HARNESS-AGNOSTIC PREDICTIVE KV CACHE PREFETCH ENGINE FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-running LLM agents alternate between model inference and tool execution, creating idle periods during which cached prefixes may be evicted. Subsequent requests can also encounter uncached tool outputs or cold-started subagents. Because inference servers do not observe tool execution, returned results, or subagent creation before the next request arrives, they cannot use these gaps to prepare the KV cache. We introduce a harness-agnostic predictive KV-cache prefetch engine that uses agent-runtime lifecycle events to move KV-cache preparation off the critical path. During tool execution, the engine issues low-priority warmups for the known prefix; after completion, it warms the anticipated next-request prefix with the actual tool result. It also prepares initial prefixes for newly spawned subagents. A contextual Thompson sampling policy selects whether and how to prefetch based on cache state, prefix length, and predicted tool duration, balancing cache reuse against warmup cost and foreground interference. Across Claude Code, Hermes, and Codex on GAIA and SWE-bench Lite, the engine reduces aggregate time to first token by 67.6%, 19.6%, and 68.9%, respectively. Subagent warming reduces median first-call TTFT by 68.6%, from 564 ms to 177 ms. At eight concurrent sessions, warmups add 0.5% prompt-token overhead. These results demonstrate that harness-visible execution gaps can be used to prepare anticipated KV state and reduce cold-prefill latency in long-running agents.

1 INTRODUCTION

KV-cache prefix reuse is a key optimization in LLM inference. When requests share a prompt prefix, systems such as vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024) reuse cached KV tensors instead of recomputing them. This is especially valuable for agents, whose contexts grow across repeated model calls (Pan et al., 2025; Bian et al., 2025).

Tool-calling agents alternate between model generation and external tool execution (Yao et al., 2023), creating three sources of cache misses. During a tool call, cache pressure may evict the session prefix; the returned output introduces uncached tokens; and a spawned subagent may start on a worker without the relevant context. The next model call may therefore require a cold prefill over an increasingly long prompt, compounding latency across turns.

Serving-layer approaches address only part of this problem. TTL pinning (Li et al., 2025) preserves existing KV blocks but cannot prefill future tool output. Server-side speculative prefetching (Zheng et al., 2026) is limited to the interval before that output arrives. Moreover, serving systems do not natively observe tool results or subagent creation, and supporting these events requires additional inference-framework integration.

We instead perform predictive KV-cache warming in the agent harness: the runtime layer that coordinates model calls, tool execution, and subagents. NeMo Relay exposes standardized lifecycle intercepts across agent frameworks. Our plugin uses these intercepts to issue low-priority warmup requests to a Dynamo-managed inference backend while tools execute. After the tool returns, it issues a result-aware prefix warmup, which we call PASTE or Predictive Augmentation with Saved Tool Executions: a low-priority request that prefills the anticipated next-request prefix, including the actual tool output. A per-tool duration model schedules each warmup, a Markov next-tool predictor records likely future context, and a Thompson Sampling bandit (Thompson, 1933) suppresses actions likely to contend with foreground traffic. No serving-layer source-code modification is required.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

This paper contributes **(1)** a harness-level architecture addressing eviction, tool-output, and subagent cache misses; **(2)** adaptive warmup scheduling and selection using duration estimation, next-tool prediction, and Thompson Sampling; and **(3)** evaluation across Claude Code, Hermes, and Codex, using two integration architectures and workloads from GAIA and SWE-bench Lite.

2 Background and Motivation

2.1 KV cache prefix reuse in transformer inference

Transformer inference consists of a prefill phase, which computes key–value (KV) tensors for the input tokens, and an autoregressive decode phase, which reuses them while generating tokens (Vaswani et al., 2017). Prefix caching extends this reuse across requests: if an exact token prefix is cached on the selected worker, the server can reuse its KV blocks instead of recomputing them (Kwon et al., 2023; Zheng et al., 2024). Otherwise, the request incurs a cold prefill whose time-to-first-token (TTFT) cost grows with prefix length. Figure 8 in Appendix A quantifies this difference in a separate microbenchmark.

2.2 Agentic workloads and the tool execution gap

ReAct-style agents alternate between model generation and external tool execution (Yao et al., 2023). Each tool invocation creates a *tool gap*, during which the agent issues no model request while waiting for the result. Table 2 in Appendix A shows that the evaluated harnesses exhibit long-tail gaps that can accommodate background warmup, whereas short gaps may not. This variation motivates the completion-time gate in Section 3.1. Agent contexts also typically retain prior turns, making much of the next request predictable before the tool completes.

2.3 Three KV-Cache miss types

Cold KV misses in agentic workloads arise from three structurally different causes:

Eviction misses. Concurrent traffic may evict a session's cached prefix during a tool gap, forcing the next request to recompute it. TTL pinning (Li et al., 2025) and cache-retention policies such as InferCept (Abhyankar et al., 2024) target this case.

Tool-output misses. A tool result introduces tokens that do not exist until the tool completes. Retention cannot cache these tokens, and prediction cannot prefill their exact content before available.

Subagent cold starts. A spawned subagent may be assigned to a worker without its required prefix. Without advance notice of the spawn and its context, the serving layer can warm that prefix only after receiving the first request.

Server-side speculative prefetching (Zheng et al., 2026) can warm predicted prefixes between model calls, but it does not natively observe tool results or subagent creation. These harness-visible events provide additional time and information for addressing the three miss types (summarized in Figure 1).

3 System Design

The prefetch engine is implemented as a NeMo Relay plugin at the agent runtime layer. It observes tool lifecycle events and issues background-hinted warmup requests to a Dynamo deployment during tool execution. This placement exposes tool and subagent state that is not available from model requests alone.

We support two integration paths. In Relay-instrumented, in-process harnesses, the plugin registers tool and LLM intercepts directly. For CLI-based harnesses such as Claude Code and Codex, a local lifecycle-hook service supplies tool events, while an inference proxy captures and normalizes the requests sent to Dynamo.

A warmup is an ordinary nonstreaming request with one output token, annotated with Dynamo agent hints. The effect of these hints depends on the configured router and backend scheduler, and they do not preempt work already executing. Our deployment enables Dynamo KV-aware routing, vLLM

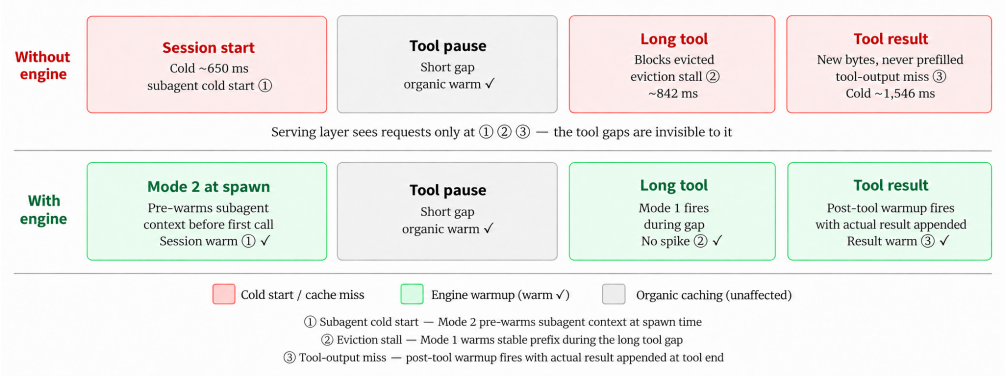


Figure 1: Three structural KV-cache miss types and the engine's response.

prefix caching, and worker KV-event publication. In the replay experiments, warmups and their corresponding foreground requests additionally carry the same `backend_instance_id`, ensuring that both reach the worker holding the warmed blocks.

3.1 The warmup firing decision

Over-firing consumes prefill capacity and may delay foreground inference, so each candidate passes four admission controls.

Cache-state check. The engine suppresses warming when the latest phase-specific KV-hit fraction is at least 0.85, indicating a hot prefix. At the initial intra-turn decision, observations older than 60 seconds are considered stale. Because the engine receives no eviction notifications, this check remains heuristic.

Concurrency cap. The in-process cap limits concurrent warmups to three, decreasing to two above 12,000 estimated prefix tokens and one above 20,000. This adaptive cap reduces the number of simultaneous warmups as prefix size, and therefore prefill work, increases.

Completion-time gate. The engine estimates warmup time from the candidate prefix length and an exponential moving average (EMA) of observed prefill throughput. After three throughput observations, it suppresses a warmup when the estimated prefill time exceeds 0.7 times the predicted tool duration. Tool duration is predicted from the historical p75 for the tool and argument class, falling back to a per-session duration EMA.

Measurement guard. Warmups are suppressed while a designated measurement request is in flight, preventing the engine's own background work from contaminating that measurement.

Eligible calls are assigned to 27 contexts using 3 buckets each for previous KV-hit fraction 0.5/0.85, prefix length 2,000/8,000 tokens, and predicted tool duration 0.5/5 seconds. Thompson sampling selects among four actions: skip; warm the captured prefix; warm a block-aligned truncated prefix; or warm the captured prefix while separately recording a Markov next-tool prediction. The latter uses the same warmup payload as the standard action; the Markov prediction is recorded separately and does not add predicted tokens to the warmup. Before tool completion, firing actions warm only the known contiguous prefix; afterward, a result-aware warmup includes the actual tool output.

Each context–action pair maintains a $\mathrm{Beta}(\alpha, \beta)$ distribution, initialized with $\alpha = \beta = 1$. With cache telemetry, the selected-action utility is

$$R = \mathrm{clip}_{[-0.5,1]}\left(h - \min\left(\frac{W}{P}, 0.2\right) - 0.2\,\mathbf{1}\left[T > 1.3\bar{T}\right]\right),$$

where h is the next request's KV-hit fraction, W is the number of prompt tokens sent in warmup requests, P is the number processed by foreground requests, T is the observed TTFT of the next foreground request, and $\bar{T}$ is its no-warmup session baseline TTFT. Without cache telemetry, utility is the normalized TTFT improvement over a no-warmup session baseline. The engine sets $s = (R + 0.5)/1.5$ and updates $\alpha \leftarrow \alpha + s$ and $\beta \leftarrow \beta + 1 - s$. For the skip action, $W = 0$, and its

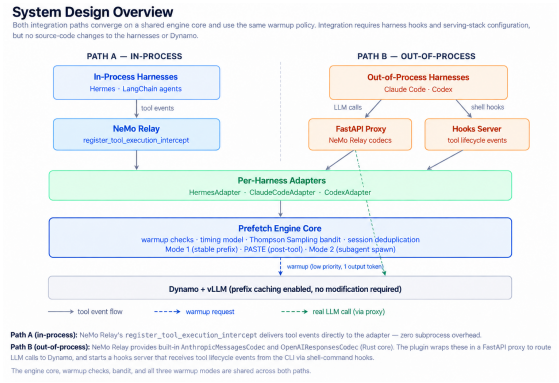


Figure 2: Prefetch engine system design.

posterior is updated from the subsequent foreground outcome in the same way. Section 4.3 evaluates this learned policy separately from the admission controls.

3.2 Warmup scheduling within the tool gap

For admitted candidates, the engine schedules warmups using online tool-duration and prefill-throughput estimates. For a prefix of L tokens and estimated prefill throughput r_{prefill}, the predicted warmup time is

$$\hat{t}_{\text{warm}} = \frac{L}{r_{\text{prefill}}},$$

which is used by the 70% completion-time check in Section 3.1. For admitted warmups, the engine schedules the request at

$$t_{\text{fire}} = t_{\text{start}} + \max\left(0, \hat{t}_{\text{tool}}^{(p75)} - \hat{t}_{\text{warm}}\right),$$

where $\hat{t}_{\text{tool}}^{(p75)}$ is the predicted p75 tool duration. This targets completion near the predicted tool end, though estimation error or queueing may cause foreground overlap.

3.3 Warmup modes

The engine provides three modes corresponding to the miss types in Figure 1.

Mode 1 (stable-prefix warmup). During tool execution, Mode 1 warms the prefix available at tool start: the system prompt, tool schemas, and conversation history through the latest assistant turn. This prefix is unaffected by the pending tool result and targets eviction misses.

PASTE (post-tool warmup). At tool completion, PASTE appends the actual result and warms the resulting prefix, targeting tokens unavailable before execution. The post-tool warmup is dispatched asynchronously and does not delay release of the subsequent foreground request; when their execution overlaps, the warmup remains background-priority. For repeated calls, the engine may reuse a cached result only for tools explicitly classified as cache-safe; MCP read-only, idempotent, and closed-world annotations serve as classification signals rather than proof of deterministic output. A cache hit allows PASTE to run at tool start and use the full execution gap.

Mode 2 (subagent-spawn warmup). When an orchestrator first invokes an unseen subagent type, the engine captures its initial on-wire prefix, including the subagent's system context and associated tools, for reuse. On subsequent spawns of that subagent type, Mode 2 retrieves this prefix and warms the subagent's reusable initial context before its first model call.

All modes use the one-token, background-hinted requests described in Section 3.

3.4 Session deduplication and worker affinity

An in-flight prefix-hash registry coalesces matching concurrent warmups into one request, allowing other sessions to await the same result. A second registry records the worker that served each prefix hash and expires entries after 10 minutes. When a matching prefix recurs, the engine routes its warmup to the recorded worker, increasing the chance that its KV blocks remain resident. In the replay experiments, warmup and foreground requests carry the same backend-instance identifier, pinning both to the same worker but not guaranteeing that the blocks have avoided eviction. Figure 16 details the engine's internal architecture.

4 Evaluation

The main evaluation experiments run on a single $8\times$H100 NVL node serving Llama-3.1-Nemotron-70B through Dynamo with a vLLM backend. We first execute benchmark tasks through each agent harness and record the resulting on-wire request traces. We then replay these traces under controlled cache conditions, preserving the prompts, tool results, and request order while isolating serving-side cache effects from agent nondeterminism. We use three general cache conditions: **A**, cache-busted requests with a near-zero KV-hit fraction; **B**, native prefix caching under concurrent NoiseRunner load; and **C**, our engine under the same load. Appendix B describes the protocol, and Table 1 summarizes the results.

4.1 Claude Code on GAIA

We evaluate Claude Code on GAIA-100 (Mialon et al., 2024) because its Bash-heavy trajectories expose both eviction and tool-output misses. On GAIA-100 under ($N = 16$) concurrent load, the engine reduces cumulative TTFT across all model calls by 67.6%, from 11,527 s to 3,735 s. 60 queries show clear benefit, 36 are neutral, and 4 are marginally harmed by less than 0.16 s each. On the longest workflow, spanning 50 sequential model calls, cumulative TTFT falls from 29.25 s to 5.34 s (81.7%), showing how the benefit compounds as context grows.

Bash tool calls have median and p95 durations of 8.6 s and 45s, respectively (Table 2). Under concurrent load, these gaps permit prefix eviction, while each returned result introduces new tokens. Accordingly, the post-Bash KV-hit fraction is only 0.003 without the engine. PASTE raises it to 0.999, makes approximately 6,993 of a 7,000-token stable prefix reusable, and reduces TTFT by approximately 466 ms per call. For repeated cache-safe tool calls, result reuse allows PASTE to fire at tool start and use the entire tool gap.

The largest gains occur after long tool gaps. For gaps exceeding 90 s, PASTE's TTFT reduction is $12.8\times$ the eviction-only penalty because it also warms newly returned tool tokens. For shorter gaps, the admission controls suppress warmups, issuing no additional inference requests.

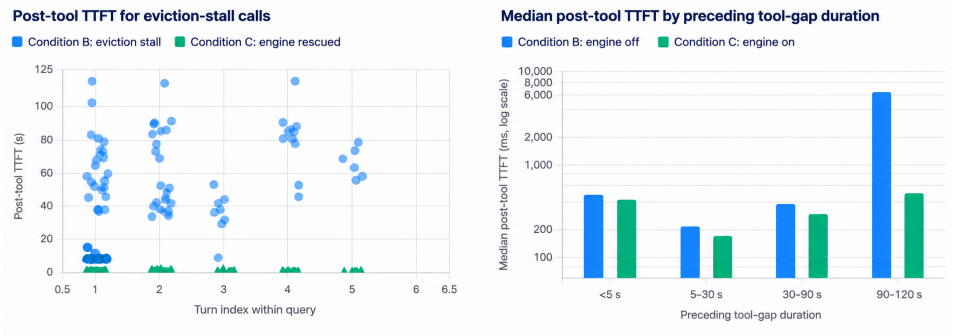


Figure 3: **Claude Code on GAIA-100 under concurrent cache pressure. Left:** compares post-tool TTFT with native prefix caching (Condition B) and the prefetch engine (Condition C); **Right:** median TTFT by preceding tool-gap duration, showing the largest benefit after long gaps.

Table 1: **Trace-replay serving results across harnesses.** Claude Code, Hermes, and Codex compare organic-cache baseline against engine-enabled execution; the multi-agent deep research workflow row reports the dedicated subagent cold-start ablation.

Harness	Benchmark	Noise N	Baseline TTFT	Engine TTFT	TTFT reduction
Claude Code	GAIA-100	16	11,527 s	3,735 s	67.6% ↓
Hermes	GAIA-250	12	1,831 s	1,471 s	19.6% ↓
Multi-agent Deep Research Workflow	Deep-research tasks	1	564 ms	177 ms	68.6% ↓
Codex	SWE-bench Lite	8	653 s	203 s	68.9% ↓

4.2 Hermes on GAIA

We next evaluate Hermes, whose mix of file reads, web searches, and code execution produces tool gaps with widely varying durations. Across 250 GAIA queries, the engine reduces cumulative TTFT by 19.6%, from 1,831s to 1,471s, saving 360s of cumulative TTFT.

Individual file-read and web-search calls have median and p95 gaps of 824ms and 12s, respectively (Table 2), while total tool-wait time per mixed-tool workflow reaches 118s at p95. The aggregate result therefore includes many short gaps in which the engine is expected to remain inactive.

Stratification confirms this opportunity-dependent behavior. On the 167 model calls following gaps of 90–120 s, the engine reduces TTFT by 87% and raises the mean KV-hit fraction from 0.358 to 0.981. Among calls that experience eviction without the engine, 53% meet the fully-rescued criterion. For gaps shorter than one second, the completion-time gate in Section 3.1 suppresses warmups whose estimated prefill time exceeds 70% of the predicted gap.

Across all queries, the TTFT gain-to-loss ratio is 9.4×: for every millisecond added by an overlapping warmup, the engine saves 9.4 ms elsewhere. 50.4% of sessions show a clear benefit, 45.0% are marginal, and 4.6% show a small regression. Regressions occur when a tool finishes earlier than predicted, causing its warmup to overlap briefly with the foreground request.

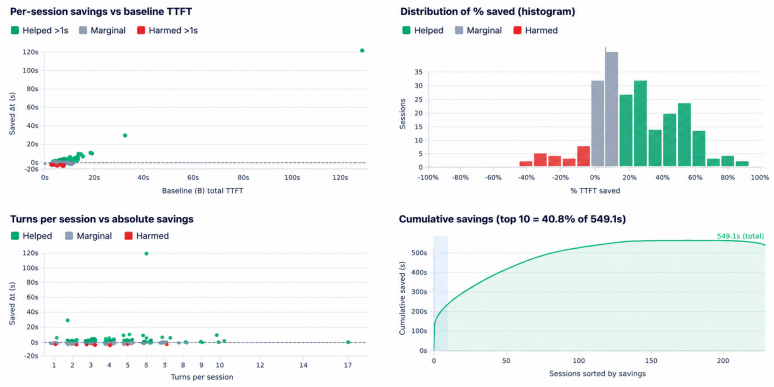


Figure 4: **Hermes on GAIA-250 under concurrent cache pressure.** Per-session results show that savings increase with available tool-gap time, while regressions occur primarily when unexpectedly short tool executions cause warmups to overlap with foreground requests.

4.3 Codex on SWE-bench Lite

We evaluate Codex on SWE-bench Lite (Jimenez et al., 2024). In an experiment replaying 128 SWE-bench Lite Codex sessions with $N{=}8$ concurrent NoiseRunner sessions, we compare Condition B against a bandit-enabled variant of Condition C that uses a shared, online Thompson Sampling policy. This configuration reduces aggregate TTFT by 68.9%, from 653.09 s to 203.01 s. Across these 128 sessions, the median per-session normalized post-tool TTFT is $C/B{=}0.266$, with all 128 sessions below $C/B{=}1$. A separate B/C experiment over 86 SWE-bench Lite sessions from Astropy, Django, scikit-learn, and SymPy reduces aggregate TTFT by 38.2%, from 887 s to 548 s.

The 86-session experiment additionally shows how benefit varies with baseline latency: the median per-session C/B TTFT ratio is 0.841 across the 32 sessions with a Condition-B median TTFT of at least $600\,\mathrm{ms}$, compared with 0.997 across the remaining 54 faster-baseline sessions. Figure 5 shows three additional trends: eviction rescue reduces cold-start TTFTs of 200–$1{,}800\,\mathrm{ms}$ to warm-cache latency; TTFT savings scale linearly with tool-call count ($R^2 > 0.95$); and the 20 highest-benefit sessions account for 50% of the $450\,\mathrm{s}$ saved.

PASTE isolation. We separately isolate PASTE's contribution on tool-output calls in a live Codex session. PASTE warms the post-tool prefix using the actual returned Bash output, targeting tokens that are unavailable before tool completion. Enabling PASTE raises the KV-hit fraction from 0.485 to 0.993, a 50.8 percentage-point increase corresponding to an additional 1,024 cached Bash-output tokens per model call.

Bandit ablation. The Thompson Sampling policy (Thompson, 1933) achieves a mean normalized TTFT of 0.282, compared with 0.286 for an imitation-learned heuristic policy trained on recorded decisions, a further 1.44% per-instance improvement. Cumulative normalized TTFT converges near 0.298 as the learned skip rate reaches 52.8%, indicating that the policy increasingly suppresses low-value warmups.

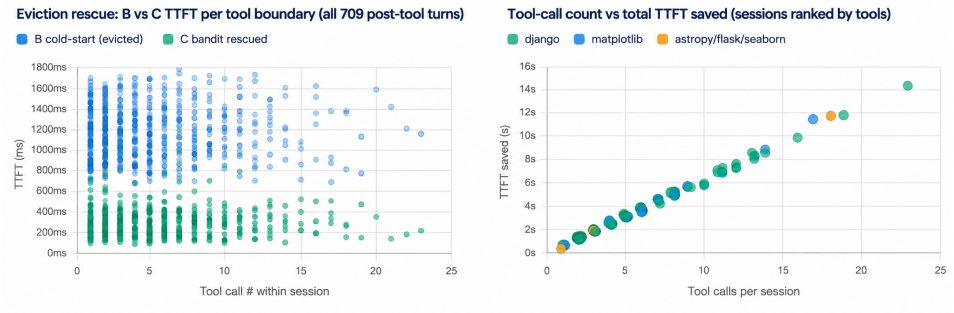


Figure 5: **Codex on SWE-bench Lite across 128 sessions.** The engine rescues eviction-affected post-tool calls, and total TTFT savings increase with the number of tool calls in a session.

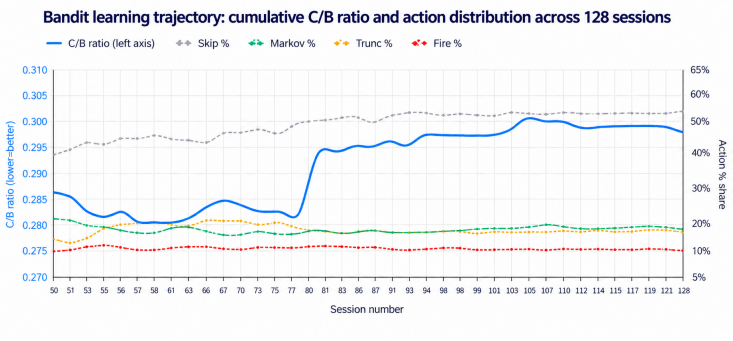


Figure 6: **Bandit learning trajectory across 128 sessions.** Early sessions explore firing more heavily; as learning progresses, the policy increasingly favors Skip and Markov actions, with the skip rate rising above 50% as the cumulative C/B TTFT ratio stabilizes near 0.298.

4.4 Subagent Cold Starts

We evaluate Mode 2 using a multi-agent deep research workflow, in which an orchestrator spawns researcher subagents configured to begin without warm KV state. Mode 2 raises the median researcher-transition KV-hit fraction from 0.006 to 0.988 and reduces median TTFT from $564\mathrm{ms}$ to $177\mathrm{ms}$, a 68.6% reduction. Appendix B provides the full results.

5 System Analysis

5.1 Overhead

To quantify the additional serving work introduced by warmups, we measure token overhead at $N \in \{1, 4, 8\}$ parallel sessions. We define token overhead as the number of prompt tokens processed by warmup requests divided by the number processed by foreground requests. In this overhead experiment, warmup requests added 7,232 prompt tokens at $N{=}1$, corresponding to approximately 2.0% of the 367K tokens processed by foreground requests. At $N{=}4$, warmups process 11,126 tokens compared with 1.47M foreground tokens (0.8%); at $N{=}8$, they process 14,837 tokens compared with 2.94M foreground tokens (0.5%).

5.2 Multi-query sessions

Figure 7 follows four consecutive tasks on the same worker. Mode 2 provides a $13.7\times$ first-call TTFT speedup by warming the initial subagent context. Registry-based worker affinity provides a further $1.6\times$ speedup on queries 2–4, while PASTE reduces tool-result TTFT spikes from 343–916ms to 38–46ms. Non-tool turns are unchanged, indicating no measurable TTFT overhead when native prefix caching is already effective.

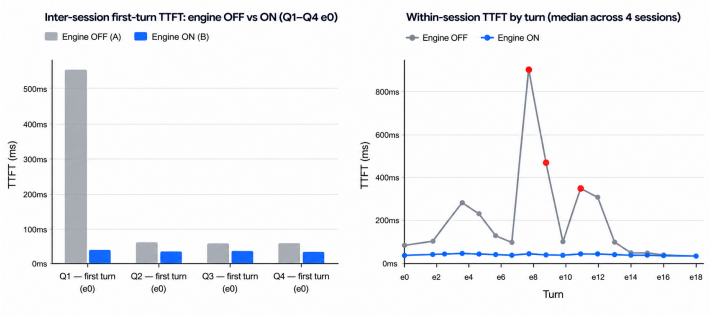


Figure 7: **Engine behavior across four consecutive agent tasks on the same worker. Left:** first-turn (e0) TTFT for each task; Mode 2 provides the initial warmup, while worker affinity preserves reusable KV state across later tasks. **Right:** median within-session TTFT across the four tasks; PASTE reduces post-tool TTFT spikes from 343–916 ms to 38–46 ms. Red markers indicate tool-result turns.

5.3 Latency predictability

Across the paired calls in Figure 11, TTFT without the engine ranges from 39 to 916ms, a $23\times$ spread caused by cache eviction. Within this paired-call experiment, with the engine, TTFT remains between 33 and 67ms, a $2\times$ spread under the same load. This reduction in tail variability limits isolated latency spikes that can violate request-level SLOs.

5.4 Cross-worker routing

A warmup is ineffective if the foreground request is routed to a different worker. We therefore evaluate KV-aware routing separately from the explicit worker affinity used in the replay experiments. With KV events published to Dynamo's Flash Indexer, KV-aware routing selects the worker holding the matching warmed prefix for 100% of requests in this experiment, achieving a 0.648 KV-hit fraction and 258ms p50 TTFT ($1.61\times$ speedup). The full engine reaches a 0.767 KV-hit fraction and 168ms p50 TTFT ($2.47\times$). Under saturation, warming an idle worker instead reduces p50 TTFT from 370ms to 277ms at the same 0.648 hit fraction (Figure 12, Appendix A).

6 Discussion and Limitations

Deployment scope and portability. The core design requires harness-visible lifecycle events, prefix-caching inference requests, and a mechanism for routing a foreground request to the worker

holding the warmed prefix. The current implementation additionally uses Dynamo-specific priority hints and KV-cache telemetry. Other serving stacks require adapters for these interfaces; evaluating an SGLang backend (Zheng et al., 2024), whose cache organization and scheduling differ from vLLM, is a natural extension.

Capacity and scheduling tradeoffs. Warmups share inference capacity with foreground requests, so their benefit depends on completing within otherwise available tool-gap capacity. Unlike TTL retention, which reserves existing KV state, Mode 1 can rewarm an evicted prefix during the tool gap, trading prefill work for cache capacity. The completion-time gate, concurrency limits, deduplication, and learned skip policy reduce late or redundant warmups, but background priority hints do not provide strict performance isolation. Consequently, gains may diminish as the deployment approaches saturation. Characterizing this boundary under broader multi-tenant workloads remains future work.

Evaluation scope. Our experiments use a model on a single 8-H100 NVL node and do not establish performance against prior cache-management systems or across other models, hardware, or multi-node deployments. Multi-node systems may introduce additional routing and transfer costs when the warmed KV state and foreground request are placed on different workers. The current mechanism also targets reusable attention KV state; hybrid architectures containing recurrent or state-space components (Gu and Dao, 2023) may require additional mechanisms for warming or transferring non-KV state.

Extensions. Three extensions follow directly from the current design. First, additional serving adapters can broaden backend coverage while preserving the harness-level policy. Second, pooling tool-duration and action outcomes across sessions can provide stronger priors for a new session's first scheduling decision. Third, combining predictive warmup with compressed or quantized KV storage could increase effective cache capacity and extend its useful concurrency range.

7 Related Work

Continuum (Li et al., 2025) retains KV state across tool calls using adaptive TTLs, while InferCept (Abhyankar et al., 2024) chooses among preserving, swapping, and discarding interrupted contexts. TokenCake (Bian et al., 2025) combines agent-aware scheduling with proactive KV offload and upload. These systems manage existing KV state but do not target the unseen tokens returned by tools. KVFlow (Pan et al., 2025) uses an Agent Step Graph to guide eviction and CPU-to-GPU prefetching, whereas PBKV (Zheng et al., 2026) uses multi-step invocation predictions for server-side cache management.

Act While Thinking (Sui et al., 2026) speculatively executes predicted tools, while IdleSpec (Choi et al., 2026) generates speculative plans during tool gaps; both target work other than KV-cache preparation. TokenDance (Bian et al., 2026) shares KV state across synchronized multi-agent rounds, while PEEK (Xie et al., 2026) exploits prefix-sharing clusters already present in the serving queue. Our engine acts before the corresponding foreground request is queued. AutoTool's finding that tool transitions exhibit low-entropy structure (Jia and Li, 2025) motivates our next-tool predictor. The resulting warmups reuse the prefix-caching mechanisms provided by vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024).

8 Conclusion

We presented a harness-layer KV-cache prefetch engine that uses tool-lifecycle events, returned results, and subagent spawns to warm anticipated prefixes without modifying serving-layer source code. Across controlled replays of benchmark-generated Claude Code, Hermes, and Codex traces, the engine reduces aggregate TTFT by up to 68.9%; for Hermes calls following 90–120 s tool gaps, it reduces TTFT by 87%. A separate overhead experiment measures only 0.5% additional prompt tokens at $N{=}8$.

These results show that tool execution gaps provide useful warmup windows and that the harness exposes the timing and context needed to use them. As agent contexts grow across turns, harness-triggered warmup can avoid increasingly expensive cold prefills while adapting to foreground load.

AI Use Statement

In this work, we used Anthropic Claude and OpenAI Codex to assist with drafting and editing prose; reviewing technical consistency and related literature; and implementing and editing code for the prefetch engine and harness adapters. The tools also assisted in organizing and presenting author-provided experimental results, but did not execute experiments or generate or alter measurements. The authors made all final decisions regarding the research methodology, experimental design, and interpretation of results. All text was reviewed and verified, and all code was reviewed and tested by the authors. We take responsibility for the final content of this work, including all text, claims, and code.

Reproducibility Statement

Dynamo, the serving framework used in our experiments, and NeMo Relay, the runtime integration layer, are publicly available under the Apache 2.0 license. Appendix B documents the hardware and model configuration, benchmark workloads, trace capture and replay procedure, cache conditions, NoiseRunner configuration, and cache-state validation protocol. Benchmark tasks are first executed through their corresponding harnesses, after which the recorded on-wire request sequences are replayed under the three evaluated cache conditions. The traces contain benchmark-generated requests rather than end-user or customer traffic. Section 3.1 specifies the Thompson Sampling contexts, actions, and feedback signals used by the engine.

References

H. Li, Q. Mang, R. He, Q. Zhang, H. Mao, X. Chen, A. Cheung, J. Gonzalez, and I. Stoica. *Continuum: Efficient and Robust Multi-Turn LLM Agent Scheduling with KV Cache Time-to-Live.* arXiv:2511.02230, 2025. A workshop version appeared at the Lifelong Agent Workshop at ICLR 2026.

H. Zheng, F. Fu, J. Wu, B. Yuan, Y. Zhang, H. Wang, Y. Zhu, X. Yan, and J. Jiang. *Efficient Serving for Dynamic Agent Workflows with Prediction-based KV-Cache Management.* arXiv:2605.06472, 2026.

W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C.H. Yu, J. Gonzalez, H. Zhang, and I. Stoica. *Efficient Memory Management for Large Language Model Serving with PagedAttention.* SOSP, 2023.

L. Zheng, L. Yin, Z. Xie, C. Sun, J. Huang, C.H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J.E. Gonzalez, C. Barrett, and Y. Sheng. *SGLang: Efficient Execution of Structured Language Model Programs.* NeurIPS, 2024.

Z. Pan, A. Patel, Z. Hu, Y. Shen, Y. Guan, W.-L. Li, L. Qin, Y. Wang, and Y. Ding. *KVFlow: Efficient Prefix Caching for Accelerating LLM-Based Multi-Agent Workflows.* NeurIPS, 2025. https://arxiv.org/abs/2507.07400

Z. Bian, F. Wu, T. Ma, and Y. Zhuo. *TokenCake: A KV-Cache-centric Serving Framework for LLM-based Multi-Agent Applications.* arXiv:2510.18586, 2025.

NVIDIA. *NeMo Relay: Multi-Language Agent Runtime.* https://github.com/NVIDIA/NeMo-Relay, 2026.

S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. *ReAct: Synergizing Reasoning and Acting in Language Models.* ICLR, 2023.

Z. Bian, F. Wu, C. Zhang, H. Dong, Y. Liang, and Y. Zhuo. *TokenDance: Scaling Multi-Agent LLM Serving via Collective KV Cache Sharing.* arXiv:2604.03143, 2026.

J. Jia and Q. Li. *AutoTool: Efficient Tool Selection for Large Language Model Agents.* arXiv:2511.14650, 2025.

G. Mialon, C. Fourrier, C. Swift, T. Wolf, Y. LeCun, and T. Scialom. *GAIA: A Benchmark for General AI Assistants.* ICLR, 2024. https://huggingface.co/datasets/gaia-benchmark/GAIA

C.E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan. *SWE-bench: Can Language Models Resolve Real-world GitHub Issues?* ICLR, 2024. https://www.swebench.com

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, and I. Polosukhin. *Attention Is All You Need.* NeurIPS, 2017.

R. Abhyankar, Z. He, V. Srivatsa, H. Zhang, and Y. Zhang. *InferCept: Efficient Intercept Support for Augmented Large Language Model Inference.* Proceedings of the 41st International Conference on Machine Learning (ICML), PMLR 235:81–95, 2024.

Y. Sui, H. Zhao, R. Ma, Z. He, H. Wang, J. Li, and Y. Yang. *Act While Thinking: Accelerating LLM Agents via Pattern-Aware Speculative Tool Execution.* arXiv:2603.18897, 2026.

D. Choi, K. Park, W. Song, S. Dingliwal, S.M. Jayanthi, J. Shin, and A. Galstyan. *IdleSpec: Exploiting Idle Time via Speculative Planning for LLM Agents.* arXiv:2605.22154, 2026.

B. Xie, Z. Wang, M. Tanaka, and Z. Zhen. *PEEK: Predictive Queue-Informed KV Cache Management for LLM Serving.* arXiv:2607.02525, 2026.

W.R. Thompson. *On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples.* Biometrika, 25(3–4):285–294, 1933.

Model Context Protocol. *Schema Reference: ToolAnnotations.* Model Context Protocol Specification, 2025-06-18. https://modelcontextprotocol.io/specification/2025-06-18/schema

A. Gu and T. Dao. *Mamba: Linear-Time Sequence Modeling with Selective State Spaces.* arXiv:2312.00752, 2023.

A Additional Figures

Harness	Tool type	p50 gap	p95 gap	p50 stable tokens
Claude Code	Bash	8.6 s	45 s	∼38,500
Hermes	read/web search	824 ms	12 s	∼28,000
Multi-Agent Deep Research Workflow	web search	1.9 s	8 s	4,835
Codex	Bash/file ops	300 ms	12 s	∼12,000
Minimal ReAct	*single tool*	—	—	*452*

Table 2: Tool-gap durations and stable prefix lengths.



Figure 8: **Full-prefix KV-cache reuse substantially reduces time to first token.** For Qwen3-0.6B with a 7,320-token prefix, TTFT decreases from 403 ms under cold prefill to 129 ms with a full-prefix cache hit (68%). For Llama-3.1-Nemotron-70B with a 6,723-token prefix, TTFT decreases from 7,649 ms to 393 ms (95%). This corresponds to 7,320 and 6,723 tokens saved from prefill recomputation, respectively. Bars are normalized independently to each model's cold-prefill TTFT; labels report absolute latency. Measurements exclude one-time model loading and compilation and use the same initialized model and generation settings. Experiments run on 8×H100 NVL 94 GB GPUs with Dynamo and vLLM, using TP=1 for Qwen3-0.6B and TP=4 for Nemotron-70B.

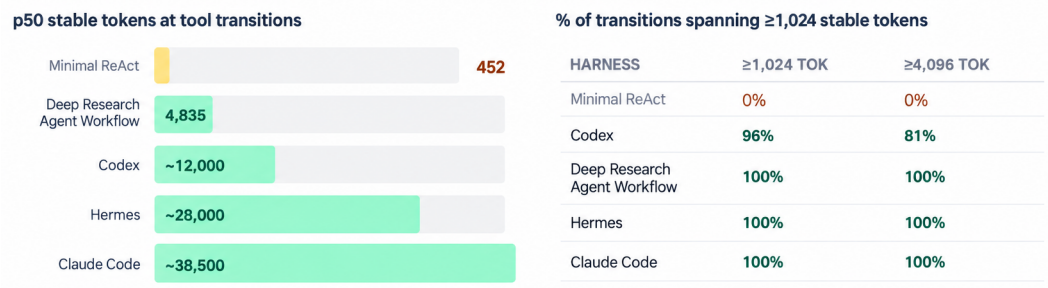


Figure 9: **Stable prefix lengths across harnesses. Left:** median shared-prefix length between consecutive LLM calls at tool transitions. **Right:** percentage of transitions retaining at least 1,024 or 4,096 stable tokens. The SGLang radix-cache block size is 1,024 tokens, so a warmup must span at least one complete block to enable reuse on the subsequent call. The production harnesses have median stable prefixes of at least 4,835 tokens and retain at least 1,024 tokens on 96–100% of transitions, whereas the minimal ReAct configuration crosses neither threshold.

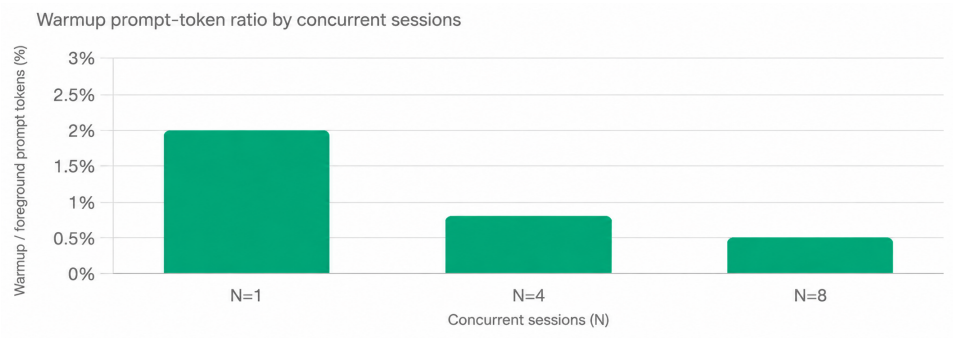


Figure 10: **Prompt-token overhead of warmup requests drops with concurrency.** Each bar shows warmup prompt tokens as a percentage of foreground prompt tokens for the indicated number of sessions. The ratio decreases across the tested conditions

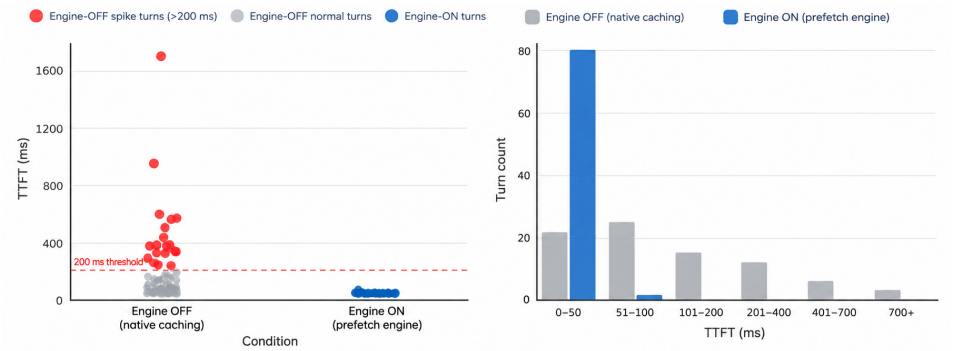


Figure 11: **TTFT variance with and without the prefetch engine. Left:** each point represents one agent turn; engine-off turns above the 200 ms analysis cutoff are highlighted in red. Without prefetching, TTFT exhibits frequent high-latency spikes, whereas engine-on turns remain below 100 ms. **Right:** binned TTFT distributions show that native caching produces a broad latency tail extending beyond 700 ms, while the prefetch engine concentrates nearly all turns in the 0–50 ms bin. Result-aware warming of tool-output content therefore suppresses latency spikes and makes per-turn TTFT lower and more consistent.

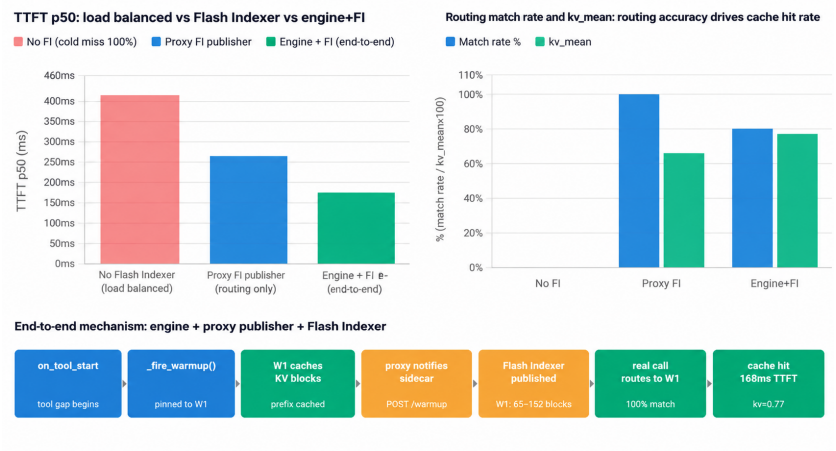


Figure 12: **Cross-worker routing and load-aware warmup dispatch. Top left:** median TTFT decreases from approximately 410 ms with ordinary load-balanced routing to 263 ms with the proxy Flash Indexer publisher and approximately 170 ms with the complete prefetch-engine-plus-Flash-Indexer path. **Top right:** routing match rate and mean KV-cache hit rate ($\mathrm{kv_mean}$) for the Flash Indexer configurations. **Bottom:** at tool start, the engine issues a warmup pinned to worker W1, the proxy publishes the cached blocks to the Flash Indexer, and the subsequent request is routed to W1, producing a cache hit and 168 ms TTFT.

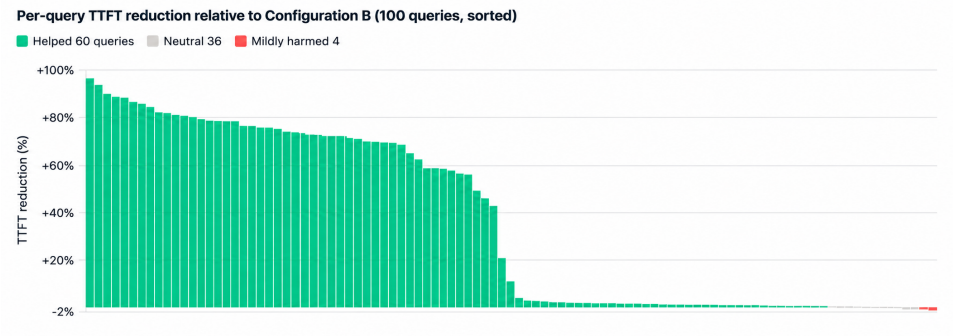


Figure 13: **Claude Code GAIA-100: per-query TTFT savings.** Queries are sorted by percentage of TTFT saved across the 100-query evaluation. The engine reduces TTFT for 60 queries, has a neutral effect for 36, and mildly increases TTFT for 4. The magnitude of improvement varies substantially across queries, with the largest savings exceeding 90%, while neutral and harmed queries remain close to zero.

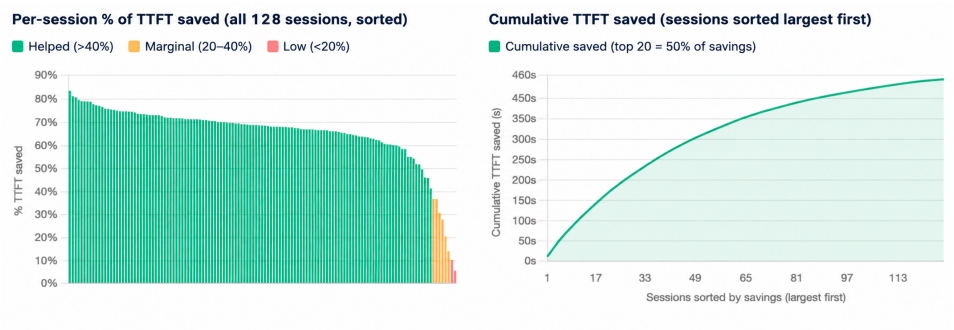


Figure 14: **Distribution and concentration of TTFT savings across 128 Codex sessions. Left:** percentage of each session's TTFT saved relative to native caching, sorted by benefit. **Right:** cumulative absolute TTFT saved as sessions are added in the same order; the first 20 sessions account for approximately half of the total savings.

B Experimental Setup and Additional Results

Hardware and experimental protocol

The main evaluation experiments run on a single node with eight H100 NVL GPUs, each with 94 GB of HBM3 memory. We serve Llama-3.1-Nemotron-70B through Dynamo with a vLLM backend, tensor parallelism of four, and prefix caching enabled. Two four-GPU workers share the node: W1 (GPUs 0–3) serves replay traffic, while W2 (GPUs 4–7) is used for trace collection and the cross-worker experiments. The measured cold-prefill latency increases by approximately 37 ms per 1,000 input tokens, and cached requests have a median warm TTFT of approximately 110 ms in the full-workflow experiment.

We use content-preserving trace replay. Benchmark tasks are first executed through their corresponding agent harnesses, and the resulting on-wire request contents and ordering are recorded. The recorded requests are then replayed on W1 under the selected cache condition. This preserves the benchmark-generated prompts and tool results while allowing cache behavior to be controlled independently. The traces contain no end-user or customer traffic.

Before each trial, we flush KV blocks on both workers and use a canary request to confirm a near-zero KV-hit fraction; trials with residual cache state are discarded. NoiseRunner generates staggered sessions with fixed prefixes to induce cache pressure and eviction during tool gaps. We use $N{=}16$ sessions with 40,000-token prefixes for Claude Code on GAIA, $N{=}8$ sessions with 10,000-token prefixes for Codex on SWE-bench Lite, and $N{=}12$ sessions for Hermes on GAIA.

Multi-Agent Deep Research Workflow Example

We evaluate Mode 2 on deep-research tasks using a multi-agent workflow in which an orchestrator decomposes each query and spawns researcher subagents for individual subtasks. Each researcher performs web searches, retrieves documents, and synthesizes its findings in a separate LLM session. In our cold-start condition, the researcher is assigned to a worker without its relevant KV state, isolating the subagent miss described in Section 2.3. On the first invocation of an unseen subagent type, the engine captures and persists its initial on-wire prefix. On subsequent spawns, Mode 2 uses the captured prefix to warm the subagent's reusable initial context before its first model call.

Mode 2 raises the first-call KV-hit fraction on warmed researcher spawns from 0.006 to 0.988, reducing TTFT by approximately 387 ms. Across the validated spawn warmups, Mode 2 caches 51,532 of 57,932 eligible prefix tokens (89%). Separately, the fraction of first calls classified as cold prefills falls from 47.1% to 11.6%, a relative reduction of 75.3%.

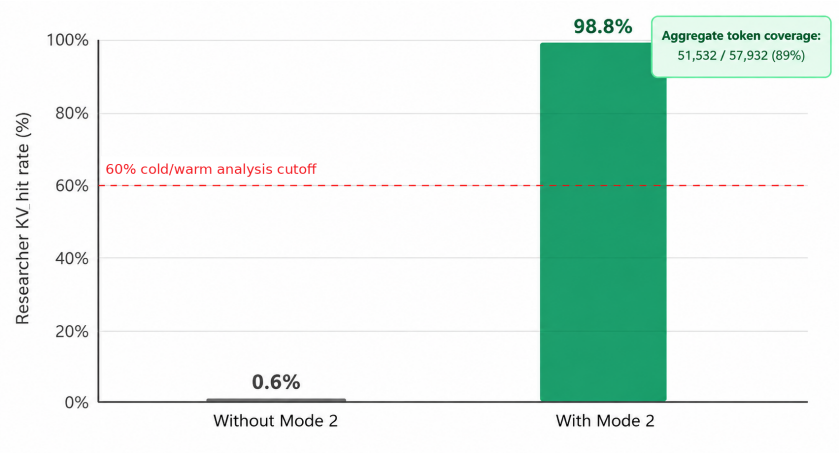


Figure 15: **Multi-agent Deep Research Workflow Mode 2 subagent spawn warmup results.** Median researcher-transition KV-hit fraction rises from 0.006 without Mode 2 to 0.988 with Mode 2 in the deep-research workflow. Across validated warmups, 51,532 of 57,932 eligible prefix tokens are covered (89%).

PREFETCH ENGINE INTERNAL ARCHITECTURE

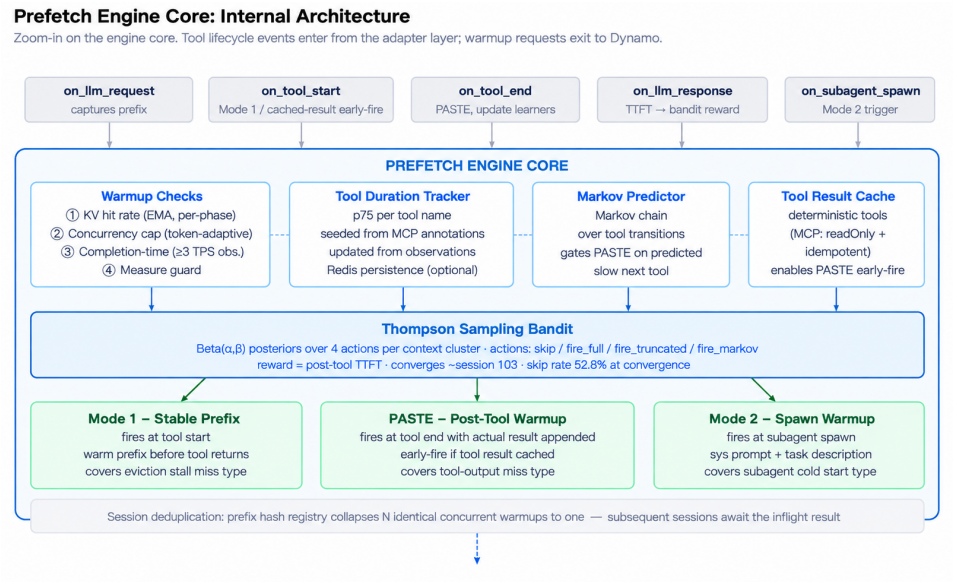


Figure 16: **Prefetch engine internal architecture.** Harness lifecycle events enter the shared engine through adapters. Cache checks, duration estimates, tool-result reuse, and action selection govern three warmup paths: stable-prefix warming, post-tool PASTE, and subagent-spawn warming. An in-flight prefix registry coalesces duplicate requests before dispatch.