Agentic Plan Caching: Test-Time Memory for Fast and Cost-Efficient LLM Agents

Qizheng Zhang **Michael Wornow** **Kunle Olukotun**

Stanford University

Abstract

LLM-based agent applications have shown increasingly remarkable capabilities in complex workflows but incur substantial costs and latency 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 agent applications where outputs depend on external data and environmental contexts. We propose **Agentic Plan Caching (APC)**, a novel **test-time memory** that extracts, stores, adapts, and reuses structured plan templates from planning stages of agent applications across semantically similar tasks to reduce the cost and latency 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 agent applications shows that our system can reduce costs by 50.31% and latency by 27.28% on average while maintaining performance, offering a more efficient solution for serving LLM-based agents that complements existing LLM serving infrastructures.

1 Introduction

Agent applications based on Large Language Models (LLMs) have shown early promise in replicating human performance on a broad range of workflows, from coding [26, 28, 61] to web navigation [23, 70] to open-ended research [3, 8] to social interactions [42, 57]. Many of these LLM-based agents follow a **two-stage pipeline**, often referred to as the *ReAct*-agent loop [63, 45], that alternates between: **(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 [67, 31] and need to interact with external tools and environments [43]. Specifically, the Plan stage is often implemented via test-time compute techniques [13, 47] like chain-of-thought reasoning [53], 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 [15, 39] or locally [35].

To reduce LLM costs, methods have been developed to optimize responses to individual queries [32, 69]. In particular, *caching* has emerged as a popular approach, with two primary implementations: **Context caching** (*e.g.,* KV cache reuse and prompt caching [20, 60, 62]) stores internal model states to speed up subsequent generations, while **semantic caching** [10, 46, 2] 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,

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

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.

To overcome these limitations, we propose **Agentic Plan Caching (APC)**, a novel **test-time memory** that 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 [39]), rather than employing an expensive model.

Although several memory architectures have been proposed to help agents store and learn from past experiences [49, 40, 51, 59, 65], these efforts primarily focus on using such memories to improve the agent's accuracy on completing workflows (*e.g.,* with fewer hallucinations [9] or with higher task success rate [51]) 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 five diverse agent workloads and find that it **reduces costs by 50.31% and latency by 27.28% (on average) while maintaining 96.61% 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 five datasets/benchmarks, and find that our approach can reduce cost by 50.31% and latency by 27.28% (on average), while maintaining 96.61% of optimal application performance.

2 Background and Motivation

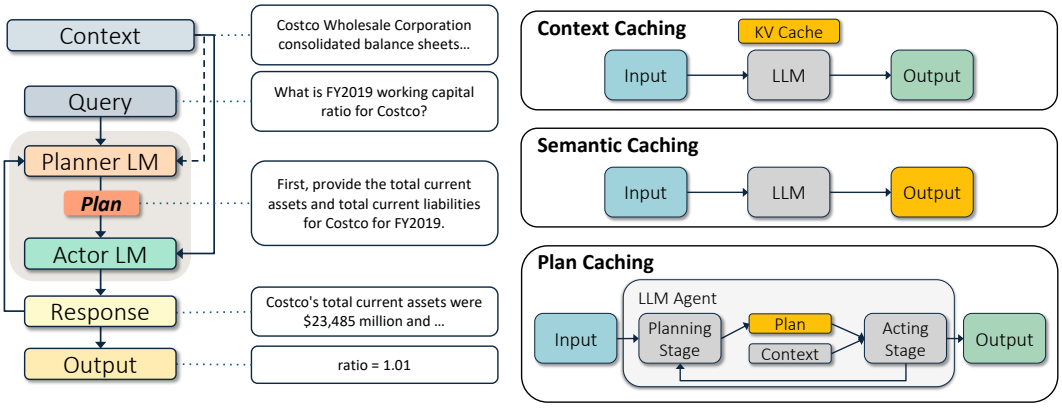
2.1 Plan-Act Agents

The rise of large language models (LLMs) has driven the rapid expansion of agentic AI applications. Unlike single-model tasks like chatbots [17], math [24], or coding [16], these applications coordinate multiple models and queries to solve complex tasks, like data-intensive reasoning [39], software engineering [64, 52], web navigation [70], etc.

Many such agentic AI applications, like multi-agent systems [50, 22] and cloud-edge LLM systems [66, 39] , follow a two-stage pipeline loop (similar to the ReAct-agent loop [63]), as shown in Figure 1a: (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 [30, 41], particularly in terms of token ingestion/generation. Optimizing these costs is crucial for scaling agentic AI applications.



(a) Plan-Act LLM application (b) Comparison of different LLM caching techniques

Figure 1: **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.

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 [20, 62, 60, 58], 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 [10, 46, 2], 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 [46].

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.

1) Model-Specific Constraints. Context caching relies on KV cache as the medium for storing and reusing knowledge [34, 20, 60]. These KV caches are inherently **model-dependent** and not easily transferable across different models [56, 33], 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 [10, 46]. 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 [39]) or dynamic environments (*e.g.,* web or GUI agents [70, 55, 54]). 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 [18], coding tasks [26]), a common challenge in agentic AI.

2.3 Related Work

Agent Memory Prior work has explored augmenting LLM agents with external memory to (1) reduce hallucinations through context-aware responses [9] and (2) enable complex, long-horizon tasks [51]. Some studies focus on defining memory formats [63] and managing memory efficiently [40, 59]. 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 [32] and SGLang [69] 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.

Case-Based Planning Case-Based Planning (CBP) [11, 48, 12] is a problem-solving paradigm where new plans are produced by adapting previously solved cases rather than constructing plans from scratch. By exploiting similarities between past and current situations, CBP supports efficient plan reuse, continual learning, and incremental refinement. While our work shares the high-level intuition of "plan storage and reuse", it targets a fundamentally different setting from classic symbolic CBP: LLM-based, neural agents performing open-ended natural-language actions. Specifically, our system (1) automatically extracts reusable plan templates at test time from unconstrained LLM generations, instead of relying on hand-crafted symbolic plans, and (2) leverages these stored plans for cost-efficient inference in LLM agents.

3 The Agentic Plan Caching (APC) Framework

In this section, we provide an end-to-end overview of our agentic plan caching framework (§3.1) and then discuss the motivations behind key design choices (§3.2).

3.1 Overview of System Design

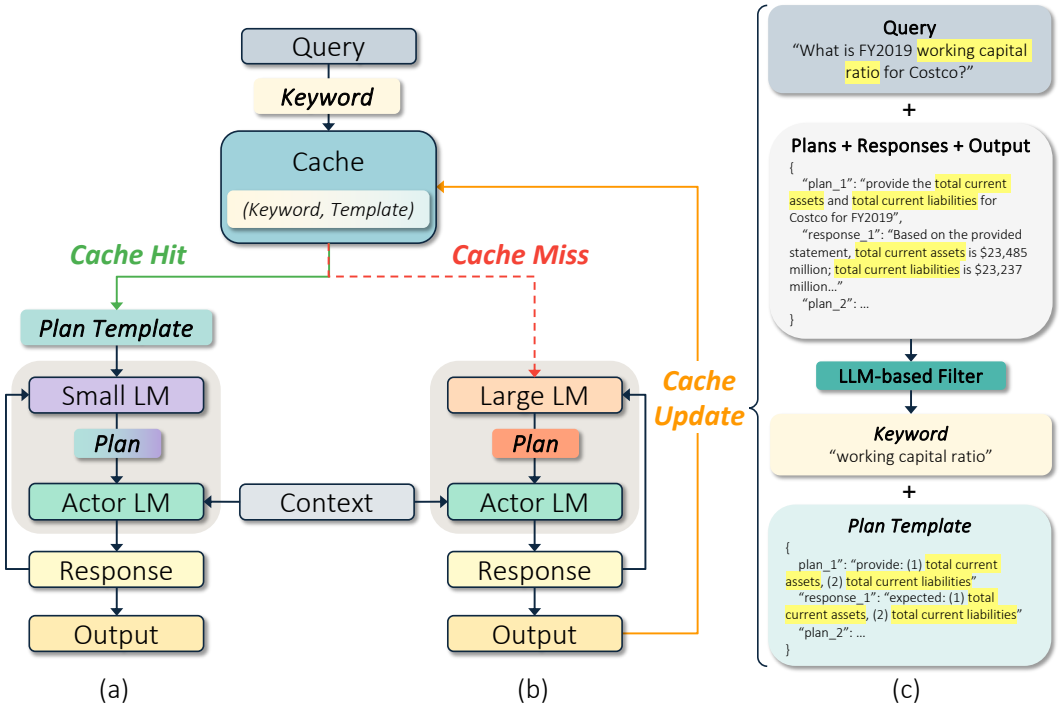


Figure 2: **Agentic Plan Caching Framework.** We show: (a) cache hit workflow, (b) cache miss workflow, and (c) plan template generation for new cache entries.

We provide an end-to-end overview of the agentic plan caching framework in Figure 2. 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"* $\rightarrow$*"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 2(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 2(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 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 2(c)).

Additional algorithmic details are provided in the Appendix.

3.2 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 [10] 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 [46]. 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 3.

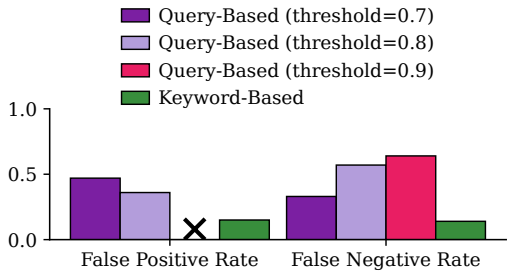


Figure 3: **Query-Based v. Keyword-Based Cache Search.** Keyword-based cache search achieves lower levels of false positive and false negative rates than query-based similarity cache search across different thresholds. This suggests that semantic similarity of queries alone may not effectively capture 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 [27] (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. We present an in-depth analysis of the overhead and scalability of fuzzy matching in §4.4.

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-3.1-8B), 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.

4 Results and Evaluation

We evaluate our agentic plan caching (APC) framework on five agent workloads that span a diverse range of data-intensive reasoning and agentic capabilities. These include long-context data reasoning (FinanceBench [25], QASPER [19]), mathematical reasoning (Tabular Math Word Problems [36], AIME 2024 and 2025 [37]), and multi-step agentic reasoning and tool use (GAIA [38]).

Our key findings are:

- **Reduced Cost:** APC reduces agent serving costs by an average of 50.31% (§4.2).
- **Reduced Latency:** APC reduces agent serving latency by an average of 27.28% (§4.3).
- **High Accuracy:** APC maintains 96.61% 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 (§4.3).

4.1 Experiment Setup

Our agentic plan caching system is built on the Minion architecture (Figure 1a) from the Minions project [39], 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 1a) to produce a final output. We set the maximum number of iterations to be 10.

While we adopt this architecture for clarity, APC is not restricted to Minion-like agent architecture; we demonstrate its integration into other agent architectures and report end-to-end results in §4.2. Additional implementation details and dataset specifications 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 [14, 21, 68]. Cost is calculated based on input/output tokens and the latest API pricing from commercial LLM providers (OpenAI API [7] and TogetherAI API [1]). 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 [5] as the planner LM and LLaMa-3.1-8B [6] as both the small planner LM and actor LM. For keyword extraction and cache generation, we use GPT-4o-mini [4]. 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 [10, 46]. Following the approach of GPTCache [10][1], we cache and reuse responses to individual queries, determining cache hits based on query-level similarity. We set similarity thresholds to be 80%, 85%, and 90%; a lookup is considered a hit if the query-level similarity is above this threshold.
- **Full-History Caching** (discussed in §3.2): Inspired by knowledge caching in retrieval-augmented generation [62, 29], 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 Main Results

As shown in Figure 4 and Table 1, agentic plan caching reduces cost by 50.31% on average while maintaining 96.61% 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.

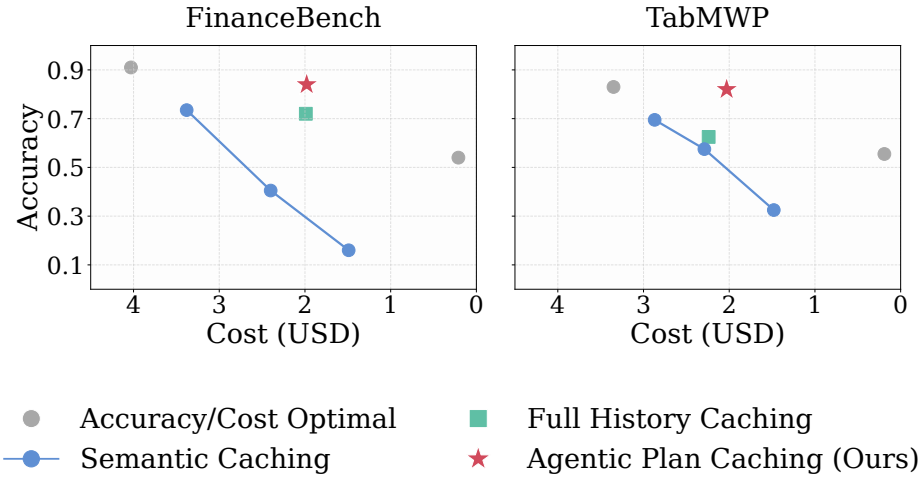


Figure 4: **Results across Four Baselines and Agentic Plan Caching.**

Results on GAIA with Open Deep Research Agent Beyond Minion-based architectures, we integrate APC into the Open Deep Research agent from the Hugging Face `smolagents` library [44] using GPT-4o as the large planner LM and GPT-4o-mini as the small planner LM. As shown in Table 1, on the GAIA benchmark [38], APC achieves a 76.42% reduction in cost (from $69.02 to $16.27) with only a 0.61% drop in accuracy (37.58% to 36.97%), demonstrating strong cost-efficiency even in complex, open-domain agent settings.

A closer analysis reveals that GAIA's heterogeneous task space, ranging from video dialog reasoning to sales computation, limits the effectiveness of keyword-based cache retrieval, as many task descriptions are highly specific and rarely recur. Despite fewer cache hits during initial planning,

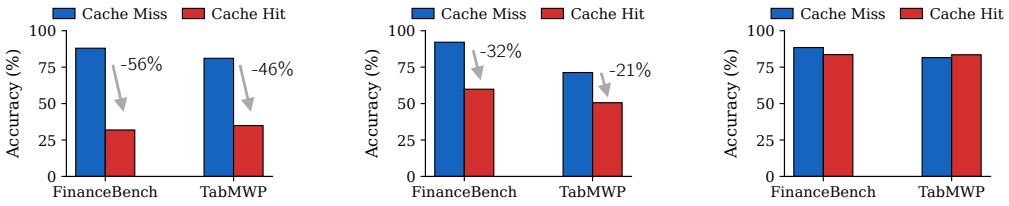
[1]We do not use the official GPTCache release as it (1) lacks support for post-GPT-4 OpenAI models and (2) relies on a deprecated version of the OpenAI API.

Method	Minion			Open Deep Research
	QASPER	AIME 2024	AIME 2025	GAIA
	Cost↓ / Accuracy↑	Cost↓ / Accuracy↑	Cost↓ / Accuracy↑	Cost↓ / Accuracy↑
Accuracy-Optimal	\$2.14 / 58.00%	\$1.14 / 64.52%	\$1.34 / 61.29%	\$69.02 / 37.58%
Cost-Optimal	\$0.21 / 53.00%	\$0.65 / 48.39%	\$0.60 / 48.39%	\$3.16 / 19.39%
APC (Ours)	\$0.78 / 57.00%	\$0.85 / 61.29%	\$0.81 / 58.06%	\$16.27 / 36.97%

Table 1: **More Results.** We evaluate APC on a diverse set of benchmarks covering reasoning and agentic capabilities, as well as agent architecture like Minion and Open Deep Research.

APC improves efficiency in re-planning phases by reusing prior plan structures, thereby reducing redundant large-model invocations.

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.

4.3 Cost and Speed Analysis

Cost Breakdown We analyze the additional overhead introduced by the agentic plan caching mechanism through a cost breakdown analysis (Table 2). 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 2, 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.

Latency Analysis We evaluated the wall-clock latency of our system and compared it to both accuracy-optimal and cost-optimal baselines. Additionally, we provide a detailed breakdown of latency incurred by each component in our plan caching pipeline. This microbenchmark is based on 100 randomly sampled queries from the FinanceBench dataset, with a cache hit rate of 46% (*i.e.,* cached plans were used for 46 of the 100 queries). As shown in the Table 3, APC reduces end-to-end latency by 27.28% on this workload. Most of the additional latency in our system comes from LLM-powered cache generation, which takes an average of 3.99 seconds per entry. To further

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 2: **Cost Analysis.** We show the breakdown of agentic plan caching costs, including main results and worst-case overhead (*i.e.,* zero cache hit rate).

Method	Plan	Act	Keyword Extraction	Cache Lookup	Cache Generation	Total
Accuracy-Optimal	1813.41	94.39	–	–	–	1959.24
Cost-Optimal	856.75	93.31	–	–	–	1004.79
APC (Ours)	1011.82	131.44	42.29	<1	215.80	1424.82

Table 3: **Latency Analysis.** Breakdown of wall-clock latency across components of the agent pipeline. All values are measured in seconds.

mitigate this overhead, we (1) automatically disable caching when the hit rate is consistently low, and (2) are actively exploring optimizations such as parallel cache generation and speculative next-query inference as part of future work. To summarize, our system offers a favorable performance trade-off: We achieve significantly lower cost than the accuracy-optimal baseline while preserving high accuracy at the cost of moderate latency, most of which is attributable to one-time cache generation.

4.4 Scalability and Cache Management

Effect of Cache Size Increasing cache size generally reduces both cost and latency, up to a point of diminishing returns. On the FinanceBench dataset, larger caches yield higher hit rates and lower end-to-end latency, as fewer entries need to be regenerated after eviction. Table 4 reports results using a simple LRU eviction policy. Beyond a certain capacity, approximately exceeding the number of unique task keywords, further enlarging the cache offers minimal benefit. In practice, users can tune cache capacity based on the desired trade-off between speed and storage.

Cache Size	Hit Rate	Cost	Accuracy	Planning Latency (s)	Cache Gen. Latency (s)	Total Latency (s)
1	2%	$3.97	92.00%	1638.85	383.87	2232.76
10	13%	$3.51	88.00%	1381.89	334.61	1911.95
20	28%	$2.95	85.00%	1248.06	289.36	1772.61
50	45%	$1.88	86.00%	1015.12	204.99	1459.92
100	46%	$1.86	85.50%	1011.81	215.80	1424.82

Table 4: **Effect of Cache Size.** Larger caches improve hit rate and reduce overall cost and latency until reaching a point of diminishing returns.

Exact Matching v. Fuzzy Matching For exact match lookups, our cache uses Python's built-in dictionary, which provides highly optimized $O(1)$ average-case lookup and insertion. To evaluate empirical performance, we measured wall-clock latency for cache hits and misses across varying cache sizes. Each measurement was averaged over 100 trials with CPU caches cleared before every run. As shown in Table 5, exact matching maintains consistently low latency up to 10^6 entries.

In contrast, fuzzy matching introduces substantial cache lookup latency that scales poorly with cache size. We implement fuzzy keyword matching in our prototype and use a semantic-similarity model (`SentenceTransformer('all-MiniLM-L6-v2')`). As shown in Table 5, this approach is orders of magnitude slower than exact matching, confirming the computational overhead of semantic search.

Cache Size	Exact Matching		Fuzzy Matching	
	Cache Hit Latency (μs)	Cache Miss Latency (μs)	Cache Hit Latency (μs)	Cache Miss Latency (μs)
10^2	13	14	57	24
10^3	15	15	75	70
10^4	16	17	581	554
10^5	22	18	10388	10317
10^6	56	37	148449	148147

Table 5: **Cache Lookup Scalability.** Fuzzy matching incurs much higher latency and scales poorly compared to exact matching. Averages over 100 trials; similarity threshold set to be 0.8.

Similarity Threshold	Hit Rate	Cost	Accuracy	Planning Latency (s)	Total Latency (s)
$= 100\%$	46%	\$1.86	85.50%	1011.82	1424.82
$> 80\%$	54%	\$1.15	83.00%	875.31	1219.73
$> 60\%$	64%	\$0.93	77.00%	720.29	1044.50

Table 6: **Fuzzy Keyword Matching Results.** Lower similarity thresholds increase hit rate and reduce cost and latency, but degrade accuracy.

We also find that under fuzzy keyword matching, lowering the similarity threshold increases the cache hit rate and reduces cost and latency, but at the expense of accuracy (Table 6). This highlights the inherent trade-off in fuzzy matching: While more aggressive matching improves efficiency, it risks introducing less relevant cached plans. Our cache interface remains flexible—users can enable fuzzy matching and tune thresholds according to their application's tolerance for semantic drift and latency-accuracy trade-offs.

4.5 Cold Start

Cold start is an inherent limitation of *test-time* plan caching (as opposed to offline caching), since the cache begins empty. In the early phase, APC experiences higher latency and cost due to frequent cache misses and the need to generate new entries. To quantify this effect, we perform a time-series analysis of cache warm-up, shown in Table 7. As the cache grows, hit rate steadily increases, leading to lower marginal cost and latency over time. In practice, if the target workload is known in advance, users can mitigate cold-start overhead by pre-populating the cache with offline samples before deployment.

Query Percentile	# Cache Entries	Hit Rate	Cost	Planning Latency (s)	Cache Gen. Latency (s)	Total Latency (s)
20th	15	14.29%	\$0.59 (32.07%)	260.19 (27.43%)	59.19 (27.64%)	358.71 (25.72%)
40th	27	24.39%	\$0.97 (52.72%)	484.88 (51.12%)	130.29 (60.86%)	689.49 (49.44%)
60th	36	36.07%	\$1.20 (65.22%)	638.12 (67.28%)	167.89 (78.41%)	926.82 (66.46%)
80th	42	40.75%	\$1.63 (88.59%)	820.94 (86.56%)	195.59 (91.35%)	1183.39 (84.86%)
100th	46	48.00%	\$1.84 (100.00%)	948.36 (100.00%)	214.11 (100.00%)	1394.56 (100.00%)

Table 7: **Cold Start Behavior.** As the cache warms up, hit rate increases and marginal cost and latency decrease. Pre-warming the cache with offline samples can further mitigate cold-start overhead.

5 Conclusion

We introduce **Agentic Plan Caching (APC)**, which shifts the focus from query-level caching (suitable for chatbots) to task-level caching (targeting LLM-based agents). Unlike traditional semantic caching, APC extracts plan templates from completed agent executions at test time, uses keyword-based retrieval to match new queries to cached plans, and leverages lightweight models to adapt these templates into context-specific task plans. By implementing agentic plan caching and evaluating it on five diverse agent workloads and two Plan-Act agent architectures, we demonstrate that our approach reduces agent serving costs by 50.31% and latency by 27.28% (on average) while maintaining 96.61% 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.

6 Acknowledgment

We thank all the anonymous reviewers and the area chair for their insightful feedback and suggestions, which significantly enhanced the quality of this work. Qizheng Zhang is supported in part by NSF CNS-2211384. Michael Wornow is supported by the NSF Fellowship, a Stanford HAI Graduate Fellowship, and Stanford Healthcare. We thank Gerry Wan for his contribution to the Open Deep Research experiments in this paper after the initial submission. We thank Avanika Narayan for helpful discussions on Minions, and thank Hanchen Li, Zhiqiang Xie, Jon Saad-Falcon, Azalia Mirhoseini, and the LMCache team for helpful discussions on the project idea and its presentation.

References

[1] api.together.ai . https://api.together.xyz/.

[2] 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/.

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

[4] GPT-4o mini: advancing cost-efficient intelligence . https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/.

[5] GPT-4o System Card . https://openai.com/index/gpt-4o-system-card/.

[6] 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/.

[7] OpenAI API Platform . https://openai.com/api/.

[8] OpenAI deep research. https://openai.com/index/deep-research/.

[9] 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.

[10] Fu Bang. 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)*, pages 212–218, 2023.

[11] Ralph Bergmann and Wolfgang Wilke. On the role of abstraction in case-based reasoning. In *European Workshop on Advances in Case-Based Reasoning*, pages 28–43. Springer, 1996.

[12] Daniel Borrajo, Anna Roubíčková, and Ivan Serina. Progress in case-based planning. *ACM Computing Surveys (CSUR)*, 47(2):1–39, 2015.

[13] Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

[14] Ding Chen, Qingchen Yu, Pengyuan Wang, Wentao Zhang, Bo Tang, Feiyu Xiong, Xinchi Li, Minchuan Yang, and Zhiyu Li. xverify: Efficient answer verifier for reasoning model evaluations. *arXiv preprint arXiv:2504.10481*, 2025.

[15] Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

[16] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[17] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael Jordan, Joseph E Gonzalez, et al. Chatbot arena: An open platform for evaluating llms by human preference. In *Forty-first International Conference on Machine Learning*, 2024.

[18] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[19] Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*, 2021.

[20] In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338, 2024.

[21] Anna Goldie, Azalia Mirhoseini, Hao Zhou, Irene Cai, and Christopher D Manning. Synthetic data generation & multi-step rl for reasoning & tool use. *arXiv preprint arXiv:2504.04736*, 2025.

[22] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*, 2024.

[23] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*, 2024.

[24] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[25] Pranab Islam, Anand Kannappan, Douwe Kiela, Rebecca Qian, Nino Scherrer, and Bertie Vidgen. Financebench: A new benchmark for financial question answering. *arXiv preprint arXiv:2311.11944*, 2023.

[26] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

[27] Shengyue Ji, Guoliang Li, Chen Li, and Jianhua Feng. Efficient interactive fuzzy keyword search. In *Proceedings of the 18th international conference on World wide web*, pages 371–380, 2009.

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

[29] Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *arXiv preprint arXiv:2404.12457*, 2024.

[30] Yizhang Jin, Jian Li, Yexin Liu, Tianjun Gu, Kai Wu, Zhengkai Jiang, Muyang He, Bo Zhao, Xin Tan, Zhenye Gan, et al. Efficient multimodal large language models: A survey. *arXiv preprint arXiv:2405.10739*, 2024.

[31] Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, et al. Measuring ai ability to complete long tasks. *arXiv preprint arXiv:2503.14499*, 2025.

[32] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion 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.

[33] Yuhan Liu, Yuyang Huang, Jiayi Yao, Zhuohan Gu, Kuntai Du, Hanchen Li, Yihua Cheng, Junchen Jiang, Shan Lu, Madan Musuvathi, et al. Droidspeak: Kv cache sharing for cross-llm communication and multi-llm serving. *arXiv preprint arXiv:2411.02820*, 2024.

[34] Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, et al. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 38–56, 2024.

[35] Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, et al. Mobilellm: Optimizing sub-billion parameter language models for on-device use cases. In *Forty-first International Conference on Machine Learning*, 2024.

[36] Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. *arXiv preprint arXiv:2209.14610*, 2022.

[37] Mathematical Association of America (MAA). American Invitational Mathematics Examination (AIME). https://maa.org/.

[38] Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023.

[39] Avanika Narayan, Dan Biderman, Sabri Eyuboglu, Avner May, Scott Linderman, James Zou, and Christopher Re. Minions: Cost-efficient collaboration between on-device and cloud language models. *arXiv preprint arXiv:2502.15964*, 2025.

[40] Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: Towards llms as operating systems. 2023.

[41] Rui Pan, Yinwei Dai, Zhihao Zhang, Gabriele Oliaro, Zhihao Jia, and Ravi Netravali. Specreason: Fast and accurate inference-time compute via speculative reasoning. *arXiv preprint arXiv:2504.07891*, 2025.

[42] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22, 2023.

[43] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565, 2024.

[44] Alexandre Roucher, Adrià Villanova del Moral, Merve, Thomas Wolf, and Clément Fourrier. Open Deep Research: Freeing Our Search Agents. https://huggingface.co/blog/open-deep-research, February 2025. Hugging Face Blog.

[45] Vishnu Sarukkai, Zhiqiang Xie, and Kayvon Fatahalian. Self-generated in-context examples improve llm agents for sequential decision-making tasks. *arXiv preprint arXiv:2505.00234*, 2025.

[46] Luis Gaspar Schroeder, Shu Liu, Alejandro Cuadron, Mark Zhao, Stephan Krusche, Alfons Kemper, Matei Zaharia, and Joseph E Gonzalez. Adaptive semantic prompt caching with vectorq. *arXiv preprint arXiv:2502.03771*, 2025.

[47] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

[48] Luca Spalzzi. A survey on case-based planning. *Artificial Intelligence Review*, 16(1):3–36, 2001.

[49] Theodore Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2023.

[50] Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692*, 2024.

[51] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

[52] Anjiang Wei, Allen Nie, Thiago SFX Teixeira, Rohan Yadav, Wonchan Lee, Ke Wang, and Alex Aiken. Improving parallel program performance through dsl-driven code generation with llm optimizers. *arXiv preprint arXiv:2410.15625*, 2024.

[53] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[54] Michael Wornow, Avanika Narayan, Krista Opsahl-Ong, Quinn McIntyre, Nigam H Shah, and Christopher Re. Automating the enterprise with foundation models. *arXiv preprint arXiv:2405.03710*, 2024.

[55] Michael Wornow, Avanika Narayan, Ben Viggiano, Ishan Khare, Tathagat Verma, Tibor Thompson, Miguel Hernandez, Sudharsan Sundar, Chloe Trujillo, Krrish Chawla, et al. Wonderbread: A benchmark for evaluating multimodal foundation models on business process management tasks. *Advances in Neural Information Processing Systems*, 37:115963–116021, 2024.

[56] Guanlong Wu, Zheng Zhang, Yao Zhang, Weili Wang, Jianyu Niu, Ye Wu, and Yinqian Zhang. 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.

[57] Zhiqiang Xie, Hao Kang, Ying Sheng, Tushar Krishna, Kayvon Fatahalian, and Christos Kozyrakis. Ai metropolis: Scaling large language model-based multi-agent simulation with out-of-order execution. *arXiv preprint arXiv:2411.03519*, 2024.

[58] Zhiqiang Xie, Ziyi Xu, Mark Zhao, Yuwei An, Vikram Sharma Mailthody, Scott Mahlke, Michael Garland, and Christos Kozyrakis. Strata: Hierarchical context caching for long context language model serving. *arXiv preprint arXiv:2508.18572*, 2025.

[59] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

[60] Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. Kvlink: Accelerating large language models via efficient kv cache reuse. *arXiv preprint arXiv:2502.16002*, 2025.

[61] John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

[62] Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving for rag with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 94–109, 2025.

[63] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[64] Genghan Zhang, Weixin Liang, Olivia Hsu, and Kunle Olukotun. Adaptive self-improvement llm agentic system for ml library development. *arXiv preprint arXiv:2502.02534*, 2025.

[65] Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025.

[66] Qizheng Zhang, Ali Imran, Enkeleda Bardhi, Tushar Swamy, Nathan Zhang, Muhammad Shahbaz, and Kunle Olukotun. Caravan: Practical Online Learning of In-Network ML Models with Labeling Agents. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 325–345, 2024.

[67] Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Arik. Chain of agents: Large language models collaborating on long-context tasks. *Advances in Neural Information Processing Systems*, 37:132208–132237, 2024.

[68] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

[69] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in Neural Information Processing Systems*, 37:62557–62583, 2024.

[70] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims we make in the abstract and in the introduction session (§1) accurately reflect the research problem we target, the solution we propose, and the contribution of our work.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We present a discussion of the limitations of this work in Appendix E.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: This paper does not contain theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We present our experiment setup in §4.1, which contains all necessary information to reproduce the main results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The data and code of this work will be open-sourced upon publication of the paper.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: Experiment settings, along with why they are set up in the current way, are disclosed in §4.1 and in the Appendix.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We properly discuss the statistical significance of our results in §4 and in the Appendix.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We report the resources we use for running the experiments (e.g. compute resources, AI API resources) in §4.1 and in the Appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: We confirm that the research conducted in the paper conform with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We present a discussion of the broader impact and societal implications of this work in Appendix E.

    Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: This paper does not involve the release of data or models that have risks for misuse.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We credit the models and the datasets we use directly via citations/URLs in §4.1 and in the Appendix.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: This paper does not introduce new assets (except for data and code, which will be open-sourced upon publication of the paper).

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing experiments or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing experiments or research with human subjects.

    Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: In §3, we describe how LLMs are used in our system as a novel method for certain tasks (e.g. data filtering).

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.

A Extended Discussion of Methods

A.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.

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 2(a))
    $o, C \leftarrow$ HandleCacheHit$(q, ctx, C[keyword], C)$ ▷ Algorithm 2
else ▷ Cache miss (Figure 2(b))
    $o, C' \leftarrow$ HandleCacheMiss$(q, ctx, keyword, C)$ ▷ Algorithm 3
end if
return o, C' ▷ Return response and possibly updated cache

The case of cache hit is demonstrated in Algorithm 2.

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

The case of cache miss is demonstrated in Algorithm 3.

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 Extended Description of Experiment Setup

B.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.

B.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 8 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 8: **LLM API pricing used in our experiments.**

B.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.

B.4 Prompts

B.4.1 Agent Prompts

We use the same prompts from the Minion protocol in the Minions project [39].

B.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

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

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

FinanceBench dataset paper proposes, specifying what qualifies as a correct answer. These rules are applied consistently across both FinanceBench and TabMWP evaluations.

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.

B.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}`

B.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}`

B.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:

C Extended Results

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 9). 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 11).

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 9: **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 10: **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 11: **Sensitivity Analysis of Actor LM: Results.**

D 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.

E Extended Discussion of Limitations, Societal Impact, and Future Directions

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.