

A Hit Rate Is a Retention Policy: Prefix-Cache Reuse in Agentic Coding Traces Compounds the Workload With the Server's Eviction Rule and the Harness's Invalidations, Which Serving Assumptions the Traces Actually Break, and the Rent-or-Buy Rule That Prices the Turn Boundary

Pranay Mahendrakar

SONYTECH · ORCID 0009-0003-7224-029X

pranaymahendrakar@sonytech.in · mahendrakarpranay@gmail.com · +91 6361723454

Abstract. Coding agents are becoming a major class of language-model traffic, and the first production-scale characterisation of that traffic, 13.5 million GitHub Copilot agent sessions, concludes that its findings "challenge assumptions underlying current LLM-serving systems and provide an empirical foundation for agent-native infrastructure." Its central evidence is the prefix-cache hit rate: about 90 percent within an agent turn, far lower across turn boundaries, 8 percent after a model switch. Other 2026 studies of agent traffic report hit rates from above 98 percent to below 17 percent. In this analysis the author reviews and synthesises published results to show that a measured hit rate is the product of three factors that the studies vary separately: the reuse the workload makes available, the probability that cached state survives until it is reused, which is set by the server's retention rule, capacity and load, and the probability that the agent harness has not invalidated the prefix by switching models or compacting context. Holding the workload and model fixed, one study's hit rate falls from 98.85 to 16.78 percent when concurrency rises from 15 to 20 agents; in another, sweeping the eviction timeout from one minute to one hour moves the achievable rate from 85.4 to 98.6 percent. The paper audits which serving assumptions the traces break, finds that statelessness was abandoned for multi-turn chat by 2024 and that context rewriting which invalidates the prefix predates agents, while the size of the state at stake, model switches made by automatic routing and LLM-written compaction are new, and separates three senses in which agent serving is said to be dominated by input, by decode or by cache charges. It states the turn-boundary retention decision as a rent-or-buy problem with a hazard-rate threshold, works a keep-alive break-even from published prices, and specifies the policy-held-fixed comparisons that would settle what is specific to agents.

Keywords: LLM serving; KV cache; prefix caching; coding agents; workload characterization; cache retention; ski rental; context compaction

SONYTECH ORCID: 0009-0003-7224-029X

1. Introduction

Language-model serving systems were designed around a request that arrives, is processed, and leaves. Continuous batching, PagedAttention (Kwon et al., 2023) and the chunked-prefill schedulers that followed (Agrawal et al., 2024) all assume that the unit of work is a request whose prompt must be prefilled and whose output must be decoded, and they optimise the interleaving of many such requests on a GPU. Prefix caching was added to that design early: SGLang's RadixAttention reuses the key-value (KV) state of previously processed prompts when a new prompt shares a prefix with them (Zheng et al., 2023), and vLLM's paged allocator supports sharing KV state within and across requests (Kwon et al., 2023). Under this design the server does not know that two requests belong to one conversation; it only notices, at admission time, that the new prompt begins with tokens it has already processed and still holds.

Coding agents strain that design in an obvious way. A coding agent such as GitHub Copilot's, Claude Code or Codex answers a user prompt by running a loop: it calls

the model, executes the tool call the model emits, appends the tool output to the context, and calls the model again, often dozens of times, before returning control (Yao et al., 2022; Yang et al., 2024). Each call resends the whole accumulated context. What its authors describe as the first production-scale characterisation of this workload, built from sampled GitHub Copilot telemetry covering 13.5 million sessions, 3.2 million users and 760.5 million model calls in one week of June 2026, reports that the median call carries 68K prompt tokens and emits 247 output tokens, that 87 percent of model calls are initiated by the agent rather than the user, and that the prefix cache serves most of the prompt within a turn but loses much of it at the boundary between turns, when the user is reading, thinking or away (Liu et al., 2026). Its authors conclude that the findings challenge the assumptions of current serving systems and motivate agent-native infrastructure.

Other characterisations of agent traffic published in 2026 report what look like incompatible facts about the same quantity. A trace of about 4,300 Claude Code and Codex sessions finds a token-weighted prefix hit rate of 95.7 percent overall (Zhu et al., 2026). A controlled study of

ReAct-style agents on five benchmarks reports hit rates of 84.6 to 99.5 percent and finds model time dominated by decode rather than prefill (Yuan et al., 2026). A study of an agentic platform at a large cloud provider reports that KV hit rates "collapse despite substantial context reuse" (Biswas et al., 2026), and a study of production traces from an agent-serving service reports that more than 40 percent of requests receive no or very limited cache hits, calling this "a starkly contradictory reality" relative to the assumption that agents naturally achieve high reuse (Yu et al., 2026). If the hit rate were a property of agentic workloads, these numbers could not all be right.

In this analysis the author reviews and synthesises published results to show that they can all be right, because a prefix-cache hit rate is not a property of a workload. It is the product of at least three factors. The first is reuse potential: the fraction of each prompt that repeats something the server has already processed, which the workload and the harness determine together. The second is survival: the probability that the cached state for that repeated prefix is still held when the next call arrives, which the server's retention rule, its memory capacity and the competing load determine. The third is validity: the probability that the prefix has not been rewritten or orphaned before reuse, which the harness determines when it switches models, compacts the context or edits the system prompt. The 2026 characterisations differ chiefly in which of these three they hold fixed, and the conflicts among them line up with that difference. Agentic coding has unusually high reuse potential; whether that potential becomes a hit is decided by policy and by the harness.

The decomposition changes what the traces can be said to show about serving assumptions. Some of the assumptions the Copilot study names, statelessness across requests above all, were abandoned for multi-turn chat by systems published in 2023 and 2024 (Gao et al., 2024; Yu et al., 2025; Qin et al., 2024). Some are being abandoned in 2025-2026 agent-serving systems, which treat programs rather than requests as the unit of scheduling and retention (Luo et al., 2025; Li et al., 2025; Guo et al., 2026). What the traces add is the magnitude of the state at stake when a retention decision is wrong, and invalidations by model switching and LLM-written compaction that originate in the agent harness rather than in the server. The paper argues that the turn boundary the Copilot study highlights is best understood as a rent-or-buy problem of the kind competitive analysis has studied since Karlin et al. (1988), and that the idle-time predictor the study proposes should be judged by the expected cost of the retention decisions it drives, not by the fraction of idle time it flags.

None of this diminishes the characterisation. Its authors describe it as the first production-scale study of coding-agent traffic, and several of its observations are

corroborated elsewhere: the separation of second-scale gaps within a turn from minute-scale gaps between turns appears in an independent trace (Zhu et al., 2026), and the longer, more variable trajectories of failing agents appear in benchmark studies (Majgaonkar et al., 2025). The claim here is narrower: the headline cache numbers in agent traces describe a workload running under a particular stack, and a statement about which serving assumptions they break must say which factor moved.

Terms are defined in Section 2, and Section 3 sets the Copilot claim beside the two ways of reading it. The three-factor identity is introduced in Section 4; reuse potential, survival and validity are then examined one per section (5, 6 and 7). Section 8 separates three senses of "dominated" that the literature uses for agent serving. Section 9 audits the serving assumptions the traces are said to break. Section 10 states the turn-boundary decision as a rent-or-buy problem, gives it as an algorithm, works a keep-alive example from published prices, and reassesses the predictor's evaluation. Section 11 gathers the measured values into two tables and two figures. Section 12 proposes five comparisons, each isolating one factor. The paper closes with the objections it takes most seriously (Section 13), the quantities nobody has published (Section 14), and the limits of its own synthesis (Section 15).

1.1 Relation to the author's earlier papers

This paper sits next to two of the author's published papers. One of them argued that "budget awareness" in language-model agents names several distinct decisions, of which allocation gains were keyed to signals measured after the fact (Mahendrakar, 2026a); that paper concerned the agent's own spending decisions, while this one concerns the serving system's decision about state the agent will need. The other argued that compression results obtained when the query is visible do not transfer to agent memory writes, where the compressor must act before any question exists (Mahendrakar, 2026b). Context compaction in coding agents, discussed in Section 7, is such a write, and the present paper adds the observation that compaction also destroys the reusable prefix, so its cost has a serving term that the memory-quality literature does not price. The object and the move here are different from both: the object is the prefix-cache hit rate reported by workload characterisations, and the move is to factor it into terms that a comparison can hold fixed.

2. Scope and terms

Call, turn, session. Following the Copilot study, a session is the lifetime of one agent conversation; a turn begins with a user prompt and includes every model call and tool call the agent makes before returning control; a step is one model call or one tool call (Liu et al., 2026). The first model call of a turn is user-initiated; the rest are agent-initiated. TraceLab uses the matching

distinction between user-initiated steps and tool-result steps (Zhu et al., 2026).

Prefix-cache hit rate. The fraction of prompt tokens in a call that are served from previously computed KV state rather than recomputed, weighted by tokens unless stated otherwise. API-level studies observe it as the provider's count of cached input tokens; server-side studies observe it directly. A per-call median, a token-weighted mean and an "ideal" rate under unlimited capacity are three different statistics, and the studies below report different ones; where the distinction matters the paper names the statistic.

Reuse potential (R). For a call, the fraction of its prompt tokens that form a prefix already processed at some earlier time by the same server or cache domain. It is fixed by what the agent sends: the conversation history, tool outputs, the system prompt, and whether earlier content is resent unchanged. It is the hit rate an infinite cache that never evicts and never invalidates would achieve.

Survival (S). The probability, given that a reusable prefix exists, that its KV state is still held in some reachable tier when the call arrives. It is fixed by the retention rule (least recently used, time-to-live, workload-aware), the capacity of each tier, and the volume of other state competing for that capacity.

Validity (V). The probability that the prefix the next call would reuse has not been made unusable by an event on the client side: a change of model, a rewrite of the context by compaction, an edit to the system prompt or tool list, or routing to a different cache domain.

Retention policy. The rule a server or provider applies to decide how long cached state is held and in which tier: GPU memory, host DRAM, or storage (Liu et al., 2025; Qin et al., 2024). A time-to-live (TTL) policy holds state for a fixed time after last use; an LRU policy evicts the least recently used entries when space is needed.

Harness. The client-side orchestration that assembles the context for each call, exposes tools, decides when to compact, and chooses or switches the model (Rombaut, 2026; Ali et al., 2026).

Scope. The paper concerns decoder-only transformer serving with exact-prefix KV reuse. Non-prefix reuse (Yao et al., 2024; Gim et al., 2023) and hybrid recurrent architectures, where exact-match constraints change the reuse arithmetic (Pan et al., 2024), are mentioned where they bear on the argument and are otherwise out of scope. Tool-execution containers, which the Copilot study also analyses, are discussed only where their idle structure informs the KV decision.

3. The claim at issue and its two readings

3.1 The Copilot characterisation

The study's table of key findings lists, among others: that coding agents differ from chat, with 87 percent of calls agent-initiated and a roughly one-to-one ratio of model calls to tool calls; that the cache is session-structured, about 90 percent cached but down to 55 percent at turn boundaries; that model switches destroy the cache, with only 8 percent cached after a switch; that context compaction is costly, occurring in 7.8 percent of sessions and cold-starting the cache; and that turn boundaries signal idle periods of 4.1 minutes for containers and 2.9 minutes for KV state (Liu et al., 2026). In its body, the per-call hit rate has a median of 98 percent, rising within a turn from about 45 percent on the first call to 86 percent on the second and 92 to 94 percent from the third onward. Same-model turn boundaries reduce the hit rate by 26 points on average and model-switch boundaries by 67 points. The study's own comparison table describes chat and completion workloads as having "Low (independent)" cache sensitivity, against "High (prefix-sharing within loop)" for coding agents, and it concludes that serving infrastructure designed for chat, with request-level scheduling, LRU eviction and independent batching, is structurally mismatched to coding agents.

Two of these numbers are presented in a way that matters for what follows. The abstract's 55 percent across turn boundaries and the body's average 26-point drop at same-model boundaries are not reconcilable from the text alone if the within-turn rate is 90 percent; the author's reading is that the two are computed over different sets of boundaries, but the paper does not say. And the idle gap at a turn boundary is reported under two definitions: a median of 172 seconds for cross-turn intervals between consecutive model calls (the study's Table 8) and a median of 1,512 seconds, about 25 minutes, for user idle time between turns (its Figure 32b). The retention decision of Section 10 depends on which of these distributions applies, and the difference is large: a 172-second gap sits inside a five-minute TTL, while a 25-minute gap does not.

3.2 The workload reading

On the first reading, the numbers describe agentic coding. Agents resend long contexts many times, reuse is high within a turn and low across turns, and model switches and compaction destroy it. Serving systems built for chat do not anticipate this pattern, so the pattern is evidence that they must change. This is the reading the study's framing invites, and it is shared by several systems papers that cite agent workloads as the reason a new design is needed (Guo et al., 2026; Yu et al., 2026; Biswas et al., 2026).

3.3 The measurement reading

On the second reading, a hit rate measured from the client side reports what one serving stack did with one workload. The Copilot study itself supplies the strongest support for this reading. It describes the cliff in hit rate between two and ten minutes of idle time as "the signature of a time-based KV cache eviction policy at the serving system", it notes that the default retention for one provider's models is 300 seconds, and it lists the absence of a server-side view among its limitations: its telemetry contains no GPU utilisation, queue depth or memory pressure (Liu et al., 2026). The traces span more than 27 models from several families, served by stacks whose retention rules the study does not observe. On this reading the 55 percent at turn boundaries is a joint measurement of the users' idle-time distribution and the providers' TTLs, and it says little about whether a server with a different rule would see the same thing.

The two readings make different predictions. The workload reading predicts that hit rates measured on agent workloads should be similar across serving stacks and should differ systematically from chat. The measurement reading predicts that hit rates on the same agent workload should move substantially when the retention rule, the capacity or the load changes, and that the agent-chat difference should appear in reuse potential rather than in observed hit rates. Sections 5 to 7 show that the second set of predictions is borne out where the literature has tested them.

3.4 Why the distinction matters

The distinction is not pedantic, because the two readings recommend different engineering. If low cross-turn reuse is a workload property, the remedy is to predict when users return and to stage state accordingly, which is what the Copilot study proposes. If it is a policy outcome, the first remedy is to price retention correctly, and the comparison that justifies a new design must hold the policy fixed. Several 2025-2026 agent-serving systems report gains over baselines whose retention rule was not designed for agents; the gain is then partly a gain from changing the rule, and some of it may be available to a general-purpose stack that changed its rule in the same way (Section 9).

4. A decomposition of the observed hit rate

For one call, let R be the fraction of its prompt tokens that are a reusable prefix, S the probability that the reusable prefix is still cached when the call arrives, and V the probability that the prefix has not been invalidated by the harness. Under exact-prefix caching, the expected token-weighted hit rate of the call is

$$H = R \times V \times S,$$

where the order of the last two factors reflects that the server cannot hold a prefix the client has abandoned.

The identity is a definition, not a measurement; its use is that each factor responds to different interventions. R responds to what the agent sends: whether earlier context is resent unchanged, how much new content each step appends, whether dynamic content is placed before stable content. V responds to harness events: model switches, compaction, prompt edits. S responds to the retention rule, the capacity of each tier and the load.

Two refinements are needed in practice. First, survival is not binary across tiers: a prefix evicted from GPU memory may survive in host DRAM or storage, in which case the hit is real but costs a reload (Liu et al., 2025; Jin et al., 2024). The cost of a call depends on which tier served it, and an API-level hit rate does not distinguish them. Second, partial survival matters for long prefixes, because an LRU policy may evict the tail of a prefix while keeping its head; the hit rate then reports the surviving fraction. Neither refinement changes the argument that the three factors are separately controllable.

The decomposition assigns each published result to the factor its design varied. The Copilot study observes H on a production workload, with R , V and S all free, and attributes changes in H within the data to events that change V (model switches, compaction) or S (idle time under a TTL). TraceLab observes H from two providers' APIs and also counts the irreducibly fresh tokens, so it can state the gap between H and R (Zhu et al., 2026). Yuan et al. (2026) control their own server and vary concurrency with the workload fixed, which varies S alone. The cloud-provider study of KV reuse estimates R for chat and API traffic by simulating an infinite cache (Wang et al., 2025). EfficientAgent varies host-tier capacity with the workload fixed, which varies S alone (Shao et al., 2026). The comparisons the workload reading needs, agent against chat at fixed S and V , are the ones least often made.

5. Reuse potential: what the workload actually contributes

5.1 Agent prompts are mostly resent context

Every study that measures it finds that most of an agent's prompt at each step repeats the previous step's prompt. In the Copilot traces the median call carries 68K prompt tokens of which a median 63K are cached, conversation history accounts for 48 percent of input tokens and function-call messages for another 28 percent, and the system prompt for only 14 percent (Liu et al., 2026). TraceLab measures reuse potential more directly. It defines fresh tokens as each step's context growth minus the previous step's output, and finds that fresh tokens are 19.0 percent of all appended tokens across its trace; about 81 percent of what the deployed caches prefilled could in principle have been served from cache (Zhu et al., 2026). Combined with the trace's 95.7 percent hit rate, the author's arithmetic puts irreducibly fresh tokens

at under one percent of all prompt tokens, so R in that trace is about 0.99. For user-initiated steps, fresh tokens are only 1.7 percent of appended tokens for Claude and 4.5 percent for Codex: nearly everything sent at a turn boundary was already processed once. Yuan et al. (2026) report the same structure from a controlled setting. Mean input-to-output ratios across their configurations range from 53.9 to 559.8, but the mean ratio of appended tokens to output tokens is only 1.5 to 7.3, with medians often below 1.5, and they put the theoretical cache-hit ratio, which is R in the present notation, at 87.9 to 99.3 percent. EfficientAgent, replaying SWE-bench Verified agents, finds that 98.1 percent of all prompt tokens had already been processed in the same task's previous call, and that 97.3 percent of the new KV state a call writes reappears in the same task's next call (Shao et al., 2026).

R for coding agents is therefore high on every call after the first, between about 0.88 and 0.99 in the three studies that compute it, and the reason is structural: the agent loop appends rather than rewrites, so each prompt is the previous prompt plus a short suffix (Yao et al., 2022). This is the property that distinguishes agent traffic from most chat traffic.

5.2 Chat and API traffic also reuse, at lower potential

The Copilot study's description of chat cache sensitivity as low and independent does not match the production characterisation of KV reuse in chat and API traffic that this synthesis located. Analysing one week of traffic at a large cloud provider, Wang et al. (2025) find ideal hit rates, under a cache of unlimited capacity, of 62 percent for a customer-facing chatbot trace and 54 percent for a business API trace. Reuse in that data is skewed, with 10 percent of KV blocks accounting for 77 percent of reuses, and in the API trace single-turn requests, not multi-turn conversations, account for 97 percent of reuses, presumably because programmatic callers share system prompts and templates. The authors also note that their ideal rates are below the more than 80 percent reported on synthetic workloads, which is a caution about synthetic baselines in either direction. Earlier chat-serving systems were built on the premise that multi-turn conversation recomputes history: CachedAttention reports up to 87 percent lower time to first token and up to 70 percent lower end-to-end cost from retaining KV state across turns (Gao et al., 2024), and Pensieve reports 1.14 to 3.0 times the throughput of vLLM and TensorRT-LLM from the same idea (Yu et al., 2025).

The agent-chat difference is therefore a difference between ideal reuse of 0.54 to 0.62 for chat and API traffic and reuse of 0.88 to 0.99 for coding agents, not a difference between independence and dependence. The figures are computed differently, the chat figure by simulating an unlimited cache over hashed blocks and the agent figures from context growth or from prefix matching between consecutive calls, so the comparison is of magnitude, not of decimals. That is a large

difference, and it raises the value of every retention decision. It is not a difference in kind.

5.3 The harness sets R as well

Reuse potential is not purely a property of the task; the harness that assembles the prompt decides how much of it is stable. Pythia's analysis of a multi-agent coding assistant attributes its low hit rates partly to the fact that specialised agents in a workflow carry distinct system prompts, which prevents prefix sharing across agent boundaries (Yu et al., 2026). Lumer et al. (2026), comparing caching strategies on a multi-turn research agent across three providers, find that prompt caching cuts API cost by 41 to 80 percent, but that caching the full context naively can raise latency, because dynamic tool results trigger cache writes for content that is never reused; placing dynamic content at the end of the system prompt and excluding dynamic tool results from the cached block gives more consistent savings. Weinberger and Hozes (2026a) find the converse effect of a harness change: compressing tool outputs to save tokens changed the agent's trajectory, adding retrieval, diagnosis and testing steps, while leaving untouched the cache traffic that dominated the bill, so that the most aggressive compression cut delivered tool-output tokens by 38.4 percent while raising the billed cost by 6.8 percent. The same authors argue, from controlled experiments, that prompt wording, inference effort and harness policy interact, so that token and cache counts are measurements of a trajectory rather than targets (Weinberger and Hozes, 2026b). A controlled swap of the orchestration layer with six models held fixed reports 41 percent lower blended cost per task and 38 percent fewer tokens per task, and lists cache-shape discipline among the mechanisms (Ali et al., 2026).

R is thus jointly owned. A characterisation that reports a reuse figure for "agentic coding" reports it for the harnesses in its sample, and Copilot, Claude Code and Codex differ in how they assemble context: TraceLab finds that Codex shows frequent small context reductions that Claude Code does not, and that Codex moves long-running tasks into the background so that its tool gaps rarely exceed five minutes (Zhu et al., 2026).

6. Survival: the retention rule, capacity and load

6.1 The same workload under different loads

The most direct evidence that a hit rate is not a workload property comes from studies that hold the workload fixed and change only the server's conditions. Yuan et al. (2026) serve Terminal-Bench and DABStep agents on one Qwen3.6-27B instance served with tensor parallelism 2 across two H100 NVL GPUs and raise the number of concurrent agent jobs. On Terminal-Bench, the prefix hit rate is 98.85 percent at a maximum concurrency of 15 and 16.78 percent at 20. On DABStep

it is 98.53 percent at 5 and 25.17 percent at 10. Generation throughput per request falls by up to 66 percent once thrashing begins. The workload, the model and the harness are unchanged across these columns; what changes is whether the active contexts fit in the memory the server can devote to them. The authors state that the hit rates they report elsewhere were measured with concurrency controlled so that thrashing does not occur, and are close to the ideal.

EfficientAgent makes the same point with host-memory offloading (Shao et al., 2026). On one coding-agent workload, offloading KV state to host DRAM speeds up one deployment, slows another and leaves a third unchanged, even where restoring a token from host memory is several times cheaper than recomputing it. The explanation offered is that an agent's prefix survives in the host tier only if the tier can hold the reusable context of the whole agent pool between two uses, which the authors call the reuse working set; in the authors' motivating measurement, 79.2 percent of the KV chunks written to the host tier were evicted before the next call that needed them. Sizing the host tier to the estimated working set cut recomputed prefill by 93.1 percent and makespan by 38.7 percent. The model is the classical one: survival under LRU is governed by the stack distance between two uses of an item (Mattson et al., 1970), and the working set is the volume a program touches within a window (Denning, 1968). The characteristic-time approximation for LRU caches makes the same quantity explicit for independent request streams (Che et al., 2002).

Sutradhara's replay of an agentic platform's traces on vLLM illustrates survival under LRU with concurrent agents (Biswas et al., 2026). The authors attribute the collapse of hit rates they observe to workload-agnostic LRU eviction, which evicts shared prefixes while their owners wait on tools, and report that Sutradhara, whose priority-aware eviction protects shared system-prompt prefixes and the partial prefills that carry earlier tool outputs, raises the global hit rate in their evaluation from 21.8 to 44.6 percent. Two features of that study bear on how its "production" numbers should be read. Its characterisation analyses 6,000 agentic requests issued on the production platform with synthetic user profiles and data, because customer prompts are not visible, and validated against customer-facing traffic; and the evaluation replays a stratified sample of 120 high fan-out requests on a 14B model on an A100 (Biswas et al., 2026). The 21.8 percent is therefore a property of that replay under that eviction rule, not a measurement of production hit rates in the sense of the Copilot or TraceLab figures. Pythia's production analysis gives the same mechanism in words: by the time an agent is re-invoked after a long tool call, intervening traffic has mostly flushed its prefix (Yu et al., 2026).

6.2 The same workload under different timeouts

TraceLab's eviction-timeout sweep varies the retention rule with the workload fixed (Zhu et al., 2026). Under an idealised rule in which a step misses completely if its preceding idle gap exceeds the timeout and hits completely otherwise, raising the timeout from one minute to one hour lifts the achievable hit rate from 85.4 to 98.6 percent, at the price of raising the ratio of suspended to active KV storage from 0.74 to 5.07. Most of the gain is cheap: at five minutes the achievable rate is about 94 percent at a storage ratio of about 1.9, and the remaining move to one hour buys about four points for about 2.7 times more storage. Prefill amplification, the ratio of prefilled tokens to irreducibly fresh ones, is 18.9 at one minute, 7.4 at five minutes and 1.8 at one hour. Against the deployed caches of the two providers, which amplify prefill by 5.3 overall, the sweep shows the size of the gap that retention alone controls. Figure 2 plots the three published points.

The Copilot data show the same dependence from the other side. Grouping turn boundaries by the preceding idle gap, the hit rate has medians at or above 95 percent for gaps under two minutes, a median near 70 percent with a long lower tail for gaps of two to ten minutes, and medians near zero to five percent beyond ten minutes (Liu et al., 2026). TraceLab reports the matching pattern: low-hit steps begin to appear after five minutes of idleness and almost all steps miss after an hour (Zhu et al., 2026). Provider retention windows vary from five minutes to 24 hours across offerings, along with minimum cacheable lengths of 1,024 to 4,096 tokens (Lumer et al., 2026). A turn-boundary hit rate measured across 27 models served under several providers' rules is an average over those rules.

6.3 What a client-side trace cannot separate

An API-level trace observes the provider's report of cached tokens and the timing of calls. It does not observe the retention rule, the capacity of each tier, whether a cache domain is shared across users, or how many other sessions competed for memory. Gu et al. (2025) show that some of these properties can be inferred from timing: their statistical audits detect cache sharing across users at seven API providers. The Copilot study states that it has no server-side view and that obtaining one would risk the confidentiality of models and serving engines (Liu et al., 2026). That is a legitimate constraint, but it limits what the trace can establish about serving assumptions. Within the trace, S and V can be partially separated, because model switches and compaction events are logged; S cannot be divided among the provider's rule, its capacity and its load. TraceLab can estimate R and therefore the gap R minus H, but it too observes the policy only through its effects (Zhu et al., 2026).

6.4 Lifespans and their dependence on the population

The cloud-provider study of chat and API traffic reports that KV state is short-lived: the 99th-percentile lifespan of a cache entry in the API workload is 97 seconds, and a cache twice the size of per-GPU memory approaches the ideal hit rate (Wang et al., 2025). In coding-agent traces, reuse gaps within a turn are seconds (a median of 1.2 seconds between consecutive model calls within a turn in the Copilot data) but gaps across turns are minutes to hours (Liu et al., 2026; Zhu et al., 2026). The bimodality is the clearest workload property in the evidence, and it is the right basis for a retention rule. It is not, however, unique to agents: BurstGPT's characterisation of Azure OpenAI services already reported counts and intervals of turns within conversations as a workload feature for serving optimisation (Wang et al., 2024), and multi-turn chat has the same human-paced gap. What differs is the size of the state left behind when the human pauses, discussed in Section 9.

7. Validity: invalidations the harness controls

7.1 Model switches

A KV cache built by one model cannot be read by another. The Copilot study finds model switches in about 6.4 percent of sessions, mostly reactive: they follow errors (a 36 percent non-success rate before switches against 8 percent baseline) or rate limiting, and some are made by the system's own automatic routing rather than by the user (Liu et al., 2026). After a switch the average hit rate is 8 percent, against an average drop of 26 points at same-model boundaries. These are V events, and they originate in the harness and the rate-limiting policy, not in the KV store. The study's own recommendations, pinning sessions to a model and pre-staging the target model's cache when a switch is unavoidable, are harness-and-router changes. A serving design that handles them must expose to the router the cost of breaking a session's cache, which is the kind of cross-layer interface that Parrot's semantic variables and Murakkab's declarative workflow specification were proposed to supply (Lin et al., 2024; Chaudhry et al., 2025).

7.2 Context compaction

When the context approaches the model's limit, coding agents compact it, rewriting older messages into a summary. The Copilot study finds compaction in 7.8 percent of sessions, but those sessions account for 44.2 percent of tokens; the median compaction drops 72.8 percent of prompt tokens, takes about 22 percent of the turn's execution time, and reduces the hit rate on the next call by a median of 66.1 percent (Liu et al., 2026). Compaction is a harness decision, and harnesses differ: a source-level analysis of 13 open-source coding-agent scaffolds finds seven distinct compaction strategies (Rombaut, 2026). The strategies differ in what they do to the prefix. Under exact-prefix caching, an LLM-written summary that replaces the history invalidates everything

after whatever stable header precedes it. Observation masking, which drops older tool outputs, was found on SWE-bench Verified to halve cost relative to the unmanaged agent while matching or slightly exceeding the solve rate of LLM summarisation (Lindenbauer et al., 2025); because masking changes the prompt from the earliest masked observation onward, its cache cost depends on how often the mask moves, and a mask advanced in large, infrequent steps preserves more of the prefix than one that slides at every step. Learned compression guidelines reduce peak tokens by 26 to 54 percent on long-horizon benchmarks (Kang et al., 2025), and parallel compaction reduces the stall that a blocking summarisation call imposes (Cim et al., 2026). None of these papers reports the hit-rate cost on the calls after compaction; Cim et al. preserve the prefix cache only within the compaction call itself, where "each worker's prefix strictly extends the previous worker's, preserving the cache" (Cim et al., 2026). The Copilot data show that the cost is not negligible: the first call after a median compaction loses most of the cached prefix it would otherwise have reused.

The server side has a partial answer that predates agents. CachedAttention handles context-window overflow by decoupling positional encoding from the saved KV state so that truncation does not invalidate it (Gao et al., 2024). Formal work on compaction treats it as a selection or generation game and relates the minimum compaction budget to communication complexity (Tirmazi et al., 2026); it does not include a term for the prefix that a compaction destroys. The decomposition suggests one. The summary a compaction writes is new and must be prefilled in any case; the cache term is the recomputation of retained recent messages that were cached at their old positions and must now be computed at new ones, weighted by S , the probability that their old state was still resident. A compaction triggered just after a long idle gap, when S is already near zero, adds almost nothing in cache terms, because the next call would have recomputed that content anyway; the same compaction mid-turn, when S is near one, converts hits into misses.

7.3 Retries and failures

The Copilot study identifies failure-driven retry loops as a source of compute amplification: tool failures occur in 9 percent of turns, and the "deep loop with failures" archetype runs a median of 36 model calls, four times the overall median, each over a growing context (Liu et al., 2026). This is consistent with the finding, from SWE-bench trajectories, that failed trajectories are longer and more variable than successful ones (Majgaonkar et al., 2025). Retries do not invalidate the prefix; they extend it, and so they raise R . Their effect on serving is through volume and on cost through the cache-read charges for the growing context, which is the subject of the next section.

8. Three senses of "dominated"

Agent serving is variously described as input-heavy, decode-dominated and cache-cost-dominated. The three descriptions are about different quantities, and each is correct for its quantity.

Tokens. By token count, agent calls are overwhelmingly input: the Copilot study reports a median input-to-output ratio above 275 to one (Liu et al., 2026), Yuan et al. (2026) report mean ratios of 53.9 to 559.8, and TraceLab a median of 214 output tokens per step (Zhu et al., 2026). This is the sense in which the Copilot study says the bottleneck shifts from generation to KV-cache efficiency.

Time. By model time, the same workloads are decode-dominated when the cache works. Yuan et al. (2026) find decode accounts for 91.0 to 98.6 percent of model execution time across their configurations, because each step prefills only the short appended suffix. The prefill-decode split that motivated disaggregated serving, a compute-bound prefill and a memory-bound decode, still holds per token (Patel et al., 2023; Zhong et al., 2024), but with high H the prefill per step is small. When H collapses under thrashing, the same workloads shift toward prefill, and the extra prefill slows decode for co-scheduled requests, which is the interference that chunked prefills were designed to bound (Agrawal et al., 2024).

Dollars. By billed cost, agent sessions are dominated by cache charges. TraceLab, pricing its trace at list rates, finds that prefix tokens account for 59.5 percent of cost, appended tokens including cache-write charges for 29.2 percent, and output tokens for 11.2 percent, even though prefix tokens are billed at about one tenth of the fresh-input rate (Zhu et al., 2026). Across 2,848 controlled Claude Code runs, Weinberger and Hozes (2026a) attribute 44.3 percent of the bill to cache creation and 35.4 percent to cache reads, against 10.4 percent for output and 1.3 percent for uncached input.

The three senses recommend different optimisations. If tokens are the target, compressing tool outputs looks attractive; if dollars are, it can backfire, as Section 5.3 noted, because it changes what is cached. If time is the target and the cache works, decode efficiency matters most, which is why Yuan et al. (2026) suggest that agent serving depends less on optimising long prompts than on keeping long-lived state resident. A claim that agent serving is "dominated" by one phase should name its quantity and, for time and dollars, the H at which it was measured.

9. Which serving assumptions the traces break

Table 2 audits the assumptions the Copilot study names, together with two the decomposition adds. For each it records the evidence that the assumption fails for agent traffic, the published systems that already relax it, and

what remains open.

Stateless, independent requests. The study's framing is that serving systems treat requests as short-lived and stateless, with multi-turn conversations supported by replaying history and recovering part of the cost through prefix caching (Liu et al., 2026). This was the premise of chat serving in 2023, and it was abandoned for chat by systems that retain conversation state across turns in GPU and host memory (Gao et al., 2024; Yu et al., 2025) and by KV-cache-centric architectures that hold state in a disaggregated pool of DRAM and SSD (Qin et al., 2024). LMCACHE reports that the total KV state its users store has grown far beyond GPU memory and provides offloading and cross-engine sharing for vLLM and SGLang (Liu et al., 2025); CacheGen and Cake address the cost of reloading stored state over networks and storage links (Liu et al., 2023; Jin et al., 2024). The assumption fails for agents, but it had already been given up for chat; what agents change is that the state at a turn boundary is far larger. The Copilot study compares its 68K median prompt with medians of 750 and 1,050 tokens reported for text-only and multimodal API calls in earlier production studies, one of them DynamoLLM (Liu et al., 2026; Stojkovic et al., 2024). With R near one, the cost of every wrong retention decision scales with that size.

Request-level scheduling. Program-level scheduling for agents is established in the 2024-2026 literature. InferCept treats a tool call as an interception rather than the end of a request and reports that recomputing contexts after interceptions accounted for 37 to 40 percent of model forwarding time in the systems it studied (Abhyankar et al., 2024). Autellix schedules calls by their program's history to reduce head-of-line blocking (Luo et al., 2025). Parrot exposes dependencies among requests through semantic variables (Lin et al., 2024). SAGA treats the agent workflow as the schedulable unit and reports eviction within 1.31 times of Belady's offline optimum when the workflow graph is visible (Guo et al., 2026; Belady, 1966). Conveyor and Sutradhara overlap tool execution with decoding and prefill (Xu et al., 2024; Biswas et al., 2026). The traces strengthen the case for these designs by showing that 87 percent of calls are agent-initiated, but they do not reveal a scheduling mismatch that these research systems have not already targeted; whether production stacks have adopted them is not reported.

LRU eviction. The evidence that workload-agnostic LRU fails for concurrent agents is strong and comes from server-side studies, not from the client-side traces (Biswas et al., 2026; Pan et al., 2025; Yuan et al., 2026; Shao et al., 2026). Replacements include eviction by predicted steps-to-execution (Pan et al., 2025), TTL pinning across tool calls priced by reload cost and queueing delay (Li et al., 2025), ranking programs by relative idleness to place them in GPU or host memory (Xia et al., 2026), agent-aware offloading during

function calls (Bian et al., 2025), and workload-aware eviction for chat and API traffic using per-category reuse probabilities (Wang et al., 2025). Compute-aware admission and eviction, which weighs the compute a hit saves against its memory footprint, was proposed for hybrid models (Pan et al., 2024). What the Copilot traces add is evidence about the cross-turn gap, where these policies, designed for tool gaps or for chat and API traffic, have not been evaluated.

Cache cost proportional to tokens. Implicit in characterising agents by token counts is that cost follows tokens. Section 8 showed that it follows cache traffic and depends on the provider's write premium and read discount. The assumption fails, and the published response is so far measurement rather than design (Zhu et al., 2026; Weinberger and Hozes, 2026a).

The server controls reuse. The decomposition adds this assumption, which no study states but several rely on: that the serving system determines the hit rate. Model switches, compaction and prompt assembly are client decisions (Section 7), and Pythia's distinct per-agent system prompts and Lumer et al.'s cache-block placement show the harness setting R (Yu et al., 2026; Lumer et al., 2026). This assumption is open, but only part of it is new to agents. Context rewriting that invalidates the prefix predates them: in multi-turn chat, truncation on context-window overflow invalidates saved KV state, and with a 4K window such overflow occurs in 30 percent of conversation sessions (Gao et al., 2024); in production API traffic, truncating inputs to the most recent tokens cut one company's prefix hit ratio "from roughly 85% to 45%" (Liu et al., 2025). What the agent traces add is model switching, some of it by automatic routing, and LLM-written compaction of prefixes far longer than chat's.

Retention without a price. The second added assumption is that retention can be decided without an explicit cost model. Prefix caches in general-purpose engines retain whatever fits, under LRU. Continuum is the clearest published exception: it pins KV state across a tool call for a TTL chosen to maximise the expected net benefit of a hit, using the empirical distribution of that tool's past durations, the reload cost and the queueing delay an evicted program would incur (Li et al., 2025). Its scope is the tool gap, measured in seconds. The turn boundary, measured in minutes to hours and with a human on the other side, has no published priced rule. Section 10 states one.

10. The turn boundary as a rent-or-buy decision

10.1 The decision

At the end of a turn, a session's KV state of L tokens is resident in some tier. The server can hold it, paying a holding price per token-second that depends on the tier, or release it, in which case the next call that would have

reused it pays to recompute the prefix, or to reload it from a lower tier. The next reusing call arrives after a random delay T . It may never arrive: the session may end, the harness may switch models, or it may compact the context. This is the rent-or-buy structure, in which renting is holding the state and buying is paying the miss, and its deterministic form has a two-competitive break-even strategy: hold until the accumulated holding cost equals the miss cost, then release (Karlin et al., 1988; Mitzenmacher and Vassilvitskii, 2020). Randomised variants improve the worst-case ratio (Karlin et al., 1994), and learning-augmented variants use a prediction of T to approach the offline optimum when the prediction is good while bounding the damage when it is not (Lykouris and Vassilvitskii, 2018; Mitzenmacher and Vassilvitskii, 2020). The keep-alive problem in serverless platforms has the same structure, and earlier work by several of the same authors used idle-time histograms to set function keep-alive (Shahrad et al., 2020); keep-alive has also been treated explicitly as caching (Fuerst and Sharma, 2021).

10.2 The expected-cost rule

When a distribution for T is available, the expected-cost rule follows by a short derivation. Let $G(t)$ be the probability that no reusing call has arrived by time t , with $G(t)$ allowed to approach a positive limit to represent sessions that never reuse the prefix, and let $f(t)$ be the corresponding density; G is written apart from the survival factor S of Section 4, which concerns the cached state rather than the next call. Let m be the holding price per token-second in the tier, $c(L)$ the cost of recomputing L tokens, and $r(L)$ the cost of restoring them from the tier (zero for GPU memory). Holding until a timeout τ costs $m L$ times the integral of G from zero to τ ; a reusing call that arrives before τ pays $r(L)$; one that arrives after pays $c(L)$. The expected cost is

$$J(\tau) = m L \int_0^\tau G(t) dt + (1 - G(\tau)) r(L) + (G(\tau) - G(\infty)) c(L),$$

and its derivative with respect to τ is $m L G(\tau) - f(\tau) (c(L) - r(L))$. Holding is worth continuing exactly while the hazard rate of a reusing return, $h(t) = f(t) / G(t)$, exceeds the threshold

$$\theta = m L / (c(L) - r(L)).$$

Three consequences follow from the derivation, and they are stated as consequences of it rather than measurements. First, if the hazard rate decreases with elapsed idle time, as it does for heavy-tailed idle distributions, the rule is a timeout: hold until h falls to θ , then release. Optimal timers for TTL caches under general inter-request distributions have been shown to depend on the monotonicity of the hazard rate, with timer policies favoured when it decreases (Ferragut et al., 2016); hit rates of TTL caches have been modelled as functions of the inter-request distribution (Jung et al.,

2003); and elastic TTL caches in the cloud have been provisioned by trading storage cost against miss cost (Carra et al., 2019). Second, invalidation removes probability mass from reusing returns and so lowers the hazard at every t : a session whose harness is likely to switch models or compact before the next call should be given a shorter hold. Third, the threshold falls as L grows if recompute cost grows faster than linearly in L , as it does for attention over long contexts, so longer contexts justify longer holds. The Copilot study's recommendation of archetype-aware retention, with longer holds for deep-loop users whose turns carry a median of 1.1M tokens and shorter for chat-only users at 23K (Liu et al., 2026), is the qualitative form of this third consequence.

Algorithm 1 states the decision with multiple tiers. Continuum's TTL rule for tool gaps is a special case with one tier, an empirical distribution of tool durations in place of G , and a queueing-delay term added to the miss cost (Li et al., 2025); the Copilot study's survival-curve predictor supplies exactly the $G(t)$ the rule needs, including the closed-form update $G(t \mid \text{idle} > t_0) = G(t) / G(t_0)$ as time passes (Liu et al., 2026).

```

Algorithm 1. Turn-boundary KV retention as rent-or-buy
input  session  $s$  at end of turn, prefix length  $L$ 
       $G_s(t)$ : survival of next prefix-reusing call
          (limit  $> 0$  = session ends, switches
           model, or compacts)
      tiers  $k = 1..K$  (GPU, DRAM, storage) with
          hold price  $m_k$ , restore cost  $r_k(L)$ 
      recompute cost  $c(L)$ 
for  $k$  in  $1..K$ :
     $\theta_k = m_k * L / (c(L) - r_k(L))$ 
if  $G_s$  unknown (no predictor):
    hold in tier 1 until accrued hold cost
    reaches  $c(L) - r_1(L)$ , then release
    (deterministic break-even, 2-competitive)
    stop
 $k = \text{cheapest-to-serve tier with } h_s(0) > \theta_k$ 
if none: release now
loop each tick  $t$  while no call has arrived:
     $h = \text{hazard of } G_s \text{ conditioned on } \text{idle} > t$ 
    if  $h < \theta_k$ :
        if  $k < K$  and  $h > \theta_{k+1}$ :
            demote to tier  $k+1$ ;  $k = k+1$ 
        else: release, stop

```

10.3 A worked keep-alive example from published prices

The Copilot study suggests a client-side version of the decision for providers whose retention is time-bounded: when the predicted idle time straddles the provider's retention window, issue a cheap keep-alive request just before the deadline to refresh the entry (Liu et al., 2026). The rent-or-buy rule prices this. TraceLab's pricing snapshot lists, for one family of models, a fresh input price of 5.00 dollars per million tokens, a five-minute cache-write price of 6.25 and a cache-read price of 0.50 (Zhu et al., 2026). On those prices, and ignoring the few output tokens a keep-alive call generates, each refresh of a prefix of L tokens costs 0.50 L per million tokens, while letting the entry lapse costs, on the user's return, a cache write at 6.25 L instead of a read at 0.50 L , a penalty of 5.75 L . The break-even is $5.75 / 0.50 = 11.5$ refreshes.

The deterministic rent-or-buy strategy therefore keeps refreshing for up to eleven windows, about 55 minutes under a five-minute window, and then lets the entry lapse; its cost is at most about twice that of an oracle that knew when the user would return. Because every term is linear in L , the break-even does not depend on the context length; the absolute stakes do.

Against the Copilot idle distribution, the two reported medians lead to different conclusions. A cross-turn gap of 172 seconds needs no refresh under a 300-second window. A user idle time of 25 minutes needs four or five refreshes, costing at most 2.5 L against a 5.75 L penalty, so refreshing pays. A heavy tail extending beyond a day, which the study reports, is the regime where the rent-or-buy cap protects the client. TraceLab's upper-bound estimate of what retaining caches across human thinking time could save, 45.9 percent of appended tokens and 12.8 percent of priced cost in its trace, bounds what any keep-alive policy could recover there before its own refresh costs are subtracted (Zhu et al., 2026).

The same arithmetic shows why server-side and client-side optima differ. For the provider, a refresh costs real memory for the window it extends; for the client it costs only the read price. Whether the listed prices reflect the provider's holding cost is not public, and a client-side keep-alive policy that is rational at list prices may shift cost onto the provider's capacity. The provider's own rule needs the holding price m , which no published study reports for production tiers. A public proxy per tier exists: CachedAttention priced its evaluation at on-demand cloud rates of "\$5/hour per A100 GPU, \$0.0088/hour/GB for DRAM and \$0.000082/hour/GB for SSD" (Gao et al., 2024). Those are rental prices, not a provider's internal cost, but they give the order of magnitude of m in each tier.

10.4 Evaluating an idle-time predictor by the decisions it drives

The Copilot study evaluates its predictor in three ways: a ROC-AUC of 0.73 for predicting whether an idle gap exceeds 60 seconds, against 0.58 for a heuristic using only the previous gap; accuracy and F1 that decline from about 80 to 88 percent at 30 seconds to 25 to 43 percent at 30 minutes; and a "captured idle time" that stays between 86 and 90 percent across thresholds (Liu et al., 2026). The study presents the stability of captured idle time, despite decaying pointwise accuracy, as the property that makes the predictor useful.

Captured idle time measures one side of the decision. On the definition as described, it credits the predictor for idle time it correctly flags as reclaimable; it does not charge the predictor for sessions it flags as idle that return within the window, each of which costs a miss on a prefix that, for a deep-loop user, may exceed a million tokens. A predictor that flagged every boundary as long-idle would, on that reading, also capture all the idle

time. The author's reading of the metric may be incomplete, since the study describes it briefly; the point stands for any metric that rewards reclaimed time without pricing wrong reclaims. Under Algorithm 1, the natural evaluation is the expected cost J of the policy the predictor drives, compared at matched holding prices against a fixed-TTL policy, the deterministic break-even policy, and an oracle. Learning-augmented analysis adds the robustness requirement: a predictor-driven policy should not do much worse than break-even when the predictor is wrong (Lykouris and Vassilvitskii, 2018). Neither comparison has been reported.

11. The evidence in two tables and two figures

Table 1 consolidates the measured values on which the argument rests. Every value is reported by the cited source; the last column records which factor of the decomposition the source's design varied, which is the author's classification.

Figure 1 plots the hit-rate rows of Table 1 as paired conditions. The spread within a study, where the workload is fixed and in most rows one factor moves, is as large as the spread between studies.

Figure 2 plots TraceLab's idealised eviction-timeout sweep: the storage that suspended sessions occupy, relative to active ones, against the prefill amplification that results.

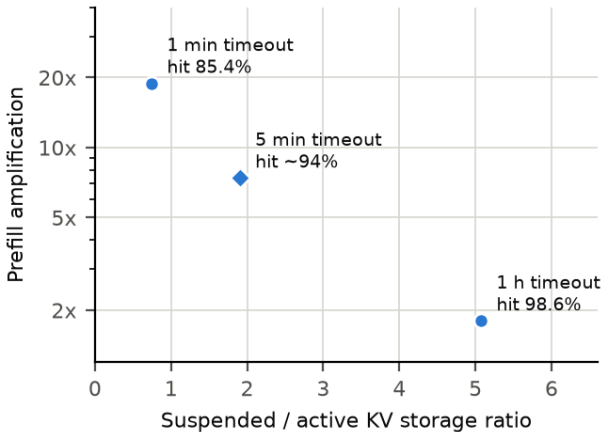


Figure 2. Retention trade-off in the TraceLab trace under an idealised timeout rule: storage ratio of suspended to active KV state against prefill amplification, at eviction timeouts of 1 minute, 5 minutes and 1 hour, labelled with the achievable hit rate. The source reports the 5-minute storage ratio and hit rate as approximate (diamond); it plots a continuous curve of which only these three points are stated in its text. Values from Zhu et al. (2026).

Table 2 summarises the audit of Section 9.

12. Tests that would settle the open part

The decomposition makes the open questions specific. Each test below names the factor it isolates.

T1. Miss-ratio curves for agent and chat traces under one server. Replay an agent trace and a chat trace

through the same engine at matched capacity per active session and sweep capacity, recording the hit rate at each point. Stack-distance analysis gives the whole curve from one pass (Mattson et al., 1970), and the working-set estimate of EfficientAgent shows it can be done for agent histories (Shao et al., 2026). The claim that agents differ from chat in their cache behaviour becomes a claim about the curves, which hold S's dependence on capacity in view, rather than about two single numbers measured under different stacks. The Copilot authors state that they plan to release sanitised traces (Liu et al., 2026); TraceLab's are public (Zhu et al., 2026). A comparison chat trace with content hashes exists for one provider (Wang et al., 2025).

T2. Turn-boundary retention priced against the predictor. On a held-out week of traces, compute the expected cost J of Algorithm 1 driven by the survival predictor, by the deterministic break-even rule, by fixed TTLs of 5 and 60 minutes, and by an oracle, at two or three assumed holding prices per tier. Report the cost per archetype, since the miss cost differs by a factor of about 50 between chat-only and deep-loop users (Liu et al., 2026). This replaces captured idle time as the measure of the predictor's value.

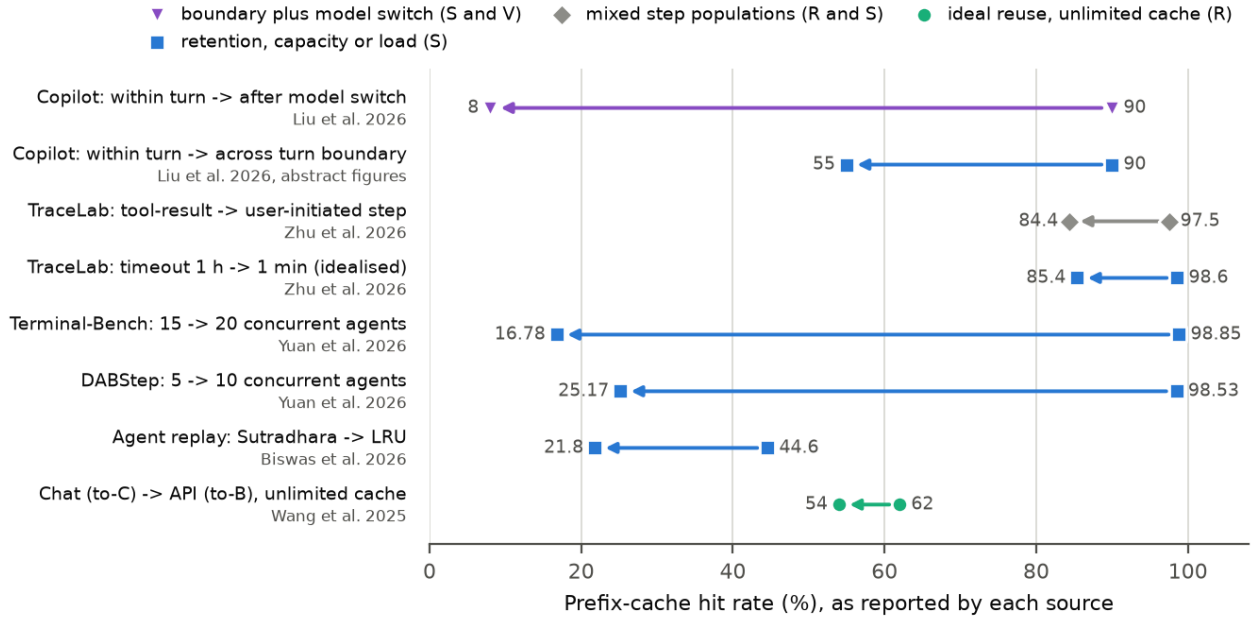
T3. Compaction that preserves the prefix. Run matched agent tasks with three compaction strategies, LLM summary, observation masking advanced in large infrequent steps, and truncation with KV kept valid, and report task success, tokens, the hit rate on the call after compaction, and billed cost. The literature has compared compaction strategies on success and tokens (Lindenbauer et al., 2025; Kang et al., 2025) and has measured the cache cost of compaction in production (Liu et al., 2026), as well as the cost of truncation in production API traffic, a prefix hit ratio falling from about 85 to 45 percent (Liu et al., 2025), but not both in one design. This isolates V.

T4. Router-aware model switching. In a system that switches models on rate limits, compare switching at turn boundaries against staged switching that prefills the target model's cache before the switch, as the Copilot study proposes, and report the hit rate and latency on the first post-switch call. This isolates the cost of V events that the router controls.

T5. Server-side ground truth for one client-side trace. For any provider that can release it under confidentiality constraints, publish the retention rule and the per-tier capacity in effect during a characterised week, even as ranges. This is the single disclosure that would let a client-side hit rate be divided into $R \times V$ and S . Gu et al. (2025) show that some of it can be audited from outside; disclosure is cheaper than inference.

Table 1. Measured values from agent and chat serving studies, by source, with the factor each comparison varies (R reuse potential, S survival, V validity).

Source	Setting	Condition	Reported value	Varies
Liu et al. 2026	Copilot, client-side	within turn, call 3 on	hit 92-94%	none
Liu et al. 2026	Copilot	same-model turn boundary	hit down 26 pts avg	S
Liu et al. 2026	Copilot	after model switch	hit 8%	S, V
Liu et al. 2026	Copilot	after compaction	hit down 66.1% median	V
Zhu et al. 2026	Claude Code, Codex API	tool-result / user steps	hit 97.5% / 84.4%	R, S
Zhu et al. 2026	same trace	fresh share of prefilled tokens	19.0%	R, S
Zhu et al. 2026	idealised timeout sweep	1 min / 1 h	hit 85.4% / 98.6%	S
Yuan et al. 2026	Terminal-Bench, own server	15 / 20 agents	hit 98.85% / 16.78%	S
Yuan et al. 2026	DABStep, own server	5 / 10 agents	hit 98.53% / 25.17%	S
Yuan et al. 2026	five benchmarks	no thrashing	decode 91.0-98.6% of time	none
Biswas et al. 2026	replay, 14B, A100	LRU / Sutradhara	hit 21.8% / 44.6%	S
Yu et al. 2026	agent-serving traces	agent-serving platform traces	over 40% of requests zero or very limited	R, S
Shao et al. 2026	SWE-bench agents	host tier at working set	prefill -93.1%, time -38.7%	S
Wang et al. 2025	chat / API, infinite cache	ideal hit	62% / 54%	R
Weinberger, Hozez 2026a	Claude Code runs	share of bill	write 44.3%, read 35.4%	none

**Figure 1.** Published prefix-cache hit rates for agent and chat workloads, each row a pair of conditions reported within one source; the arrow runs from the favourable to the unfavourable condition. Colour and marker give the factor that changed: retention, capacity or load (S); a turn boundary combined with a model switch, so survival and validity together (S and V); step populations that differ in fresh share as well as in idle gap (mixed, R and S); or, for the last row, two workloads compared at their ideal rate under unlimited capacity (R). The Copilot within-turn value of 90 and the turn-boundary value of 55 are the source's abstract figures. Values from Liu et al. (2026), Zhu et al. (2026), Yuan et al. (2026), Biswas et al. (2026) and Wang et al. (2025).**Table 2.** Serving assumptions said to be broken by agent traffic, with the evidence and the published systems that relax them.

Assumption	Evidence it fails for agents	Already relaxed by	Open part
Stateless requests	R 0.88-0.99 per call	CachedAttention, Pensieve, Mooncake, LMCake	size of state at turn boundary
Request-level scheduling	87% of calls agent-initiated	InferCept, Autellix, Parrot, SAGA	cross-turn priority
LRU eviction	hit 98.85% to 16.78% under load	KVFlow, Continuum, MORI, TokenCake	human-paced gaps
Cost follows tokens	cache charges about 80% of bill	none; measured only	priced retention
Server controls reuse	switch and compaction invalidate	none	harness-server interface
Retention needs no price	TTL chosen by default	Continuum (tool gaps)	turn-boundary rule

13. The strongest case against this paper

The Copilot study already says this. The study attributes the turn-boundary cliff to a time-based eviction policy, cites a 300-second default, and proposes keep-alive requests to work around it (Liu et al., 2026). On this objection the present paper restates a point the source makes. The reply is that the source makes it in its body while its abstract and key-findings table present the cache numbers as properties of agentic coding, its comparison table describes chat cache sensitivity as low and independent, and its conclusion addresses current serving systems in general. The decomposition is needed to turn the body's observation into a statement about which assumptions are broken, and the chat comparison is contradicted by the production study of chat reuse that this synthesis located (Wang et al., 2025).

For an operator the observed rate is what matters. A provider planning capacity cares about the hit rate its own stack achieves, not about a hypothetical stack with a different rule. This is correct, and the paper does not dispute the operational value of observed rates. It disputes the inference from observed rates to claims about what agent workloads require of serving systems in general, which is the inference the systems literature draws when it cites agent traces as motivation.

Rent-or-buy is textbook. The derivation in Section 10 is standard, and Continuum has already applied a priced TTL to tool gaps (Li et al., 2025). The contribution here is the application to the turn boundary with an explicit invalidation term, the observation that the predictor the Copilot study built supplies the input the rule needs, and the critique of evaluating it by captured idle time. A reader who regards that as a modest contribution is not being unreasonable.

The decomposition is too clean. R , S and V are not independent in practice. Higher load raises both the chance of eviction and the chance of rate-limit-driven model switches, coupling S and V ; compaction lowers R for later calls as well as invalidating the current prefix; and partial-prefix survival makes S a fraction rather than a probability. The identity $H = R \times V \times S$ holds per call by definition, but averaging across calls with correlated factors does not factor cleanly. The paper uses the decomposition to classify comparisons, where it is robust, not to predict aggregate rates from separately estimated factors, where it would not be.

14. What is not known

The holding price of KV state per token-second in each tier of a production serving system is not published. Without it, the threshold in Algorithm 1 cannot be set from public information, and the worked example of Section 10.3 uses list prices, which reflect pricing decisions rather than costs. On-demand rental prices per tier, such as those CachedAttention used, are a proxy for m , not a measurement of it (Gao et al., 2024).

The distribution of the time to a user's next turn, conditioned on what a server can observe, is known only to the extent of one predictor with a ROC-AUC of 0.73 at a 60-second threshold (Liu et al., 2026). Whether richer features would do better is not known; the study reports that alternative model classes changed the ROC-AUC only marginally.

How the turn-boundary hit rate in the Copilot traces varies across the 27 models and their providers' retention rules is not reported. The aggregate is therefore not interpretable as the behaviour of any one stack.

Whether prefix-preserving compaction costs task success relative to summarisation in production coding agents is not known. The benchmark evidence on observation masking is encouraging (Lindenbauer et al., 2025) but comes mainly from one scaffold, SWE-agent, with what its authors call "initial evidence" on OpenHands, and from one benchmark.

The extent of cross-user prefix sharing in agent traffic is not known. Shared system prompts across sessions raise R at the cost of a timing side channel (Gu et al., 2025), and none of the agent characterisations reports how much of its hit rate comes from cross-session sharing.

Whether the Copilot workload, collected in the United States over one week in June 2026, represents other regions, other agents or later versions of the same agent is not known; the study notes that new tools, models and autonomy strategies appear weekly, while reporting that its characterisation results remained consistent from January to June 2026 (Liu et al., 2026).

15. Limitations

The paper rests on a small number of 2026 characterisation studies, several of them preprints that have not been replicated, and its central anchor is a single industrial trace whose server side is not visible. Table 1 juxtaposes values measured under different statistics, per-call medians in one case and token-weighted means in another, and Figure 1 should be read as showing within-study contrasts, not as a common scale. The classification of each comparison by factor is the author's, and some rows, such as Pythia's, mix factors. The derivation in Section 10 assumes a single reuse event per boundary, holding costs linear in L , and a known hazard, and it ignores the queueing effects that Continuum shows can dominate miss costs at high load (Li et al., 2025). The worked example uses one provider family's list prices as reported by TraceLab and ignores the output tokens of keep-alive calls. The paper does not assess the tool-container half of the Copilot study's resource analysis.

16. Conclusion

A prefix-cache hit rate reported for an agent workload is the product of the reuse the agent makes available, the survival its cached state is granted by the server, and the

validity the harness leaves it. Agentic coding has very high reuse potential, 0.88 to 0.99 against an ideal 0.54 to 0.62 for chat and API traffic, and that difference is real and large. Whether it becomes a hit is decided by the retention rule, the capacity and the load, which move the same workload's hit rate from 99 to 17 percent, and by harness events such as model switches and compaction, which the server cannot prevent. The serving assumptions the traces break are therefore of three kinds: some were abandoned for chat before agents arrived, some are being replaced by program-level and priced retention in 2025-2026 systems, and two are new at least in part: that the server controls reuse, which truncation in chat and API traffic had already contradicted and which agents break further through model switching and LLM-written compaction of far longer prefixes, and that a turn boundary needs no priced rule. The turn boundary is a rent-or-buy problem whose threshold is a hazard rate; the Copilot predictor supplies its input, and it should be judged by the cost of the decisions it drives. The comparisons that would settle what is specific to agents, miss-ratio curves under one server and priced retention evaluated against an oracle, are specifiable and can be run on traces that are already public or promised.

References

- Abhyankar, R., He, Z., Srivatsa, V., Zhang, H., & Zhang, Y. (2024). InferCept: Efficient Intercept Support for Augmented Large Language Model Inference. *arXiv:2402.01869*. doi:10.48550/arXiv.2402.01869
- Agrawal, A., Kedia, N., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B. S., Tumanov, A., & Ramjee, R. (2024). Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. *arXiv:2403.02310*. doi:10.48550/arXiv.2403.02310
- Ali, M. S., Novik, A., Boddupally, A., et al. (2026). The Harness Effect: How Orchestration Design Sets the Token Economics of Enterprise Agentic AI. *arXiv:2607.06906*. doi:10.48550/arXiv.2607.06906
- Belady, L. A. (1966). A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2). doi:10.1147/sj.52.0078
- Bian, Z., Wu, F., Li, Z., Ma, T., & Zhuo, Y. (2025). TokenCake: A KV-Cache-centric Serving Framework for LLM-based Multi-Agent Applications. *arXiv:2510.18586*. doi:10.48550/arXiv.2510.18586
- Biswas, A., Goel, K., S, S., Mohan, J., Khare, A., Parayil, A., Ramjee, R., & Bansal, C. (2026). Sutradhara: An Intelligent Orchestrator-Engine Co-design for Tool-based Agentic Inference. *arXiv:2601.12967*. doi:10.48550/arXiv.2601.12967
- Carra, D., Neglia, G., & Michiardi, P. (2019). TTL-based Cloud Caches. *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. doi:10.1109/INFOCOM.2019.8737546
- Chaudhry, G. I., Choukse, E., Qiu, H., Goiri, I., Fonseca, R., Belay, A., & Bianchini, R. (2025). Murakkab: Resource-Efficient Agentic Workflow Orchestration in Cloud Platforms. *arXiv:2508.18298*. doi:10.48550/arXiv.2508.18298
- Che, H., Tung, Y., & Wang, Z. (2002). Hierarchical Web caching systems: modeling, design and experimental results. *IEEE Journal on Selected Areas in Communications*, 20(7). doi:10.1109/JSAC.2002.801752
- Cim, M., Topcu, B., Das, C., & Kandemir, M. T. (2026). Parallel Context Compaction for Long-Horizon LLM Agent Serving. *arXiv:2605.23296*. doi:10.48550/arXiv.2605.23296
- Denning, P. J. (1968). The working set model for program behavior. *Communications of the ACM*, 11(5). doi:10.1145/363095.363141
- Ferragut, A., Rodriguez, I., & Paganini, F. (2016). Optimizing TTL Caches under Heavy-Tailed Demands. *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*. doi:10.1145/2896377.2901459
- Fuerst, A., & Sharma, P. (2021). FaasCache: keeping serverless computing alive with greedy-dual caching. *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. doi:10.1145/3445814.3446757
- Gao, B., He, Z., Sharma, P., Kang, Q., Jevdjic, D., Deng, J., Yang, X., Yu, Z., & Zuo, P. (2024). Cost-Efficient Large Language Model Serving for Multi-turn Conversations with CachedAttention. *arXiv:2403.19708*. doi:10.48550/arXiv.2403.19708
- Gim, I., Chen, G., Lee, S., Sarda, N., Khandelwal, A., & Zhong, L. (2023). Prompt Cache: Modular Attention Reuse for Low-Latency Inference. *arXiv:2311.04934*. doi:10.48550/arXiv.2311.04934
- Gu, C., Li, X. L., Kuditipudi, R., Liang, P., & Hashimoto, T. (2025). Auditing Prompt Caching in Language Model APIs. *arXiv:2502.07776*. doi:10.48550/arXiv.2502.07776
- Guo, D., Wu, J., & Yiu, S. M. (2026). SAGA: Workflow-Atomic Scheduling for AI Agent Inference on GPU Clusters. *arXiv:2605.00528*. doi:10.48550/arXiv.2605.00528
- Jin, S., Liu, X., Zhang, Q., & Mao, Z. M. (2024). Compute Or Load KV Cache? Why Not Both? *arXiv:2410.03065*. doi:10.48550/arXiv.2410.03065
- Jung, J., Berger, A. W., & Balakrishnan, H. (2003). Modeling TTL-based Internet caches. *IEEE INFOCOM 2003*. doi:10.1109/INFCOM.2003.1208693
- Kang, M., Chen, W.-N., Han, D., Inan, H. A., Wutschitz, L., Chen, Y., Sim, R., & Rajmohan, S. (2025). ACON: Optimizing Context Compression for Long-horizon LLM Agents. *arXiv:2510.00615*. doi:10.48550/arXiv.2510.00615
- Karlin, A. R., Manasse, M. S., Rudolph, L., & Sleator, D. D. (1988). Competitive snoop caching. *Algorithmica*, 3. doi:10.1007/BF01762111
- Karlin, A. R., Manasse, M. S., McGeoch, L. A., & Owicki, S. (1994). Competitive randomized algorithms for nonuniform problems. *Algorithmica*, 11. doi:10.1007/BF01189993
- Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., & Stoica, I. (2023). Efficient Memory Management for Large Language Model Serving with PagedAttention. *Proceedings of the 29th Symposium on Operating Systems Principles*. *arXiv:2309.06180*. doi:10.48550/arXiv.2309.06180
- Li, H., He, R., Mang, Q., Zhang, Q., Mao, H., Chen, X., Zhou, H., Zhang, H., Cheung, A., Gonzalez, J., & Stoica, I. (2025). Continuum: Efficient and Robust Multi-Turn LLM Agent Scheduling with KV Cache Time-to-Live. *arXiv:2511.02230*. doi:10.48550/arXiv.2511.02230
- Lin, C., Han, Z., Zhang, C., Yang, Y., Yang, F., Chen, C., & Qiu, L. (2024). Parrot: Efficient Serving of LLM-based Applications with Semantic Variable. *arXiv:2405.19888*. doi:10.48550/arXiv.2405.19888
- Lindenbauer, T., Slinko, I., Felder, L., Bogomolov, E., & Zharov, Y. (2025). The Complexity Trap: Simple Observation Masking Is as Efficient as LLM Summarization for Agent Context Management. *arXiv:2508.21433*. doi:10.48550/arXiv.2508.21433
- Liu, B., Qiu, H., Goiri, I., Fonseca, R., Bianchini, R., & Choukse, E. (2026). Agentic Coding in the Wild: Characterizing GitHub Copilot Traces at Production Scale. *arXiv:2608.00101*. doi:10.48550/arXiv.2608.00101
- Liu, Y., Li, H., Cheng, Y., Ray, S., Huang, Y., Zhang, Q., Du, K., Yao, J., Lu, S., Ananthanarayanan, G., Maire, M., Hoffmann, H., Holtzman, A., & Jiang, J. (2023). CacheGen: KV Cache Compression and Streaming for Fast Large Language Model Serving. *arXiv:2310.07240*. doi:10.48550/arXiv.2310.07240

- Liu, Y., Cheng, Y., Yao, J., An, Y., Chen, X., Feng, S., Huang, Y., Shen, S., Zhang, R., Du, K., & Jiang, J. (2025). LMCACHE: An Efficient KV Cache Layer for Enterprise-Scale LLM Inference. arXiv:2510.09665. doi:10.48550/arXiv.2510.09665
- Lumer, E., Nizar, F., Jangiti, A., Frank, K., Gulati, A., Phadate, M., & Subbiah, V. K. (2026). Don't Break the Cache: An Evaluation of Prompt Caching for Long-Horizon Agentic Tasks. arXiv:2601.06007. doi:10.48550/arXiv.2601.06007
- Luo, M., Shi, X., Cai, C., Zhang, T., Wong, J., Wang, Y., Wang, C., Huang, Y., Chen, Z., Gonzalez, J. E., & Stoica, I. (2025). Autellix: An Efficient Serving Engine for LLM Agents as General Programs. arXiv:2502.13965. doi:10.48550/arXiv.2502.13965
- Lykouris, T., & Vassilvitskii, S. (2018). Competitive caching with machine learned advice. arXiv:1802.05399. doi:10.48550/arXiv.1802.05399
- Mahendrakar, P. (2026a). Three Things Called Budget Awareness: Observability, Forecasting and Allocation in LLM Agents, Why Every Published Allocation Gain Is Keyed to a Signal Measured After the Fact, and the Run-to-Run Variance No Forecast Is Scored Against. Life of Research. doi:10.5281/zenodo.22884949
- Mahendrakar, P. (2026b). Compressed Once, Read Many Times: Why Prompt-Compression Results Do Not Transfer to Agent Memory, What the Query-Agnostic Line Already Showed, and the Two Properties of a Memory Write No Protocol Yet Scores. Life of Research. doi:10.5281/zenodo.22969435
- Majgaonkar, O., Fei, Z., Li, X., Sarro, F., & Ye, H. (2025). Understanding Code Agent Behaviour: An Empirical Study of Success and Failure Trajectories. arXiv:2511.00197. doi:10.48550/arXiv.2511.00197
- Mattson, R. L., Gecsei, J., Slutz, D. R., & Traiger, I. L. (1970). Evaluation techniques for storage hierarchies. IBM Systems Journal, 9(2). doi:10.1147/sj.92.0078
- Mitzenmacher, M., & Vassilvitskii, S. (2020). Algorithms with Predictions. arXiv:2006.09123. doi:10.48550/arXiv.2006.09123
- Pan, R., Wang, Z., Jia, Z., Karakus, C., Zancato, L., Dao, T., Wang, Y., & Netravali, R. (2024). Marconi: Prefix Caching for the Era of Hybrid LLMs. arXiv:2411.19379. doi:10.48550/arXiv.2411.19379
- Pan, Z., Patel, A., Hu, Z., Shen, Y., Guan, Y., Li, W.-L., Qin, L., Wang, Y., & Ding, Y. (2025). KVFlow: Efficient Prefix Caching for Accelerating LLM-Based Multi-Agent Workflows. arXiv:2507.07400. doi:10.48550/arXiv.2507.07400
- Patel, P., Choukse, E., Zhang, C., Shah, A., Goiri, I., Maleki, S., & Bianchini, R. (2023). Splitwise: Efficient generative LLM inference using phase splitting. arXiv:2311.18677. doi:10.48550/arXiv.2311.18677
- Qin, R., Li, Z., He, W., Zhang, M., Wu, Y., Zheng, W., & Xu, X. (2024). Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving. arXiv:2407.00079. doi:10.48550/arXiv.2407.00079
- Rombaut, B. (2026). Inside the Scaffold: A Source-Code Taxonomy of Coding Agent Architectures. arXiv:2604.03515. doi:10.48550/arXiv.2604.03515
- Shahrad, M., Fonseca, R., Goiri, I., Chaudhry, G., Batum, P., Cooke, J., Laureano, E., Tresness, C., Russinovich, M., & Bianchini, R. (2020). Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. arXiv:2003.03423. doi:10.48550/arXiv.2003.03423
- Shao, K., Chen, J., Yu, J., Li, X.-H., Tao, C., Wang, Y., Lin, H., Cheng, K.-T., Tsui, C. Y., & Bai, H. (2026). EfficientAgent: What Makes KV Cache Offloading Work for Concurrent Agents? arXiv:2609.33762. doi:10.48550/arXiv.2609.33762
- Stojkovic, J., Zhang, C., Goiri, I., Torrellas, J., & Choukse, E. (2024). DynamoLLM: Designing LLM Inference Clusters for Performance and Energy Efficiency. arXiv:2408.00741. doi:10.48550/arXiv.2408.00741
- Tirmazi, H., Markelon, S., Bishop, A., & Mitzenmacher, M. (2026). Context Compaction Theory. arXiv:2608.01326. doi:10.48550/arXiv.2608.01326
- Wang, J., Han, J., Wei, X., Shen, S., Zhang, D., Fang, C., Chen, R., Yu, W., & Chen, H. (2025). KVCache Cache in the Wild: Characterizing and Optimizing KVCache Cache at a Large Cloud Provider. arXiv:2506.02634. doi:10.48550/arXiv.2506.02634
- Wang, Y., Chen, Y., Li, Z., Kang, X., Fang, Y., Zhou, Y., Zheng, Y., Tang, Z., He, X., Guo, R., Wang, X., Wang, Q., Zhou, A. C., & Chu, X. (2024). BurstGPT: A Real-world Workload Dataset to Optimize LLM Serving Systems. arXiv:2401.17644. doi:10.48550/arXiv.2401.17644
- Weinberger, S., & Hozes, A. (2026a). Token Reduction Is Not Cost Reduction. arXiv:2607.12161. doi:10.48550/arXiv.2607.12161
- Weinberger, S., & Hozes, A. (2026b). Prompt-Induced Waste in Coding Agents: Reasoning, Effort, Harness Design, and End-to-End Cost. arXiv:2608.01347. doi:10.48550/arXiv.2608.01347
- Xia, T., Li, H., Li, Z., Chen, X., Kang, H., Qiao, Y., Xu, Y., & Stoica, I. (2026). Idleness is Relative: Exploiting Tool-Call Idle Windows for Offloading in Agentic Systems with MORI. arXiv:2606.00866. doi:10.48550/arXiv.2606.00866
- Xu, Y., Kong, X., Chen, T., & Zhuo, D. (2024). Conveyor: Efficient Tool-aware LLM Serving with Tool Partial Execution. arXiv:2406.00059. doi:10.48550/arXiv.2406.00059
- Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., & Press, O. (2024). SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. arXiv:2405.15793. doi:10.48550/arXiv.2405.15793
- Yao, J., Li, H., Liu, Y., Ray, S., Cheng, Y., Zhang, Q., Du, K., Lu, S., & Jiang, J. (2024). CacheBlend: Fast Large Language Model Serving for RAG with Cached Knowledge Fusion. arXiv:2405.16444. doi:10.48550/arXiv.2405.16444
- Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022). ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629. doi:10.48550/arXiv.2210.03629
- Yu, L., Lin, J., & Li, J. (2025). Stateful Large Language Model Serving with Pensieve. Proceedings of the Twentieth European Conference on Computer Systems. doi:10.1145/3689031.3696086
- Yu, S., Shu, J., Ni, Y., Qian, K., Li, X., Wang, Y., Zhang, J., Xu, Z., Yang, S., Zhu, L., et al. (2026). Pythia: Exploiting Workflow Predictability for Efficient Agent-Native LLM Serving. arXiv:2604.25899. doi:10.48550/arXiv.2604.25899
- Yuan, Y., Nayak, A., Kundu, S., & Talati, N. (2026). Agentic AI Workload Characteristics. arXiv:2605.26297. doi:10.48550/arXiv.2605.26297
- Zheng, L., Yin, L., Xie, Z., Sun, C., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., Barrett, C., & Sheng, Y. (2023). SGLang: Efficient Execution of Structured Language Model Programs. arXiv:2312.07104. doi:10.48550/arXiv.2312.07104
- Zhong, Y., Liu, S., Chen, J., Hu, J., Zhu, Y., Liu, X., Jin, X., & Zhang, H. (2024). DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. arXiv:2401.09670. doi:10.48550/arXiv.2401.09670
- Zhu, K., Jacob, M., Ma, C., Pan, Y., Wang, S., Krishnamurthy, A., & Kasikci, B. (2026). TraceLab: Characterizing Coding Agent Workloads for LLM Serving. arXiv:2606.30560. doi:10.48550/arXiv.2606.30560