CAN A MAIN AGENT REUSE THE KV CACHE OF ITS SUBAGENTS? A SYSTEMATIC EMPIRICAL STUDY

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent language systems decompose a task among a main agent and its subagents, but this parallelism is paid for in input tokens: the conclusion a subagent returns is prefilled again from scratch once it is concatenated into the main agent's prompt, even though the subagent has already computed its key–value (KV) cache. Reusing that cache directly is a form of position-independent caching (PIC); however, unlike the whole-document reuse studied so far, a subagent's conclusion is an interior span that carries no attention sink, and its reuse fidelity has not been characterized. We ask a fundamental question: can a main agent reuse the KV states that its subagents produce, and when can it not? We build these capabilities into a vLLM-based serving system that grafts subagent KV states directly into the main agent's cache without recomputation, and we answer the question empirically along three axes—whether a reused span preserves its content (a Retell test), whether the main agent can answer questions about a single reused span, and whether reuse survives when many spans must be combined. We find that reusing a single subagent's conclusion closely tracks full recomputation even with no recomputation of its own—within single-digit F1 points, and near-lossless for the strongest model—consistent with the absence of an attention sink in an interior span, but that accuracy falls sharply once the main agent must combine many reused spans at once. Because single-span reuse already succeeds without recomputation, this aggregation gap is not explained by the attention sink alone and points to a further, still-uncharacterized obstacle to jointly reusing many partial segments.

1 INTRODUCTION

Large language models (LLMs) have become general-purpose engines for tasks once considered distinctly human. Trained on broad corpora, models such as Qwen3 and GLM-4.7 follow instructions, reason over long contexts, and generate fluent text and code (Qwen Team, 2025b; Z.ai, 2026). Their impact is now visible across domains: they write and repair software (Chen et al., 2021; Jimenez et al., 2024), operate productivity applications and computer interfaces (Xie et al., 2024), and assist—or increasingly automate—stages of scientific research (Lu et al., 2024). A central driver of this breadth is the *agent* abstraction, which lets an LLM invoke external tools, observe their results, and act on the world rather than merely emit text (Yao et al., 2023; Schick et al., 2023). Agentic frameworks extend this idea to multiple cooperating agents (Wu et al., 2023; Hong et al., 2023; Li et al., 2023): a system can decompose a task, dispatch subagents in parallel, and finish sooner than a single agent would. This capability is expensive, however, and the expense is paid in input tokens, which dominate the cost of long agent prompts (OpenAI, 2026; Anthropic, 2026a; Google, 2026). Multi-agent execution compounds the problem: running k agents over shared context can multiply token consumption several-fold relative to a single agent.

To reduce token consumption, a growing body of work reuses the *key–value (KV) cache* so that identical input need not be recomputed on every call. Because the KV states of a prompt segment can be computed once and reused, providers bill such reused (cached) input tokens at a fraction—often around one tenth—of the normal input price (Anthropic, 2026b). In the multi-agent setting, two forms of reuse arise. The first is *within-context* reuse inside a single agent. This form is not specific

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.

to multi-agent systems: in any multi-turn session, a later turn can reuse the KV cache of earlier turns while it generates new tokens. The second is *across-context* reuse between agents. When a subagent is forked from the main agent, for example, the subagent inherits the main agent's prefix and need not recompute it, reusing the main agent's KV cache directly (Kwon et al., 2023; Zheng et al., 2024; Pan et al., 2025). Such transfer is valid only when the agents share the same underlying model.

These reuse mechanisms share a strict requirement: the cached tokens must form an exact prefix of the new prompt. Agentic workloads, unfortunately, have two characteristics that cause prefix matching to miss a large fraction of reuse opportunities. First, agents load many documents—skills, memories, and reference material—whose contents recur across requests but whose position within the prompt is neither controllable nor predictable (Packer et al., 2023). The KV cache of a given document or skill can therefore rarely be reused, even though such documents recur constantly across agentic requests. Second, in multi-agent workflows the main agent dispatches subagents to investigate or execute subtasks, and each returns a chunk of tokens. The KV cache for that chunk was already computed on the subagent side, yet it cannot be handed to the main agent directly: once the returned chunk is concatenated into the main agent's prompt, its preceding context—and hence its prefix—has changed.

To overcome the prefix constraint, prior work has proposed *position-independent caching* (PIC), which takes two forms. The first is whole-document reuse. Each document, skill, or memory is prefilled once in isolation from position zero, and its KV cache is stored. When the item is later needed, the stored KV states are retrieved and concatenated, and a light recomputation restores their validity in the new context. A common recipe recomputes the first few tokens of each item, which removes the attention-sink artifacts introduced by prefilling from position zero (Yao et al., 2025; Hu et al., 2025; Yang et al., 2025; Gim et al., 2024). The second form is partial-segment reuse, in which only an interior span of a previously prefilled KV cache is reused. Such a span typically excludes the first few tokens and therefore carries no attention sink. How readily and how faithfully these spans can be reused has not been studied. Yet this is precisely the case that arises when a main agent reuses the KV cache corresponding to a subagent's conclusion.

In this paper we study partial-segment PIC, because it is exactly the form of reuse that occurs in the interaction between a main agent and its subagents. The KV states a subagent produces for its conclusion necessarily encode the content of that conclusion, and may further entangle information from the preceding context. This raises a fundamental question: if these states already carry the conclusion, why can the main agent not simply reuse them, and when reuse does degrade quality, why? Answering this question is the goal of our study.

We answer three questions in turn. **(Q1) Does a partial-segment PIC cache faithfully preserve the content of its span?** We design a *Retell* experiment in which the main agent is asked to restate a subagent's conclusion directly from the grafted KV cache, and find that it reproduces the span with a token-level F1 of 89–99% across three models (89.5, 90.1, and 98.6), against 95.7–100 for full recomputation. **(Q2) Can the main agent act on a reused span, not merely reproduce it?** We then ask the main agent to answer questions about a single reused span and find that its accuracy stays within roughly two percentage points of full recomputation on average—corpus-averaged F1 gaps of 0.03–1.72 pp at two chunks, widening to 2.21–3.90 pp at ten—essentially matching it. **(Q3) Does reuse hold when many spans must be combined?** Finally, in an end-to-end deep-research workflow we feed back the conclusions of multiple subagents per turn and measure final-answer quality; accuracy degrades as the number of subagents grows.

Our analysis yields two findings that separate partial-segment PIC from whole-document PIC by one difference while aligning them by one similarity. The difference concerns the attention sink. Because a partial span usually omits the opening tokens, it exhibits no attention sink; a single subagent's conclusion can therefore be grafted and reused as is, with no recomputation, and both Retell fidelity and single-span question answering stay within single-digit F1 points of full recomputation (near-lossless for the strongest model). For reusing one document, the absence of an attention sink is thus what makes reuse inexpensive, though a nonzero residual gap remains. The similarity concerns aggregation. As with whole-document PIC, accuracy falls sharply once a query depends on many reused spans jointly, and in the end-to-end agentic trace quality degrades markedly as the number of subagent conclusions fed back to the main agent grows—even though each span individually carries no attention sink. Because single-span reuse already succeeds without recomputation, this aggregation gap cannot be attributed to the attention sink; whether many partial segments can be

reused together therefore appears to be governed by a further, still-uncharacterized factor, which we leave as an open problem for future work on partial-segment PIC—one that, we conjecture, may call for recomputation operating across documents rather than within a single span.

2 Background and Related Work

2.1 KV Cache

We study decoder-only Transformer language models with causal self-attention (Vaswani et al., 2017). Inference has two phases: *prefill* processes the prompt and stores, for every layer and head, the key and value vectors of each token—the *key–value (KV) cache* (Kwon et al., 2023)—and *decoding* then generates tokens one at a time, attending to this cached history without recomputing it. Because prefill cost grows with prompt length, reusing the KV cache of repeated input across requests is a major serving optimization, and it underlies the reduced price that providers charge for cached input tokens.

Cross-request reuse takes two forms, distinguished by whether the cached tokens must form an exact prefix of the new prompt. **Prefix-based reuse** serves the common initial tokens of a request—system instructions, few-shot examples, or a shared conversation history—from cache. PagedAttention manages the cache in fixed-size blocks to reduce fragmentation and enable sharing (Kwon et al., 2023); RadixAttention organizes reusable prefixes in a radix tree (Zheng et al., 2024); and RAGCache and KVFlow extend prefix reuse to retrieval and to agent workflows, respectively (Jin et al., 2024; Pan et al., 2025). Prefix reuse is exact, but it applies only when the cached tokens are a contiguous prefix, so identical text preceded by a different context cannot be reused. **Position-independent caching (PIC)** relaxes this constraint, reusing a cached segment under a different preceding context at the cost of limited recomputation. Prompt Cache reuses schema-defined modules (Gim et al., 2024); CacheBlend recomputes a small subset of token states to restore cross-chunk attention (Yao et al., 2025); EPIC formalizes PIC and recomputes a few leading tokens per chunk to remove spurious attention sinks (Hu et al., 2025); and KVLink, Cache-Craft, and CacheSlide further study independent encoding, chunk selection, and position shifts (Yang et al., 2025; Agarwal et al., 2025; Liu et al., 2026).

Both forms confront the same underlying fact: a segment's KV states depend on the context that precedes them. Let $\mathcal{C}(s \mid c)$ denote the KV entries for a token segment s obtained by processing the concatenation $c \parallel s$. Even under identical model weights and identical tokens in s,

$$\mathcal{C}(s \mid c_{\text{sub}}) \neq \mathcal{C}(s \mid c_{\text{main}}), \tag{1}$$

because a different preceding context changes both the positions of s and its hidden representations. Correcting the rotary position embedding (RoPE) of cached keys realigns positions but does not remove the influence of a different prefix on higher-layer keys and values (Su et al., 2023; Yang et al., 2025). Prefix reuse sidesteps this problem by retaining the original causal context, whereas PIC must contend with it. This distinction is central to our question of whether a main agent can reuse the KV states that its subagents produce.

2.2 Agents and Multi-Agent Systems

An LLM-based *agent* uses model calls to pursue a task, maintaining task context and selecting actions such as tool calls; ReAct interleaves reasoning and acting (Yao et al., 2023), and Toolformer teaches models to call tools directly (Schick et al., 2023). Multi-agent frameworks coordinate multiple agents with distinct roles (Wu et al., 2023; Hong et al., 2023; Li et al., 2023). We call the coordinating agent the *main agent* and an agent to which it delegates a subtask a *subagent*: the main agent supplies a task description and context, each subagent runs its own model–tool loop, and the returned result is appended to the main agent's context as a tool message. Subagents may run sequentially or concurrently, so one main-agent prompt can gather many subagent outputs. Different agents may share the same weights while keeping distinct instructions and histories, but sharing weights does not imply sharing KV states: in text-mediated interaction the receiver reprocesses the returned text as part of its own prompt, and the sender's KV cache is not transferred with it.

Agents often obtain context through retrieval-augmented generation (RAG), which prepends retrieved document chunks to the prompt (Lewis et al., 2020). The chunks recur across requests, creating reuse

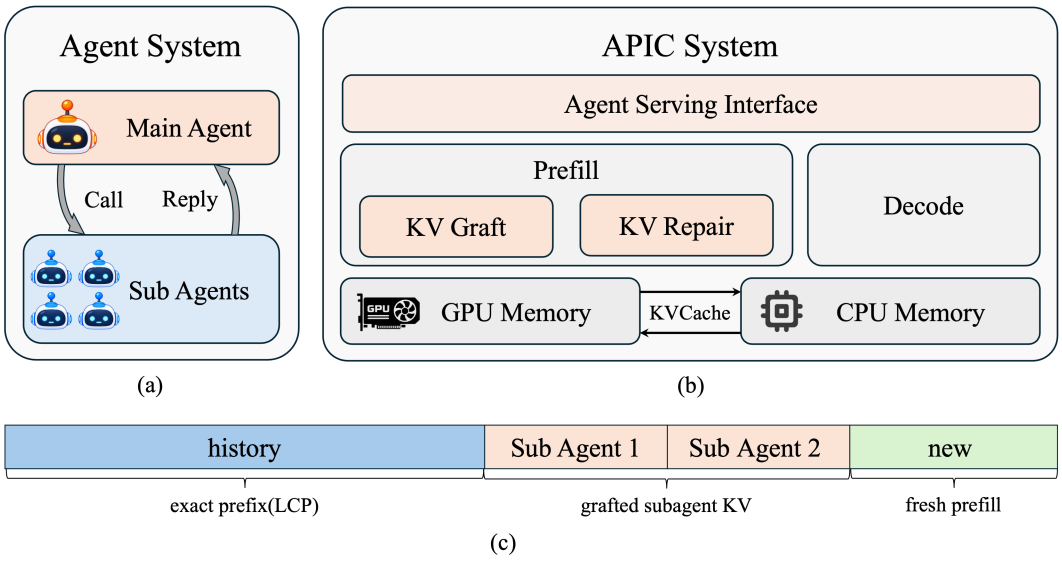

Figure 1: Overview of subagent KV cache reuse. (a) Agent collaboration: a main agent invokes multiple subagents, which return their responses to the main agent. (b) An inference system adapted to support subagent KV reuse, enabling the main agent to reuse the KV caches of subagent responses. (c) The token sequence of the main agent's subsequent input, with colors distinguishing the exact prefix whose KV cache is reused, the concatenated response tokens from subagents sub_1 and sub_2 whose KV caches are reused, and new tokens requiring prefill.

opportunities, but their surrounding context and order vary, placing RAG in the position-independent rather than the prefix setting.

Reuse across agents has begun to receive direct attention. TokenDance shares KV caches collectively across the synchronized rounds of multi-agent systems, storing sibling caches as sparse differences from a shared copy (Bian et al., 2026), and Liang et al. (2026) show that reuse can weaken cross-candidate interactions and change an LLM judge's selections even when aggregate accuracy is stable—a text match alone does not guarantee useful reuse. Our work targets a specific point in this space: a main agent reusing the KV cache a subagent produced for its conclusion. Because that conclusion is an interior span rather than a prompt prefix, the setting is a form of PIC—one that Equation 1 makes non-trivial.

3 System Overview

Figure 1 presents our system, which combines a deep research agent application with an inference system that supports subagent KV cache reuse. The application coordinates research through main–subagent interactions, while the inference system retains and reuses KV states when subagent responses become part of the main agent's input.

Deep research agent system. We implement a hierarchical research workflow in which a main agent breaks a user query into focused subtasks and delegates them to subagents (Figure 1(a)). Each subagent maintains its own interaction history and uses search and reading tools to gather evidence from a document collection. It returns its findings to the main agent as a tool result. The main agent integrates these findings, identifies remaining gaps, and requests further research or synthesizes a final answer. The workflow also supports a reviewer subagent that checks a proposed answer against the sources. Repeated delegation and synthesis place previously generated subagent responses into subsequent main-agent prompts, creating the reuse opportunity targeted by our inference system.

Agent Serving Interface. We implement the inference system with a vLLM execution backend and three components: Agent Serving Interface, KV Graft, and KV Repair (Figure 1(b)); as discussed

below, KV Repair is currently a no-op placeholder that we retain as an extension point for future work. The serving interface embodies a co-design of the agent and inference layers: instead of treating the model as an opaque text-in, text-out service, we make the agent aware of the underlying inference system and let it expose its execution structure, so that the inference system can in turn reuse state across agent calls. Concretely, the serving interface extends OpenAI-compatible chat completion requests with an agent mode, a session identifier, and an agent execution trace. These fields associate model calls with the same research session and distinguish main-agent and subagent requests. The server retains token-aligned KV snapshots from these requests in CPU memory, allowing reuse across agent calls without keeping all retained snapshots resident on the GPU. The application exchanges messages and session metadata without directly manipulating KV tensors.

KV Graft. When the main agent resumes, the system matches its new prompt against stored history to reuse an exact common prefix, and locates matching subagent response tokens in the rendered prompt. KV Graft inserts the corresponding cached states in prompt order and supports RoPE rebasing to align cached keys with their destination positions. As illustrated in Figure 1(c), the resulting token sequence combines an exact reusable prefix, concatenated subagent responses, and new tokens. A custom KV connector loads the selected cached states from CPU memory into vLLM's GPU paged KV storage; segmented prefill computes the intervening tokens and remaining suffix before normal decoding. Because a subagent's conclusion is an interior span that carries no attention sink, KV Graft reuses its states directly, without recomputing any tokens within the span; only unmatched tokens and template delimiters are prefilled normally. The system tracks the exact-prefix boundary to prevent grafted (approximate) states from being treated as exact prefix hits in later rounds, and falls back to exact-prefix reuse with suffix recomputation when a graft plan fails validation.

KV Repair. KV Repair is a stage that would recompute selected states of a grafted span under the main agent's preceding context. Because grafting interior spans without any recomputation already preserves quality for single-document reuse (Section 5), the deployed system leaves this component as a no-op; we keep it as an explicit extension point for future recomputation algorithms aimed at the multi-document regime, and discuss this in Section 4.

4 IMPLEMENTATION

We build an agent runtime and an inference service on top of vLLM, connected through a custom KV connector that requires no changes to vLLM's source. Rather than layering an agent over an unmodified text-in, text-out API, we co-design the agent and inference layers: the service extends OpenAI-compatible chat completion requests with a session identifier and an execution trace that make the inference system aware of the agent's task and delegation structure, associating model calls with one research task and distinguishing main-agent from subagent requests. Exposing this agent-level structure to the serving system is precisely what lets it retain token-aligned KV snapshots in CPU memory and reuse them across agent calls.

Reuse is realized by *KV Graft*, which locates the subagent response spans in the main agent's rendered prompt, rebases their rotary position embeddings to the destination positions, and composes them with freshly prefilled tokens through segmented prefill, leaving the causal attention mask unchanged. Because a subagent's conclusion is an interior span that carries no attention sink, we graft its cached states directly and reuse them without recomputing any tokens within the span; only unmatched tokens—report formatting, template delimiters, and the tokens between grafted spans—are prefilled normally. Appendix A details the deep-research runtime, the agent session API, and the graft procedure.

KV Repair: a currently-null component. Our architecture reserves a separate KV Repair stage that would, in principle, recompute part of a grafted span under the main agent's preceding context. We initially adapted the recomputation recipes of EPIC (Hu et al., 2025) and CacheBlend (Yao et al., 2025) into this stage, but found that reusing an interior span without any recomputation already comes within single-digit F1 points of full prefill for a single document (Section 5); the recomputation therefore served no purpose, and we disabled it. KV Repair is thus currently a null component: it is present in the system but performs no work. We keep it as an explicit extension point for future

work. The natural next step is to explore recomputation algorithms for the multi-document regime—reusing on the order of ten to twenty or more independently prefilled documents at once—where our experiments show quality degrading. We conjecture that such repair may not be expressible as single-document, within-span recomputation of the kind EPIC applies; restoring quality when many spans are combined may instead require recomputation that operates *across* documents, reconciling states that were each prefilled in isolation. We leave the design of such cross-document repair to future work.

5 Evaluation

5.1 Experimental Setup

We evaluate subagent KV reuse at three levels: reproducing an actual subagent response (**KVCache-Retell**), answering questions from a cached suffix (**KVCache-QA**), and synthesizing answers across repeated main–subagent interactions (**Agent-Trace**). These experiments separate fidelity to the returned text, the accessibility of information in reused states, and final task quality.

Models and platform. We evaluate Qwen3-8B, Qwen3-30B-A3B-Instruct-2507, and GLM-4.7-Flash using their released checkpoints (Qwen Team, 2025a;b; Z.ai, 2026). These models cover both dense and mixture-of-experts (MoE) architectures and different attention implementations. Qwen3-8B is dense and Qwen3-30B-A3B-Instruct-2507 is MoE; both use full attention with grouped-query attention (GQA). GLM-4.7-Flash combines MoE with multi-head latent attention (MLA). This selection allows us to examine KV reuse across different model architectures and KV representations. We abbreviate Qwen3-30B-A3B-Instruct-2507 as Qwen3-30B-A3B in figures. Experiments run on a server equipped with eight NVIDIA H100 GPUs. Throughout, KV reuse grafts each matched subagent span directly into the main agent's cache without recomputing any of its tokens: an interior span carries no attention sink, so we reuse it as is.

Comparison and metrics. We compare KV reuse against full prefill, which recomputes the KV states from the supplied text. We evaluate answer quality using token-level F1, reported as percentages; score differences are expressed in percentage points (pp).

5.2 KVCache-Retell: Fidelity to Generated Subagent Responses

Protocol. This experiment tests whether the main agent can access the subagent's output content through the KV states corresponding to its output tokens, using reproduction of that content as an operational probe. We randomly select 20 long evidence documents from BrowseComp-Plus (Chen et al., 2025). For each document, the main agent delegates reading to a subagent, which produces factual notes from the supplied text. We vary the subagent's generation budget over $\{64, 128, 256, 512, 1024, 2048\}$ tokens, yielding 120 cases per model and condition. The main agent is then instructed to reproduce the returned notes. KV reuse transfers the response's cached states to the main agent; full prefill re-encodes the returned text. The generation budget is an upper bound rather than an exact returned-text length: generation can stop early, and incomplete trailing sentences are removed before the response is returned. The runs use temperature zero and disable thinking. We compute token-level F1 for each reproduction against the text actually returned by its own subagent, not against a human summary of the source document.

Results. Figure 2 shows that the main agent reproduces the subagent's output with high token F1 when reusing its cached states. Across 120 cases per model, reuse achieves 89.47, 90.10, and 98.59 F1 for Qwen3-8B, Qwen3-30B-A3B, and GLM-4.7-Flash, respectively, compared with 97.72, 95.69, and 100.00 under full prefill. The corresponding gaps are 8.25, 5.59, and 1.41 pp, indicating that reuse retains much of the baseline's reproduction quality, with particularly close agreement for GLM. Content remains accessible even at the 2,048-token generation budget: reuse achieves 94.12, 84.77, and 92.06 F1, respectively. Although the remaining gap varies across models and budgets, the high scores show that the main agent can recover substantial output content from the transferred KV states.

Implications for output accessibility. These results support subagent KV reuse as a promising way to make subagent outputs accessible to the main agent. Reproducing the text associated with the

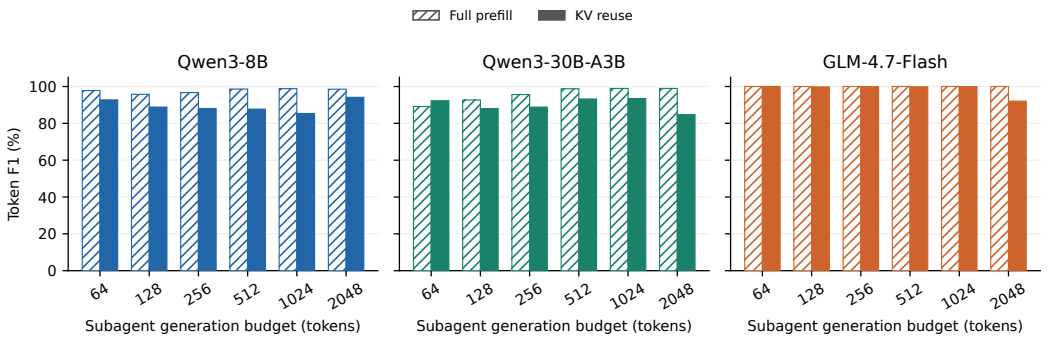

Figure 2: KVCache-Retell token F1 (percentage scale) versus subagent generation budget, with 20 documents per bar. Solid bars denote KV reuse and hatched bars full prefill. Each condition generates its own target response; the bars measure workflow-level reproduction fidelity.

transferred states requires access to the output content itself, not merely to selected facts within it. The high reproduction F1 therefore provides evidence that the main agent can effectively "see" much of the subagent's output through reused KV, without fully re-encoding that output. This is evidence of substantial content accessibility, rather than guaranteed recovery of every token: F1 measures lexical overlap and does not establish exact sequence reproduction.

5.3 KVCache-QA: Information Access from a Cached Suffix

Task construction. We use long contexts from HotpotQA (Yang et al., 2018), 2WikiMultihopQA (Ho et al., 2020), and the PassageCount task in LongBench (Bai et al., 2024). Each context is divided into $C \in \{2, 5, 10\}$ chunks. Using only the final chunk, we generate one question at each of **five question levels**: **L1, Fact Retrieval**; **L2, Information Aggregation**; **L3, Reasoning**; **L4, Deep Analysis**; and **L5, Multi-hop / Synthesis**. These are newly constructed questions rather than the original benchmark questions. Each model–corpus–chunk configuration contains 200 contexts and 1,000 question–answer evaluations, balanced at 200 questions per level. Inspired by RULER's aggregation and multi-hop tasks (Hsieh et al., 2024) and LongBench's diverse QA tasks (Bai et al., 2024), this five-level diagnostic distinguishes fact access from increasingly compositional uses of cached information. The taxonomy is specific to our study and does not assume a calibrated difficulty ordering.

Protocol and motivation. Let $x_1, \ldots, x_C$ denote the document chunks. We first prefill their concatenation and retain only the states associated with the final chunk, x_C. In the reuse setting, the model answers a question by reusing these cached suffix states, whereas the baseline jointly prefills all chunks together with the question. We focus on the final chunk because its states are computed under a preceding context that is unavailable at reuse time, matching the key mismatch encountered when a receiving agent reuses states produced by a subagent. This experiment therefore isolates the effect of reusing suffix states under a changed prefix.

Results. Figure 3 shows that suffix KV retains substantial question-relevant information, but the gap depends on the model and source corpus. With two chunks, the mean over the three corpora changes from 46.33 to 44.61 for Qwen3-8B, from 47.57 to 46.17 for Qwen3-30B-A3B, and from 50.36 to 50.33 for GLM-4.7-Flash. At ten chunks, the corresponding gaps widen to 3.90, 3.41, and 2.21 pp. PassageCount-derived questions are the most sensitive in this setting: the ten-chunk gaps are 5.22, 6.50, and 8.02 pp, respectively.

Implications for subagent KV reuse. These results highlight substantial potential for subagent KV reuse: in this suffix-QA diagnostic, reuse achieves answer quality close to full prefill, with corpus-averaged gaps of only 0.03–1.72 pp at two chunks and 2.21–3.90 pp at ten chunks across the three models. Fact access is particularly well preserved: at ten chunks, the mean F1 decreases from 86.58 to 84.70, compared with losses of 3.19–3.74 pp for L2–L5. The larger gaps at ten chunks are consistent with greater sensitivity when the retained suffix is conditioned on a larger preceding portion

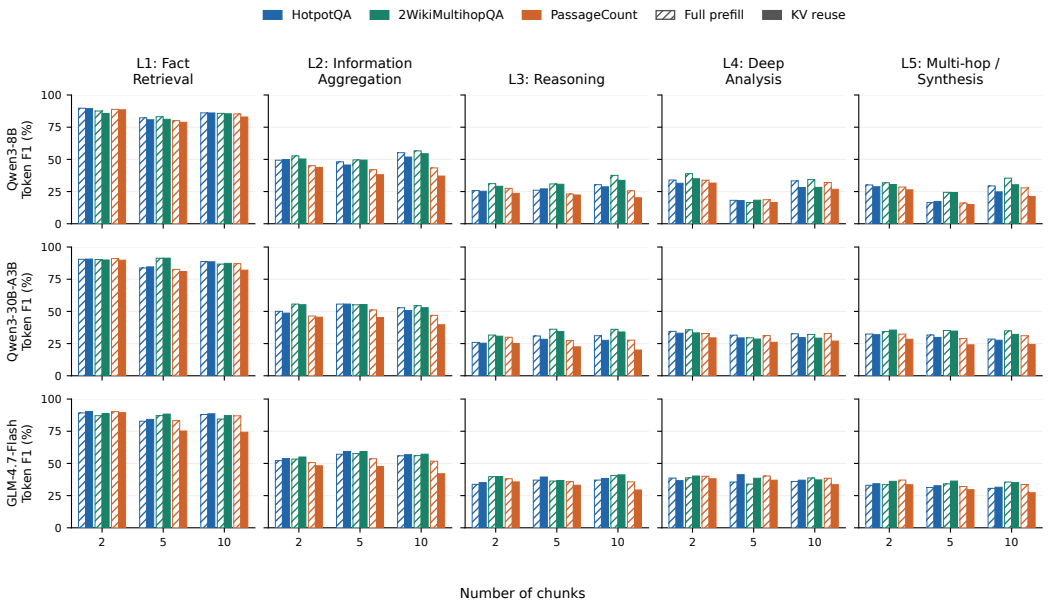

Figure 3: KVCache-QA token F1. Solid bars denote KV reuse and hatched bars full prefill; rows identify the three models and columns the five question levels. Colors identify source corpora; the horizontal axis gives the chunk count. Each bar averages 200 questions at the indicated level. All panels share a 0–100 F1 scale.

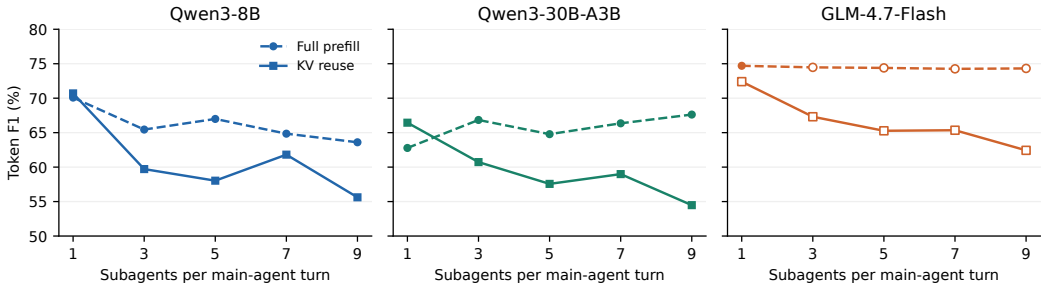

Figure 4: Agent-Trace token F1 as a function of subagents per turn. Solid lines denote KV reuse and dashed lines full prefill. Scores follow the archived summaries, which exclude execution failures where completion counts are available. Hollow markers denote summaries without matching per-query validation: GLM at $k > 1$ lacks per-query records, and the updated GLM reuse summary at $k = 1$ differs from its archived per-query records.

of the context. A possible explanation is that a longer source prefix introduces more prefix-dependent information into the suffix KV states, increasing their mismatch with the receiving agent's context.

5.4 Agent-Trace: Quality under Iterative Delegation

Workload. We use 200 BrowseComp-Plus queries and retain their supporting documents while removing distractor documents. BrowseComp-Plus supplies a fixed corpus with supporting evidence and challenging negatives (Chen et al., 2025); our evidence-only variant isolates reading, delegation, and synthesis from retrieval over those negatives. It is therefore not directly comparable to the original benchmark's leaderboard. A streamlined deep-research workflow implemented in BenchAgent lets the main agent delegate document reading, integrate returned findings, and iteratively produce a final answer. We compare KV reuse with full prefill at $k \in \{1, 3, 5, 7, 9\}$ subagents per main-agent turn. This is a per-turn configuration, not the total number of subagent calls in a task; actual trajectories and total call counts may differ between conditions.

Final-answer quality. Figure 4 shows that the effect of reuse becomes more pronounced in the multi-response settings. At $k = 1$, Qwen3-8B and Qwen3-30B-A3B have F1 scores of 70.70 and 66.44 under reuse, versus 70.09 and 62.78 for full prefill. At $k = 9$, their reuse scores are 55.62 and 54.48, compared with 63.60 and 67.62 for full prefill: gaps of 7.98 and 13.14 pp. Every Qwen configuration with $k > 1$ has lower F1 under reuse, although the gap is not strictly monotonic in k. GLM's summaries show the same direction, from a 2.30 pp gap at $k = 1$ to 11.88 pp at $k = 9$.

Implications for multi-agent KV reuse. Agent-Trace reveals both the promise and the current limits of reuse in an end-to-end workflow. With one subagent per turn, reuse achieves comparable final-answer quality to full prefill across the three models, showing that transferred KV states can support subsequent reasoning and task completion. With multiple subagents per turn, however, all evaluated models score below full prefill, and the gaps at $k = 9$ are substantially larger than at $k = 1$. Together with KVCache-QA and KVCache-Retell, this suggests that making individual outputs accessible is not sufficient to preserve quality when the main agent must combine multiple outputs. Because each reused span individually carries no attention sink and single-span reuse needs no recomputation, this aggregation gap cannot be attributed to the attention sink, and its cause remains open. One possible explanation is that states produced under different subagent contexts become harder to integrate into a shared main-agent context, with discrepancies affecting subsequent reasoning and delegation. The runs do not isolate this mechanism, since reuse can also change the execution trajectory. Overall, Agent-Trace extends the evidence for KV reuse from local content access to end-to-end task execution: the main agent can use transferred subagent states to integrate findings, continue reasoning, and produce final answers. The close agreement with full prefill at $k = 1$ particularly highlights the potential of KV transfer as a communication mechanism between agents.

5.5 Overall Findings

Together, the three experiments tell a consistent story about partial-segment reuse without recomputation. KVCache-Retell shows that the main agent can reproduce much of a single subagent's output content directly from its grafted states, and KVCache-QA shows that it can answer questions about that content; in both single-span settings, reuse stays close to full prefill even though no tokens within a span are recomputed, consistent with the absence of an attention sink in an interior span. Agent-Trace extends these capabilities to iterative reasoning and task completion, and with one subagent per turn reuse matches full prefill—but as the number of subagent conclusions combined per turn grows, quality degrades sharply. That single-span reuse succeeds without recomputation while multi-span aggregation degrades indicates that the attention sink governs whether a single document can be reused, yet does not explain why many partial segments resist being reused together; we leave this aggregation obstacle as an open question for future work on partial-segment PIC, and conjecture that addressing it may require recomputation across documents rather than the single-document, within-span recomputation used by prior whole-document PIC. Overall, the high single-span fidelity and the close agreement at one subagent per turn highlight the potential of KV states as a communication mechanism between agents, allowing the main agent to use subagent outputs without fully re-encoding them.

6 Conclusion

We have studied whether a main agent can reuse the key–value cache that its subagents produce—partial-segment position-independent caching, where the reused span is an interior region of a generation, not a prompt prefix. Reusing a single subagent's conclusion preserves its content and supports downstream reasoning with little loss and needs no recomputation, because an interior span carries no attention sink. Combining many spans at once is markedly harder: quality degrades as the number of subagent conclusions grows. Because single-span reuse already succeeds without recomputation, this gap is not explained by the attention sink; we leave its cause, and the cross-document repair it may require, to future work. Overall, subagent-to-main cache transfer is a practical way to cut the token cost of multi-agent systems—at least when a single conclusion is reused at a time—and we hope it encourages serving systems to treat cross-agent KV reuse as a first-class optimization.

AI USE STATEMENT

We used generative AI tools as assistants throughout this work, under author supervision, in two ways. First, we used LLM assistance for drafting and editing the prose of this paper; the authors wrote and verified every technical claim, number, and conclusion. Second, we used LLM-based coding assistants to help implement parts of the agent runtime, the serving system, and the experiment scripts, and to help debug and run experiments; the authors reviewed the resulting code and validated all reported results. We did not use generative AI to fabricate data, citations, or experimental results, and we take full responsibility for the final text, code, and claims of this work.

REFERENCES

Shubham Agarwal, Sai Sundaresan, Subrata Mitra, Debabrata Mahapatra, Archit Gupta, Rounak Sharma, Nirmal Joshua Kapu, Tong Yu, and Shiv Saini. Cache-Craft: Managing chunk-caches for efficient retrieval-augmented generation. arXiv:2502.15734v1, 2025.

Anthropic. Anthropic API pricing. `https://www.anthropic.com/pricing`, 2026a.

Anthropic. Prompt caching. `https://docs.anthropic.com/en/docs/build-with-claude/prompt-caching`, 2026b.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, 2024.

Zhuohang Bian, Feiyang Wu, Chengrui Zhang, Hangcheng Dong, Yun Liang, and Youwei Zhuo. TokenDance: Scaling multi-agent LLM serving via collective KV cache sharing. arXiv:2604.03143, 2026.

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

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Sahel Sharifymoghaddam, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. BrowseComp-Plus: A more fair and transparent evaluation benchmark of deep-research agent. arXiv:2508.06600, 2025.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. arXiv:2311.04934v2, 2024.

Google. Gemini API pricing. `https://ai.google.dev/gemini-api/docs/pricing`, 2026.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. arXiv:2308.00352, 2023.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024.

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Qin Zhang, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. EPIC: Efficient position-independent caching for serving large language models. arXiv:2410.15332v3, 2025.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. RAGCache: Efficient knowledge caching for retrieval-augmented generation. arXiv:2404.12457v2, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. arXiv:2309.06180v1, 2023.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, 2020.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. arXiv:2303.17760, 2023.

Sichu Liang, Zhenglin Wang, Chujiajia, Pengfei Xia, Hui Zang, and Deyu Zhou. When KV cache reuse fails in multi-agent systems: Cross-candidate interaction is crucial for LLM judges. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7205–7230, 2026.

Yang Liu, Yunfei Gu, Liqiang Zhang, Chentao Wu, Guangtao Xue, Jie Li, Minyi Guo, Junhao Hu, and Jie Meng. CacheSlide: Unlocking cross position-aware KV cache reuse for accelerating LLM serving. In *24th USENIX Conference on File and Storage Technologies (FAST 26)*, pp. 83–99, 2026.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist: Towards fully automated open-ended scientific discovery. arXiv:2408.06292, 2024.

OpenAI. OpenAI API pricing. `https://openai.com/api/pricing/`, 2026.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. arXiv:2310.08560, 2023.

Zaifeng Pan, Ajjkumar Patel, Zhengding Hu, Yipeng Shen, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. KVFlow: Efficient prefix caching for accelerating LLM-based multi-agent workflows. arXiv:2507.07400v1, 2025.

Qwen Team. Qwen3-8B model card. Hugging Face model repository, 2025a.

Qwen Team. Qwen3-30B-A3B-Instruct-2507 model card. Hugging Face model repository, 2025b.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. arXiv:2104.09864v5, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. arXiv:2308.08155v2, 2023.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, 2024.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. KVLink: Accelerating large language models via efficient KV cache reuse. arXiv:2502.16002v4, 2025.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, 2018.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. arXiv:2405.16444v3, 2025.

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

Z.ai. GLM-4.7-Flash model card. Hugging Face model repository, 2026.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. arXiv:2312.07104v2, 2024.

A Implementation Details

This appendix expands the implementation summarized in Section 4.

Software stack and implementation size. We implement the agent runtime in Python and the inference service using vLLM for model execution, PyTorch for tensor operations, and Transformers for tokenization and model utilities. The recorded backend validation environment uses Transformers 5.12.1, PyTorch 2.11.0+cu130, and a vLLM 0.23.1rc1 development build at commit `a2f713002`. FastAPI and Uvicorn provide the HTTP service. The core BenchAgent and LMInfer packages contain 1,252 and 5,200 physical lines of Python, respectively, including comments, docstrings, and embedded prompts, but excluding tests, experiments, and example scripts. The latter count covers both inference backends and shared serving utilities; the four modules dedicated to vLLM execution, planning, and KV transfer account for 633 lines. The vLLM integration uses a custom KV connector without modifying vLLM's source. The current backend executes requests serially on one GPU and supports dense Qwen3 models with full attention and FP16/BF16 KV states.

A.1 Deep Research Runtime

Research state and evidence access. BenchAgent implements deep research over a supplied document collection. Each investigation creates a source catalog with stable document IDs and immutable text snapshots. A SQLite FTS5 index stores overlapping passages of up to 3,000 characters with a stride of 2,700 characters. Search returns ranked passages with source IDs and character offsets; reading tools retrieve additional context by offset. This separates evidence access from agent reasoning and makes returned findings traceable to the original documents.

Delegation and synthesis. The main agent maintains the investigation history and decomposes the query into focused research questions. Each delegation starts a separate subagent history containing its instructions, the research question, and selected task context. The subagent runs a model–tool loop and returns findings and unresolved gaps; the runtime appends the result to the main agent's history as a tool message. Only one delegation is executed per main turn, allowing the main agent to reconsider the evidence before choosing the next task. Subagents do not delegate further.

The runtime provides two research workflows. The general workflow keeps a shared evidence ledger linking claims and verbatim quotations to query requirements. A claim can be recorded only if its quotation occurs in a passage read by the worker. Before accepting a final answer, a reviewer subagent must reopen the cited passages and approve the same answer, synthesis, and evidence list; adding evidence invalidates the previous review. These checks establish quotation provenance and review coverage, while semantic assessment remains a model judgment. A document-scoped workflow instead assigns one document per delegation, supplies bounded passage packets, and allows targeted search and reading within that document. It requires an initial pass over all documents before targeted reinspection and filters findings whose quotations are absent from the supplied passages. Both workflows bound model calls and tool iterations, record execution traces and token usage, and expose unresolved gaps when research cannot be completed. Structured reports may be normalized before returning to the main agent; consequently, cache reuse depends on matching the actual rendered tokens, rather than assuming that every report is returned verbatim.

A.2 Agent Session API

Request association. An agent session represents one research task spanning multiple model calls. Requests use `/v1/chat/completions` with three additional fields: `mode="agent"`, `session_id`, and `trace`. The first request omits the session ID; the server creates one and returns it to the shared client for subsequent main-agent and subagent calls. A missing trace is rejected, as is an unknown session ID. The trace records execution transitions, and its final entry identifies the active role: `main` denotes the main agent and other names denote subagents. For example, successive requests may carry `[main]`, `[main, sub]`, and `[main, sub, main]`. The trace remains unchanged during a subagent's internal tool loop and extends when control returns to the main agent or enters another subagent.

Snapshot lifecycle. After generation, the server associates the request's token sequence and KV snapshot with its session and trace. It retains the latest main-agent snapshot and one latest snapshot per subagent invocation, replacing intermediate snapshots from the same invocation. Each entry records the output's starting offset and the valid exact-prefix length. Only tokens that have actually passed through the model are stored: a final sampled token without computed KV is excluded. Retained session snapshots reside in CPU memory and selected states are loaded for GPU execution.

When a main request resumes, the server selects reuse candidates and transfers ownership of the current subagent batch to that request before execution. Consumed references are released after use; subagent entries arriving later belong to a new batch and are not removed by the earlier request's completion. Session bookkeeping runs in the server event loop, while model execution runs through the serialized backend. Both streaming and non-streaming completions update usage and cache metadata after generation. An explicit release endpoint and idle-session cache eviction bound the lifetime of retained snapshots. If execution fails after cache consumption, a retry can recompute from the prompt text.

A.3 KV Graft

Locating reusable response spans. Graft planning occurs when control returns from a subagent to the main agent, after the server renders the chat template and tokenizes the new prompt but before prefill. The planner first computes the longest common prefix with stored history. Beyond that prefix, it identifies tool-result windows using the tokenizer's role delimiters and matches them against stored subagent output tokens. For each window, it searches for the longest contiguous token match among the remaining subagent candidates in invocation order. Each subagent output contributes at most one span, with a minimum match length of four tokens. The selected span records its source offset, destination offset, tokens, and per-layer KV slices. Tokens changed by report formatting, template insertion, or boundary tokenization remain outside the match and are recomputed. Before execution, the planner checks token equality, source lengths, destination bounds, and non-overlap; invalid plans fall back to exact-prefix reuse and suffix prefill.

Composing KV during prefill. The engine processes valid spans in destination order. Starting from the reusable exact prefix, it prefills the gap preceding a span, appends the selected subagent KV along the sequence dimension at every layer, and continues with the next gap. For a span shifted from

source position s to destination position t, RoPE rebasing applies the relative rotation $R(t-s)$ to cached keys while leaving values unchanged. The custom connector maps the composed prefix into vLLM's paged KV blocks, including spans that do not align with block boundaries. Each computed interval is executed as a contiguous suffix of the available cache through an internal vLLM request. Intermediate sampled tokens are discarded; only the final stage generates the user-visible response. The planner leaves the final prompt token for computation to obtain next-token logits. Thus, disjoint reuse spans are realized through segmented execution without changing the causal attention mask.

Approximate reuse and fallback. Grafted states were produced under a different preceding context, so they are approximate rather than exact. The cache records the first grafted (approximate) position, and subsequent exact-prefix matching cannot cross that boundary, preventing approximate states from later being treated as an exact prefix hit. An exact mode disables grafting and recomputes the suffix as a reference execution path, which we use as the full-prefill baseline in our experiments.