responses: `{past_actor_responses}`
Current message:

D. Extended Results

D.1. Ablation Study: How Different Caching Methods Impact Accuracy

Cache-Miss v. Cache-Hit Accuracy To assess the impact of caching on application performance, we compare cache-miss and cache-hit accuracy across semantic caching, full-history caching, and agentic plan caching (Figure 5). For semantic and full-history caching, cache-hit accuracy is significantly lower than cache-miss accuracy, indicating a performance trade-off despite potential cost savings. In contrast, agentic plan caching maintains consistent accuracy regardless of cache-use status, demonstrating its ability to preserve application performance without degradation.



(a) Semantic caching (b) Full-history caching (c) Agentic plan caching

Figure 5. **Accuracy Comparison across Caching Methods.** While semantic caching with threshold=0.9 in (a) and full-history caching in (b) experience notable accuracy drops during cache hits, agentic plan caching in (c) maintains stable performance across datasets.

D.2. Sensitivity Analysis: Results with More Models

We evaluate the robustness of agentic plan caching under different choices of large planner LMs, small planner LMs, and actor LMs. Our key findings are:

- **Consistent Gains Across Models:** Agentic plan caching consistently reduces cost and maintains high accuracy across a variety of model choices, beyond those presented in §4.
- **Model Selection Matters:** Despite consistent gains of our method, choosing the right model remains crucial. For example, in most cases, Claude 3.5 Sonnet outperforms GPT-4o in accuracy as the large planner LM but incurs significantly higher cost (Table 2). Similarly, smaller or cheaper models do not always yield better accuracy-cost tradeoffs. For example, using Llama-3.2-3B as the actor LM often leads to both higher cost and lower accuracy compared to Llama-3.1-8B due to insufficient response quality that triggers more Plan-Act iterations (Table 4).

Method	Large Planner LM	Small Planner LM	Actor LM	FinanceBench	TabMWP
				Cost↓ / Accuracy↑	Cost↓ / Accuracy↑
Accuracy-Optimal	GPT-4o	-	Llama-3.1-8B	$4.03 / 91.00%	$3.35 / 83.00%
Accuracy-Optimal	Claude 3.5 Sonnet	-	Llama-3.1-8B	$5.77 / 94.50%	$5.09 / 85.50%
Cost-Optimal	Llama-3.1-8B	-	Llama-3.1-8B	$0.21 / 54.00%	$0.19 / 55.50%
Cost-Optimal	Llama-3.2-3B	-	Llama-3.1-8B	$0.09 / 63.00%	$0.08 / 57.00%
Full-History Caching	GPT-4o	Llama-3.1-8B	Llama-3.1-8B	$1.99 / 72.00%	$2.24 / 62.50%
Full-History Caching	Claude 3.5 Sonnet	Llama-3.1-8B	Llama-3.1-8B	$3.13 / 68.00%	$2.80 / 65.00%
Agentic Plan Caching (Ours)	GPT-4o	Llama-3.1-8B	Llama-3.1-8B	$1.86 / 85.50%	$2.03 / 82.00%
Agentic Plan Caching (Ours)	Claude 3.5 Sonnet	Llama-3.1-8B	Llama-3.1-8B	$2.56 / 88.00%	$2.73 / 81.50%

Table 2. **Sensitivity Analysis of Large Planner LM: Results.**

Method	Large Planner LM	Small Planner LM	Actor LM	FinanceBench	TabMWP
				Cost↓ / Accuracy↑	Cost↓ / Accuracy↑
Full-History Caching	GPT-4o	Llama-3.1-8B	Llama-3.1-8B	\$1.99 / 72.00%	\$2.24 / 62.50%
Full-History Caching	GPT-4o	Qwen-2.5-7B	Llama-3.1-8B	\$2.34 / 72.50%	\$2.15 / 67.50%
Full-History Caching	GPT-4o	Llama-3.2-3B	Llama-3.1-8B	\$1.93 / 67.00%	\$1.67 / 56.00%
Agentic Plan Caching (Ours)	GPT-4o	Llama-3.1-8B	Llama-3.1-8B	\$1.86 / 85.50%	\$2.03 / 82.00%
Agentic Plan Caching (Ours)	GPT-4o	Qwen-2.5-7B	Llama-3.1-8B	\$1.66 / 90.00%	\$1.75 / 80.50%
Agentic Plan Caching (Ours)	GPT-4o	Llama-3.2-3B	Llama-3.1-8B	\$1.62 / 84.00%	\$1.88 / 80.00%

Table 3. **Sensitivity Analysis of Small Planner LM: Results.**

Method	Large Planner LM	Small Planner LM	Actor LM	FinanceBench	TabMWP
				Cost↓ / Accuracy↑	Cost↓ / Accuracy↑
Accuracy-Optimal	GPT-4o	-	Llama-3.1-8B	\$4.03 / 91.00%	\$3.35 / 83.00%
Accuracy-Optimal	GPT-4o	-	Qwen-2.5-7B	\$3.97 / 91.00%	\$3.06 / 87.50%
Accuracy-Optimal	GPT-4o	-	Llama-3.2-3B	\$4.16 / 81.50%	\$4.43 / 74.00%
Cost-Optimal	Llama-3.1-8B	-	Llama-3.1-8B	\$0.21 / 54.00%	\$0.19 / 55.50%
Cost-Optimal	Llama-3.1-8B	-	Qwen-2.5-7B	\$0.23 / 58.50%	\$0.17 / 65.50%
Cost-Optimal	Llama-3.1-8B	-	Llama-3.2-3B	\$0.17 / 53.00%	\$0.24 / 57.00%
Full-History Caching	GPT-4o	Llama-3.1-8B	Llama-3.1-8B	\$1.99 / 72.00%	\$2.24 / 62.50%
Full-History Caching	GPT-4o	Llama-3.1-8B	Qwen-2.5-7B	\$2.31 / 70.50%	\$1.77 / 60.50%
Full-History Caching	GPT-4o	Llama-3.1-8B	Llama-3.2-3B	\$2.20 / 60.00%	\$2.22 / 58.00%
Agentic Plan Caching (Ours)	GPT-4o	Llama-3.1-8B	Llama-3.1-8B	\$1.86 / 85.50%	\$2.03 / 82.00%
Agentic Plan Caching (Ours)	GPT-4o	Llama-3.1-8B	Qwen-2.5-7B	\$1.77 / 90.50%	\$1.76 / 84.00%
Agentic Plan Caching (Ours)	GPT-4o	Llama-3.1-8B	Llama-3.2-3B	\$1.85 / 80.00%	\$2.18 / 73.00%

Table 4. **Sensitivity Analysis of Actor LM: Results.**

D.3. Cost Analysis

Cost Breakdown We analyze the additional overhead introduced by the agentic plan caching mechanism through a cost breakdown analysis (Table 5). On average, keyword extraction and cache generation account for only 1.04% of the total cost. This minimal overhead is achieved because: (1) extracting higher-level goals or intents from task queries can be effectively handled by lightweight models at the scale of GPT-4o-mini or smaller, and (2) cache generation leverages rule-based methods to extract templates and uses a lightweight language model only for filtering out query-specific or context-specific details, which is a task well-suited to compact models.

Worst-Case Cache Overhead We assess the overhead incurred under the worst-case scenario, where the cache hit rate is zero. As shown in Table 5, even in this scenario, the cost from keyword extraction and cache generation is minimal (1.31% on average). In practical deployment, a potential mitigation strategy is to dynamically disable caching when hit rates remain persistently low.

Component	FinanceBench		TabMWP	
	Main Results	Worst Case	Main Results	Worst Case
Large Planner LM	\$1.7544 (94.17%)	\$3.9227 (97.36%)	\$1.9823 (97.76%)	\$3.3292 (98.33%)
Small Planner LM	\$0.0168 (0.90%)	–	\$0.0095 (0.47%)	–
Actor LM	\$0.0705 (3.78%)	\$0.0529 (1.31%)	\$0.0170 (0.84%)	\$0.0128 (0.38%)
Cache Overhead	**\$0.0213 (1.15%)**	**\$0.0535 (1.33%)**	**\$0.0190 (0.93%)**	**\$0.0438 (1.29%)**
- Keyword Extraction	\$0.0050 (0.27%)	\$0.0050 (0.13%)	\$0.0025 (0.12%)	\$0.0025 (0.07%)
- Cache Generation	\$0.0163 (0.88%)	\$0.0485 (1.20%)	\$0.0165 (0.81%)	\$0.0413 (1.22%)
Total	\$1.8630 (100%)	\$4.0291 (100%)	\$2.0278 (100%)	\$3.3858 (100%)

Table 5. **Cost Analysis.** We show the breakdown of agentic plan caching costs, including main results and worst-case overhead.

E. Example Agent Workflows and Cached Plan Templates

This section illustrates example agent workflows and their corresponding cached plan templates for data-intensive reasoning tasks. We begin with a simple single-turn task (more complex multi-turn examples are presented in our experiments):

Task: What is FY2019 working capital ratio for Costco? Define working capital ratio as total current assets divided by total current liabilities. Round your answer to two decimal places. Give a response to the question by relying on the details shown in the statement of financial position.
Context: `{costco_financial_documents}`

Below is the full agent workflow between a large planner LM and an actor LM, executed without caching (prompts omitted for brevity):

Planner: 1. Decompose the Task: The task requires calculating the working capital ratio for Costco for FY2019. This involves two main components: identifying the total current assets and the total current liabilities for FY2019 from the statement of financial position.
2. Explain Each Component:
- Identify Total Current Assets: This is important because it represents the resources that Costco can convert into cash within a year. Knowing this figure is essential for calculating the working capital ratio.
- Identify Total Current Liabilities: This represents the obligations that Costco needs to settle within a year. This figure is crucial for determining the working capital ratio as it provides the denominator in the calculation.
- Calculate Working Capital Ratio: This step involves dividing the total current assets by the total current liabilities. The result will give us the working capital ratio, which is a measure of Costco's short-term financial health.

3. Formulate a Focused Message: The next step is to ask the small language model for the specific figures of total current assets and total current liabilities for Costco in FY2019. This will allow us to perform the calculation.
4. Conclude with a Final Answer: Once we have the figures, we can calculate the working capital ratio and provide the final answer rounded to two decimal places.
Generated plan (as a message to the actor LM): Please provide the total current assets and total current liabilities for Costco for FY2019 from the statement of financial position.
Actor: Based on the provided statement of financial position for Costco Wholesale Corporation as of September 1, 2019, the total current assets and total current liabilities are:
- Total current assets: $23,485 million
- Total current liabilities: $23,237 million

Planner: 1. What information we have gathered: We have obtained the total current assets and total current liabilities for Costco as of September 1, 2019. The total current assets are $23,485 million, and the total current liabilities are $23,237 million.
2. Whether it is sufficient to answer the question: Yes, this information is sufficient to calculate the working capital ratio for Costco for FY2019.
3. If not sufficient, what specific information is missing: Since we have both the total current assets and total current liabilities, no additional information is needed.
4. If sufficient, how we would calculate or derive the answer: To calculate the working capital ratio, we divide the total current assets by the total current liabilities. The formula is:
Working Capital Ratio = Total Current Assets / Total Current Liabilities
Substituting the given values: Working Capital Ratio = $23,485 million / $23,237 million
Calculating this gives us approximately 1.01 when rounded to two decimal places.
Conclusion: The working capital ratio for Costco for FY2019 is 1.01.

The following plan template is extracted from this successful workflow and stored in the cache:

Keyword: working capital ratio
Plan Template: 1. Plan (round 1): Retrieve total current assets and total current liabilities from the statement of financial position.
2. Expected response from actor LM (round 1): Total current assets and total current liabilities.
3. Plan (round 2): The working capital ratio can be determined by:
Working Capital Ratio = Total Current Assets / Total Current Liabilities.

Later, for tasks with the same extracted keyword, such as the example below:

Task: What is FY2021 working capital ratio for Best Buy? Define working capital ratio as total current assets divided by total current liabilities. Round your answer to two decimal places. Please base your judgments on the information provided primarily in the statement of financial position.
Context: `{bestbuy_financial_documents}`

We reuse the cached plan template and adapt it using a small planner LM:

Planner: (Using cached plan template) Please provide the total current assets and total current liabilities for Best Buy in FY2021 from the statement of financial position, so I can calculate the working capital ratio.
Actor: According to the Consolidated Balance Sheets, the total current assets for Best Buy in FY2021 are $12,540 million, and the total current liabilities are $10,521 million.
Planner: (Using cached plan template) The FY2021 working capital ratio for Best Buy is calculated by dividing the total current assets by total current liabilities. With total current assets of $12,540 million and total current liabilities of $10,521 million, the working capital ratio is $12,540 million / $10,521 million = 1.19. Therefore, the working capital ratio is 1.19, rounded to two decimal places.

Using the cached template significantly shortens the agent execution log, reducing token usage for expensive planner LMs. This efficiency gain comes from:

- Avoiding redundant planning for repeated tasks.
- Knowing when sufficient information has been gathered to terminate the workflow, thus avoiding unnecessary Plan-Act iterations.

F. Extended Discussion of Related Work, Limitations, Societal Implications, and Future Work

F.1. Related Work

Agent Memory Prior work has explored augmenting LLM agents with external memory to (1) reduce hallucinations through context-aware responses (ama, a) and (2) enable complex, long-horizon tasks (Wang et al., 2024b). Some studies focus on defining memory formats (Yao et al., 2023) and managing memory efficiently (Packer et al., 2023; Xu et al., 2025). While our caching system can be adapted as a form of agent memory, it diverges by targeting serving cost reduction rather than enhanced capability, a largely unexplored area.

LLM Serving Engines Existing LLM serving engines like vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024) optimize general query inference at scale through techniques such as KV cache management and request scheduling. Our approach is compatible with these systems, extending their capabilities to incorporate cost-effective caching for agentic AI scenarios.

F.2. Limitations, Societal Implications, and Future Work

Limitations and Challenges First, we focus on two-stage Plan-Act agent architecture in this work. More complex multi-agent systems could present new challenges for maintaining cache consistency across multiple components. Second, for highly dynamic workloads with frequent task variations, the benefits of caching may diminish as historical plans may be less applicable. Finally, our evaluation primarily emphasizes cost reduction. Future work could consider additional system metrics such as latency, throughput, and computational overhead.

Broader Impact and Societal Implications We believe that the proposed agentic plan caching framework has broader implications for AI accessibility and democratization. By reducing LLM serving costs, this framework could enable smaller enterprises, academic institutions, and individual developers to deploy agentic AI systems without incurring prohibitive API costs. Additionally, plan caches generated by advanced, commercial LLMs could potentially be shared or adapted for use with open-source models (as shown in our experiments), facilitating greater access to state-of-the-art agentic capabilities without direct reliance on expensive, closed-source APIs (*e.g.,* from OpenAI). This approach also raises questions about the long-term impact on data privacy, especially in cases where plan caches contain sensitive or proprietary information. Ensuring cache privacy and data security in LLM agents requires further research.

Future Directions Several future directions could extend the utility of agentic plan caching. First, more advanced cache look-up and plan adaptation methods (like retrieval-augmented generation) might further enhance the relevance of cached plans in complex workflows. Second, enabling user-configurable cache parameters (*e.g.,* cache size, eviction strategies, fuzzy matching policies) could provide more control over caching strategies and allow for tailored cost-performance trade-offs. Finally, integrating the idea of agentic plan caching into existing LLM and agent serving frameworks at production scale would further enhance its applicability and impact. Overall, we hope this work inspires further research on optimizing the efficiency and cost-effectiveness of agentic AI systems.