MuLA: One Prefill, Many Agents via Efficient KV Cache Reuse in Multi-LoRA Agents Systems

Anonymous authors
Paper under double-blind review

Abstract

The multi-LoRA agents framework, in which a collection of specialized agents share a single base LLM while each is equipped with its own LoRA adapter, has been widely adopted in complex real-world applications. In such pipelines, a downstream agent has to prefill the context produced by upstream agents before generating its own response. Because upstream agents typically accumulate long contexts, re-prefilling the entire context from scratch for every downstream agent incurs prohibitive computational overhead and inference latency. Yet we find that this repeated prefill is largely redundant: the KV cache of upstream agents can be substantially reused, requiring only lightweight sparse corrections. To this end, we propose MuLA (Multi-LoRA-Agents Sparse KV Cache Migration), a framework that efficiently reuses upstream KV caches by selectively recomputing only a small subset of critical token positions. Our empirical analysis shows that LoRA-switch-induced KV cache deviations are sparse over tokens and delayed to deep layers. Critically, cheap attention-concentration signals collected in the upstream agents forward pass strongly anticipate these error hotspots. Building on this insight, we exploit attention concentration as a cheap early indicator to proactively identify tokens requiring correction when switching LoRA adapters between agents, thereby completing sparse recomputation before the target deviations fully manifest, while balancing both accuracy and efficiency in the agents workflow. Results show that MuLA preserves nearly full-prefill quality with merely **10%∼25%** token recomputation, delivering up to **3.54**× ∼ **6.96**× downstream prefill speedup.

1 Introduction

LLMs are increasingly deployed as agents and multi-agent systems, in which multiple specialized agents often collaborate to accomplish complex tasks (Zhang et al., 2025; Han et al., 2025; Wang et al., 2025a). Existing multi-agent deployments can be categorized into two paradigms. The first paradigm uses a single powerful model to play different agent roles through role-specific prompts. The agents differ only in their instruction prompts while sharing exactly the same model parameters. This prompt-based design is flexible when the backbone model has strong instruction-following capabilities. However, resource constraints often require the deployment of smaller models, which may not reliably follow instructions and can consequently lead to degraded system robustness (Lu et al., 2025; Wang et al., 2024). The second paradigm trains models to acquire the capabilities required for different agent roles. This approach is necessary when the available model is too weak to maintain distinct agent behaviors reliably through prompting alone. Parameter-efficient fine-tuning, particularly LoRA (Hu et al., 2022), has become a common solution in this setting. Each agent is equipped with a lightweight LoRA adapter, while all agents share the same base model. This configuration has become an increasingly practical paradigm for role-specialized agents and multi-adapter serving (Liu et al., 2026a; Wang et al., 2025a; Sheng et al., 2024). Unlike the first paradigm, it introduces agent-specific model parameters, making KV cache reuse across agents fundamentally more challenging. MuLA is designed specifically for this multi-LoRA agent setting.

In the target scenario of MuLA, upstream agents commonly build long contexts with thousands of tokens. On one hand, naive reuse by downstream agents yields non-negligible performance degradation due to sparse LoRA-induced deviations in the cached KV states. On the other hand, full re-prefill repeats substantial computation and introduces a critical latency bottleneck. In fact, the upstream agent has already computed a complete KV cache for the shared context, and LoRA

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.

deviations are empirically small and sparse. These observations motivate KV cache reuse with minor corrections.

Comparing the two paradigms, it is relatively more challenging in the second paradigm to realize KV cache reuse via minor corrections. Deviations caused by prefixes arise only from changes in the prefix context and positions, and are therefore easier to predict and correct. In contrast, the central challenge of the multi-LoRA setting is that introducing LoRA adapter parameters directly changes the model's forward computation, injecting new deviations at every layer that accumulate and amplify with depth. Existing KV cache reuse techniques designed for prefix shifts and reuse (Kwon et al., 2023; Gim et al., 2024; Yao et al., 2026) cannot be directly applied to the multi-LoRA agent setting, where model parameters themselves vary across agents. LoRA-induced deviations span the entire network and are difficult to infer from shallow-layer deviation alone, making them harder to predict and correct.

To address the aforementioned difficulties, a series of analyses and experiments yield two findings that inform the design of our method:

1. **Prefix-induced and LoRA-induced deviations follow different propagation patterns.** Prefix-induced deviations typically follow a propagation pattern in which shallow-layer KV deviations reliably anticipate deep-layer error hotspots. By contrast, LoRA adapter switching continuously injects new representational discrepancies at every transformer layer. Critically, for LoRA-induced errors, shallow-layer KV deviations are themselves weak predictors of deep-layer deviation hotspots of the downstream agents.
2. **Attention concentration can serve as a cheap early predictor of LoRA-induced KV deviations.** We observe that tokens receiving high attention from the upstream agent model in shallow layers tend to exhibit larger LoRA-induced KV deviations in deeper layers of the downstream agents. This suggests that highly attended tokens are also more likely to be locations where the adapter modifies representations. Accordingly, the top-K positions selected by shallow-layer attention concentration from upstream agent can serve as the target set for downstream agents' KV correction.

Based on these findings, we propose MuLA (Multi-LoRA-Agent Sparse KV Cache Migration), which migrates an upstream agent's KV cache for reuse by a target agent by selectively recomputing only a small subset of critical token positions. MuLA uses observation attention score obtained from the upstream agent to identify tokens requiring correction when switching LoRA adapters between agents. The positions selected by MuLA are sparsely recomputed layer by layer using the target LoRA parameters. We evaluate MuLA across multiple agent communication tasks, including tool use, long-context question answering, and other representative multi-agent workloads. These tasks measure both end-to-end performance and work efficiency, allowing us to assess MuLA's applicability and efficiency in common agent collaboration scenarios. The results consistently show that sparse KV correction substantially reduces prefill computation while maintaining generation quality close to that of full target-agent prefill, demonstrating that MuLA generalizes across diverse tasks and varying numbers of chained agents.

Our contributions can be summarized as follows:

- **A systematic analysis of the properties of LoRA-induced KV deviations.** We analyze their hotspot locations, layer-wise propagation and show that LoRA injects new discrepancies at every layer that cannot be inferred from shallow-layer KV deviations alone. This explains why prefix-oriented correction methods do not directly transfer.
- **Uncovering upstream attention concentration as an efficient and prescient predictor for deviation.** Tokens with highly concentrated attention in shallow layers tend to exhibit larger LoRA-induced KV deviations in deeper layers, enabling attention from the upstream forward pass to select top-K correction positions.
- **Proposing a practical framework for sparse KV cache migration.** MuLA combines attention-guided token selection, target-LoRA layer-wise sparse recomputation, and onset caching. It reduces prefill computation while preserving quality close to full target prefill and generalizes across multi-LoRA agents paradigms.

2 Related Work

Multi-Agent Systems with Role-Specific LoRA Adapters. Role-specific LoRA adapters provide a parameter-efficient way to instantiate specialized agents over a shared backbone (Liu et al., 2026a; Lee et al., 2026; Yu et al., 2024; Shekar & Krishnan, 2025). VideoMind (Liu et al., 2026a) exemplifies this paradigm through a Chain-of-LoRA workflow, assigning different adapters to planning, temporal grounding, verification, and answering stages in long-video reasoning. Similar designs have been explored in code generation, retrieval and reasoning, role-playing and many other cooperative task decomposition (Lee et al., 2026; Yu et al., 2024; Shekar & Krishnan, 2025). Meanwhile, multi-LoRA serving systems enable efficient deployment, batching, scheduling, and switching of heterogeneous adapters over a common backbone (Chen et al., 2024; Sheng et al., 2024; Li et al., 2025). These studies establish shared-backbone multi-LoRA execution as a practical agentic paradigm; however, sequential LoRA agents generally reconstruct separate KV caches even when processing largely overlapping interaction histories.

KV Cache Reuse across Models and Agents. Existing studies have explored KV cache reuse, alignment, and communication across adapters, fine-tuned models, or collaborating agents (Wang et al., 2026a; Fu et al., 2026; Woo et al., 2026; Shi et al., 2026; Ye et al., 2025; Bian et al., 2026). aLoRA preserves compatibility with the base-model cache by activating the adapter only after designated invocation tokens (Greenewald et al., 2025); however, this restriction prevents adaptation of preceding representations and cannot transfer caches already generated under an active adapter. LRAgent (Jeon et al., 2026) decomposes KV states into shared base-model and adapter-dependent components to reduce redundant computation among LoRA agents. DroidSpeak instead identifies critical layers across related fine-tuned models and recomputes all tokens within those layers (Liu et al., 2026b), and RelayCaching reuses the upstream decoding cache with a fixed token subset at a representative layer (Geng et al., 2026), which may not adapt to dynamic deviations across layers. In contrast, MuLA directly adapts a materialized upstream LoRA cache by recomputing only a sparse subset of important tokens within each layer, achieving a good balance between generation quality and efficiency.

General Selective KV Cache Recomputation for Reuse. In addition to the KV cache reuse for the agent scenario mentioned above, there are also reuse strategies for other scenarios (such as RAG). In these additional scenarios, selective recomputation has been studied as an efficient approach to repairing approximate KV caches (Hu et al., 2025; Wang et al., 2025b; 2026b; Yan et al., 2026; Yang et al., 2025b). Most representatively, CacheBlend (Yao et al., 2026) targets retrieval-augmented generation, where independently precomputed document caches omit cross-chunk dependencies. It estimates cache deviations and selectively recomputes a small subset of tokens at each layer, demonstrating that approximate caches can be corrected without a full prefill. In contrast to their focus on prefix changes within a single model, our method targets parameter-induced discrepancies between LoRA agents and independently selects a layer-specific subset of important tokens at every layer, allowing the recomputation pattern to adapt to the propagation of cross-adapter deviations.

3 MuLA

In a multi-LoRA agent system, M agents share a single base LLM, each equipped with its own adapter, and context is handed off sequentially along the pipeline. At each hand-off, the downstream agent inherits a KV cache materialized under its predecessor's adapter; reusing it directly degrades generation quality. The root cause is that each projection matrix is $W^{(m)} = W^{\text{base}} + A^{(m)}B^{(m)}$, where $A^{(m)} \in \mathbb{R}^{d \times r_{\text{rank}}}$ and $B^{(m)} \in \mathbb{R}^{r_{\text{rank}} \times k}$ ($r_{\text{rank}} \ll d, k$). Whenever $A^{(m-1)}B^{(m-1)} \neq A^{(m)}B^{(m)}$, the K and V projections differ at every layer, and each layer amplifies these discrepancies with depth, so the inherited cache deviates from a full re-prefill under the target adapter. In this section, we provide a detailed account of our findings and methods.

3.1 Scenario Formulation

Let $\mathcal{M}$ denote the shared backbone LLM and $\Delta W^{(m)}$ the LoRA adapter deployed by agent $m \in \{1, \ldots, M\}$. We denote by $\mathcal{M}_{\Delta W^{(m)}}(X \mid \text{KV})$ the operation that computes the token sequence X with adapter $\Delta W^{(m)}$ conditioned on an existing cache KV. In a multi-LoRA agents sequential workflow, a task is collaboratively processed across M agents with clear division of labor. Agent 1

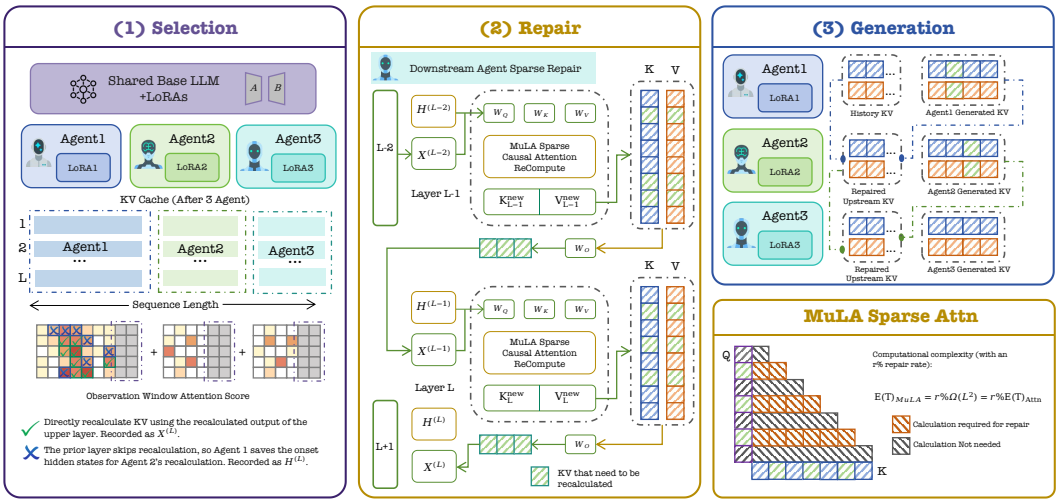


Figure 1: Overview of the MuLA sparse KV cache migration pipeline for multi LoRA agents inference. MuLA reuses the KV cache produced by upstream Agent and only performs sparse recomputation on a small set of critical tokens for downstream Agent.

receives an initial user prompt S_1 and generates its response D_1. Subsequently, each downstream agent $m \geq 2$ takes a role-specific transition prompt or instruction delta I_m, continues from the accumulated multi-turn conversation history, and decodes its contribution D_m. Let C_m be the cumulative conversation context prior to Agent m's generation:

$$C_1 = S_1, \qquad C_m = C_{m-1} \oplus D_{m-1} \oplus I_m \quad (m \geq 2), \tag{1}$$

where $\oplus$ denotes sequence concatenation, and $N_m = |C_m|$ denotes the cumulative context length.

Under the standard *full re-prefill* baseline, when execution transfers to Agent m, the disparity between adapter parameters ($\Delta W^{(m)} \neq \Delta W^{(m-1)}$) forces Agent m to recompute the entire historical context from scratch:

$$\mathrm{KV}_m^{*} = \mathcal{M}_{\Delta W^{(m)}}(C_m). \tag{2}$$

Across an M-agent pipeline, full re-prefilling incurs a prohibitive cumulative complexity of $O\big(\sum_{m=1}^{M} N_m^2\big)$, which quadratically compounds as the shared multi-turn history expands.

Conversely, *direct KV reuse* retains the materialized cache from upstream agents and only encodes the incremental delta I_m:

$$\mathrm{KV}_m^{\mathrm{reuse}} = \mathrm{KV}_{m-1}^{\mathrm{gen}} \ \oplus \ \mathcal{M}_{\Delta W^{(m)}}\big(I_m \mid \mathrm{KV}_{m-1}^{\mathrm{gen}}\big), \tag{3}$$

where $\mathrm{KV}_{m-1}^{\mathrm{gen}}$ contains the full cache accumulated up to D_{m-1}. Although direct reuse achieves optimal prefill efficiency, parameter heterogeneity across LoRAs introduces projection discrepancies at every layer, which self-attention progressively amplifies through deep layers and successive hand-offs, resulting in a degradation in the generation quality. This issue cannot be ignored.

MuLA Workflow. To optimize the expensive computational cost while preventing error cascading across multiple hand-offs, MuLA migrates the upstream cache via *sparse selective repair*. At the transition to Agent m, let $\mathcal{F}_l^m \subseteq \{1, \ldots, |C_{m-1} \oplus D_{m-1}|\}$ denote the cumulative union of critical token positions identified across historical hand-offs at layer l under a nominal budget ratio $r \in (0, 1]$. MuLA constructs the downstream execution cache KV'_m via a two-stage prefill:

$$\mathrm{KV}'_m = \underbrace{\mathcal{M}_{\Delta W^{(m)}}\big(I_m \mid \mathcal{R}(\mathrm{KV}_{m-1}^{\mathrm{gen}};\, \mathcal{F}_{1:l}^m,\, \Delta W^{(m)})\big)}_{\text{Delta Prefill: encode incremental delta } I_m} \cup \underbrace{\mathcal{R}\big(\mathrm{KV}_{m-1}^{\mathrm{gen}};\, \mathcal{F}_{1:l}^m,\, \Delta W^{(m)}\big)}_{\text{Sparse Selective Repair on history}}, \tag{4}$$

where $\mathcal{R}(\cdot; \mathcal{F}_{1:l}^m, \Delta W^{(m)})$ sparsely recomputes only the critical subset $\mathcal{F}_{1:l}^m$ using the target adapter $\Delta W^{(m)}$ while leaving unselected positions untouched. By restricting history recomputation from

$O(\sum_{m=1}^{M} N_m^2)$ to $O(\sum_{m=1}^{M} |\mathcal{F}^m| \times N_m) \approx O(r \sum_{m=1}^{M} N_m^2)$, MuLA bounds the multi-handoff computational cost to a modest fraction r while preserving generation fidelity across arbitrary M-agent cascades. Our goal is therefore to accelerate multi-agent workflows with as small a recompute ratio r as possible while preserving model performance.

3.2 Observation

In the same multi-agent setting formulated in Section 3.1, we examine the layer-wise dynamics of LoRA-induced KV deviations across two collaborating agents processing the same sequence. We compare their materialized KV caches layer by layer while recording the upstream agent's attention scores from its observe window over the historical sequence. The deviation at layer l and token position t is quantified via cosine distance:

$$\Delta\mathrm{KV}^{(l)}(t) = 1 - \cos\left\langle \mathrm{KV}_1^{(l)}(t),\ \mathrm{KV}_2^{(l)}(t) \right\rangle. \tag{5}$$

Key finding. Tracking upstream agent attention scores and ΔKV across layer depth reveals a pronounced layer-lag phenomenon: certain token positions exhibit sharp attention concentration already in shallow layers (e.g., layers 3∼5), yet their corresponding LoRA-induced KV deviations only become appreciable in substantially deeper layers (e.g., from layer 9 onward). Crucially, the shallow-layer attention top-K sets overlap heavily with the deep-layer ΔKV hotspots (Figure 2). This alignment also consistently holds across multi-turn hand-offs in a 5-agent cascade (Appendix G.3.1).

Why this happens. As analyzed in Appendix A, LoRA adapter switching injects subtle weight perturbations that yield negligible error in shallow layers, yet each layer progressively amplifies them across depth into deep-layer error cascades (Appendix Figure 4). Crucially, LoRA adaptation naturally tends to modify representations of tokens with high attention mass, as altering these pivotal positions produces the steepest gradient updates to redirect model outputs. As evidenced in Appendix C, the overlap between top-attention tokens and top-deviation tokens rises rapidly throughout LoRA fine-tuning. Hence, it is reasonable that early attention concentration correlates with downstream LoRA error hotspots.

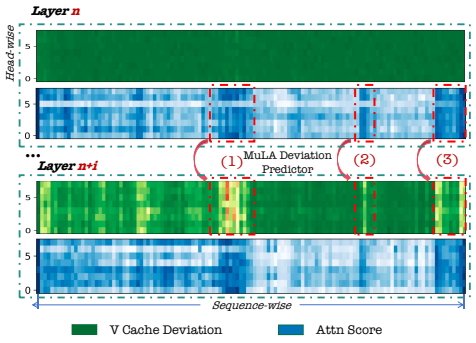


Figure 2: We compare the upstream-agent attention scores (blue heatmaps) from shallow layers and the LoRA-induced V-cache deviation (green heatmaps) in deeper layers. Red dashed boxes highlight three representative token positions: tokens with high attention scores in shallow layers evolve into high-deviation hot-spots in deeper layers.

Key insight. Full downstream prefill is prohibitive, while shallow KV mismatch fails to detect deep LoRA corruptions. Instead, MuLA exploits upstream shallow attention concentration as an ultra-cheap early predictor to identify critical tokens before deviations manifest, incurring near-zero selection cost (Appendix B). Such proactive identification enables repair to begin at shallow layers, preventing errors from accumulating into deep-layer cascades before it is too late to correct them.

3.3 Method

In an arbitrary hand-off from Agent $m-1$ to Agent m, the upstream agent has processed previous inputs and decoded its response, leaving a materialized KV cache. The downstream Agent m inherits this context and activates its dedicated LoRA adapter $\Delta W^{(m)}$. We aim to migrate the upstream KV cache with corrections applied only to a small fraction of critical tokens, matching the quality of full forward computation. This cuts prefill-recomputation overhead and improves multi-agent workflow efficiency. MuLA achieves this goal through a three-stage pipeline across each hand-off: Select, Repair, and Generate. For the pipeline overview, please refer to Figure 1.

3.3.1 SELECT: ATTENTION-GUIDED TOKEN REPAIR SELECTION

During the forward and decoding pass of the upstream Agent $m-1$, we compute the observe attention. Hidden states are buffered during upstream prefill and decoding; only those corresponding to onset tokens are retained after selection at the end of decoding. At layer l, let the key and value tensors be $\mathbf{K}^{(l)} \in \mathbb{R}^{H_{\text{kv}} \times N \times d}$, and $\mathbf{V}^{(l)} \in \mathbb{R}^{H_{\text{kv}} \times N \times d}$, where H_{attn} denotes the number of attention heads, H_{kv} the number of KV heads (under grouped-query attention $H_{\text{attn}} = m_{\text{gqa}} \cdot H_{\text{kv}}$), N the current sequence length, and d the head dimension.We take only the queries inside the recent observe window, $\mathbf{Q}_{\text{obs}}^{(l)} \in \mathbb{R}^{H_{\text{attn}} \times L_{\text{obs}} \times d}$ (with $L_{\text{obs}} \ll N$), and attend over its historical keys. For each attention head g and each query position q within the observe window, the attention weight on the j-th historical position is calculated as:

$$\mathbf{A}_{g,q,j}^{(l)} \;=\; \frac{\exp\!\left(\dfrac{1}{\sqrt{d}}\,\langle\, \mathbf{Q}_{g,\,q,\,:}^{(l)},\; \mathbf{K}_{\kappa(g),\,j,\,:}^{(l)} \,\rangle + \mathbf{M}_{q,j}\right)}{\displaystyle\sum_{t=1}^{N} \exp\!\left(\dfrac{1}{\sqrt{d}}\,\langle\, \mathbf{Q}_{g,\,q,\,:}^{(l)},\; \mathbf{K}_{\kappa(g),\,t,\,:}^{(l)} \,\rangle + \mathbf{M}_{q,t}\right)}, \tag{6}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product in d-dimensional space, $\kappa(g) = \lfloor g / m_{\text{gqa}} \rfloor$ maps attention head g to its corresponding KV head under GQA, and the causal mask $\mathbf{M}$ is defined as $\mathbf{M}_{q,t} = 0$ for $t \le q + s_j$ and $\mathbf{M}_{q,t} = -\infty$ for $t > q + s_j$, where s_j denotes the offset of the observe-window queries relative to the global positions. In our implementation, we maintain a queue to efficiently update the observation attention throughout the decoding process.

We then aggregate them along two dimensions (across the observe-window queries and across all attention heads) to obtain, for each historical position j, a compact attention signal $\mathbf{s}_j^{(l)}$:

$$\mathbf{s}_j^{(l)} \;=\; \frac{1}{|\mathcal{W}| \cdot H_{\text{attn}}} \sum_{q \in \mathcal{W}} \sum_{g=1}^{H_{\text{attn}}} \mathbf{A}_{g,q,j}^{(l)}, \tag{7}$$

where $\mathcal{W}$ is the set of query positions inside the observe window ($|\mathcal{W}| = L_{\text{obs}}$), and summing over all H_{attn} query heads, so that $\mathbf{s}^{(l)} \in \mathbb{R}^N$ and $\sum_j \mathbf{s}_j^{(l)} = 1$. Tokens inside the observe window are assigned the maximum score, so that recently processed positions are not overlooked.

To suppress stochastic fluctuations between neighboring tokens, we apply a sliding-average pooling of window size w along the position axis, yielding the final attention-concentration score:

$$\mathbf{a}_j^{(l)} \;=\; \frac{1}{w} \sum_{\tau=j-\lfloor w/2 \rfloor}^{j+\lceil w/2 \rceil - 1} \mathbf{s}_\tau^{(l)}, \tag{8}$$

So $\mathbf{a}^{(l)} \in \mathbb{R}^N$ and $\sum_j \mathbf{a}_j^{(l)} = 1$. At every layer l, Agent $m-1$ selects top-k positions $S_l^{(m-1)}$ exclusively from its *own segment* ΔC_{m-1} (comprising its incremental input I_{m-1} and generated response D_{m-1}, with length ΔN_{m-1}):

$$S_l^{(m-1)} \;=\; \Big\{\, i \in \Delta C_{m-1} \;\Big|\; a_i^{(l)} \;\ge\; a_{(k)}^{(l)} \Big\}, \tag{9}$$

where $k = \lceil r \cdot \Delta N_{m-1} \rceil$ is bounded by the budget ratio $r \in (0, 1]$, and $a_{(k)}^{(l)}$ denotes the k-th largest score within this segment. Historical selections made by earlier upstream agents are directly inherited without re-selection:

$$\mathcal{F}_l^{(m)} \;=\; \mathcal{F}_l^{(m-1)} \;\cup\; S_l^{(m-1)} \qquad \big(\text{with } \mathcal{F}_l^{(0)} = \varnothing \text{ and } m \ge 2\big). \tag{10}$$

This once-selected-always-repaired design confines attention scoring to each agent's local segment, avoiding redundant full-sequence scans while preserving an accumulated union of critical positions across hand-offs. Note that this selection draws exclusively on signals obtained from the upstream agents forward pass and requires zero additional forward computation for downstream agents. For more design considerations, please refer to Appendix D.

3.3.2 REPAIR: SPARSE KV RECOMPUTATION

Given the per-layer selected sets to repair $\{\mathcal{F}_l^{(m)}\}_{l=0}^{L-1}$, MuLA performs layer-wise sparse recomputation using the target Agent m's LoRA parameters. Let the inherited source KV cache at layer l be $\mathbf{K}^{(l)}, \mathbf{V}^{(l)} \in \mathbb{R}^{1\times H_{\text{kv}}\times N\times d}$, and let the recorded upstream hidden state be $\mathbf{H}^{(l)} \in \mathbb{R}^{1\times N\times d_{\text{model}}}$, where d_{model} is the model hidden dimension and each projection satisfies $\mathbf{W}_*^{\text{agent}_m} = \mathbf{W}_*^{\text{base}} + \mathbf{A}_*^m\mathbf{B}_*^m$ with $\mathbf{A}_*^m \in \mathbb{R}^{d_{\text{proj}}\times r_{\text{rank}}}$ and $\mathbf{B}_*^m \in \mathbb{R}^{r_{\text{rank}}\times d_{\text{model}}}$ ($r_{\text{rank}} \ll d_{\text{model}}, d_{\text{proj}}$).

Onset cache. Just as shown in the "selection" part of Figure 1, at layer l, every selected position $i \in \mathcal{F}_l^{(m)}$ requires an input hidden state, which can come from two sources: if i enters the selected set for the first time at layer l, the input is taken from the upstream recorded hidden state $\mathbf{H}^{(l)}$; if i was already selected and recomputed in the previous layer, the input is the carried-over intermediate recalculated representation $\mathbf{x}_i'^{(l-1)}$ from layer $l-1$. The boundary between the two cases is captured by the layer-wise onset set $O_l^{(m)}$:

$$\mathcal{F}_{-1}^{(m)} = \varnothing, \qquad O_l^{(m)} \;=\; \mathcal{F}_l^{(m)} \setminus \mathcal{F}_{l-1}^{(m)} \;=\; \mathcal{F}_l^{(m)} \cap \overline{\mathcal{F}_{l-1}^{(m)}}. \tag{11}$$

Hence

$$\mathbf{x}_i^{(l)} = \begin{cases} \mathbf{H}_i^{(l)}, & i \in O_l^{(m)}, \\ \mathbf{W}_l^{\text{agent}_m}\left(\mathbf{x}_i'^{(l-1)}\right), & i \in \mathcal{F}_l^{(m)} \cap \mathcal{F}_{l-1}^{(m)}. \end{cases} \tag{12}$$

Note that $\bigcup_l O_l^{(m)}$ defines the onset cache entries: tokens first entering the recomputation trajectory, and thus the only positions needing pre-stored prior hidden states. MuLA retains prior hidden states only for this onset set instead of all $L \times N$ positions. This reduces the recomputation-input footprint from $O(L \times N)$ to $\sum_l |O_l^{(m)}|$, rendering layer-wise sparse recomputation GPU-memory efficient; see Fig. 10. Here, $\mathbf{W}_l^{\text{agent}_m}(\cdot)$ denotes the layer-l parameters of downstream agent m, which recomputes the current representation from the recalculated hidden representation of the previous layer, as described in the following paragraph.

Recomputation. The recomputation at layer l follows the same parameters as the downstream agent, but is restricted to the selected positions $\mathcal{F}_l^{(m)}$. For each $i \in \mathcal{F}_l^{(m)}$, we apply query, key, and value projections using the target Agent$_m$ parameters, with Qwen3 as our running example:

$$\mathbf{q}_i = \mathbf{W}_Q^{\text{Agent}_m}\,\text{Norm}\big(\mathbf{x}_i^{(l)}\big), \quad \mathbf{k}_i = \mathbf{W}_K^{\text{Agent}_m}\,\text{Norm}\big(\mathbf{x}_i^{(l)}\big), \quad \mathbf{v}_i = \mathbf{W}_V^{\text{Agent}_m}\,\text{Norm}\big(\mathbf{x}_i^{(l)}\big). \tag{13}$$

After the projections are broadcast along the GQA axis, the recomputed keys and values are written back **in-place** into the corresponding positions of the KV cache:

$$\mathbf{K}^{(l)}_{\mathcal{F}_l^{(m)},:} \leftarrow \big[\,\mathbf{k}_i\,\big]_{i\in\mathcal{F}_l^{(m)}}, \qquad \mathbf{V}^{(l)}_{\mathcal{F}_l^{(m)},:} \leftarrow \big[\,\mathbf{v}_i\,\big]_{i\in\mathcal{F}_l^{(m)}}, \tag{14}$$

i.e., only the row slices indexed by the selected set are updated while all other positions in the cache remain untouched.

Causal prefix attention. Given the partially updated cache, each selected query attends causally over its prefix $[1{:}i]$:

$$\mathbf{u}_i \;=\; \sum_{t=1}^{i} \frac{\exp\Big(\frac{1}{\sqrt{d}}\,\langle\,\mathbf{q}_i,\;\mathbf{k}_t\,\rangle\Big)}{\sum_{s=1}^{i}\exp\Big(\frac{1}{\sqrt{d}}\,\langle\,\mathbf{q}_i,\;\mathbf{k}_s\,\rangle\Big)}\;\mathbf{v}_t, \tag{15}$$

where $\mathbf{k}_t, \mathbf{v}_t$ are the keys and values from the cache. The attention output is projected as $\mathbf{z}_i = \mathbf{W}_O^{\text{Agent}_m}\mathbf{u}_i$. We implement this attention computation using a custom Triton kernel to improve GPU memory utilization and computational efficiency; see B.3.

Feed-forward update. The transformer block then completes the residual connection and the feed-forward update:

$$\begin{aligned} \mathbf{h}_i^{(l)} &= \mathbf{x}_i^{(l)} + \mathbf{z}_i, \\ \mathbf{x}_i'^{(l)} &= \mathbf{h}_i^{(l)} + \mathbf{W}_{\text{down}}^{\text{Agent}_m}\,\sigma\big(\mathbf{W}_{\text{gate}}^{\text{Agent}_m}\mathbf{h}_i^{(l)} \odot \mathbf{W}_{\text{up}}^{\text{Agent}_m}\mathbf{h}_i^{(l)}\big), \end{aligned} \tag{16}$$

where σ denotes the activation function and $\odot$ denotes element-wise multiplication. The output $\mathbf{x}_i'^{(l)}$ becomes the carried-over hidden state for position i. After processing all L layers, the MuLA repair procedure is complete. MuLAs acceleration performance is analyzed in Appendix B.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

We evaluate MULA under three complementary hand-off settings of increasing complexity. (1) *Long-context one-shot migration*: a controlled two-agent hand-off on LongBench, following the setting of DroidSpeak (Liu et al., 2026b), where Agent 1 comprehends a long input and Agent 2 answers from the migrated cache—this isolates the minimal migration unit so that any quality gap stems solely from KV migration. (2) *Real-world two-agent workflow*: API-Bank tool-use dialogues, where Agent 1 analyzes the request and Agent 2 emits the API call from the migrated cache. (3) *Serial multi-agent migration*: a GSM8K five-agent pipeline where five role-specific adapters (solver, analyst, programmer, inspector, arbiter) hand off context successively, testing whether sparse repair survives compounding errors across multiple migrations.

All experiments use Qwen (Yang et al., 2025a) and Llama (Grattafiori et al., 2024) backbones with role-specific LoRA adapters, on NVIDIA A100 GPUs. Training recipes and hyperparameter ablation are detailed in Appendix E; latency and memory efficiency analyses are reported in Appendix B.

4.2 BENCHMARKS AND BASELINES

LongBench (Bai et al., 2024) is a long-context benchmark spanning QA, summarization, classification, retrieval, and code tasks, with contexts of up to tens of thousands of tokens. We use it for the one-shot migration setting, where Agent 1 prefill the long input and Agent 2 produces the final answer from the migrated cache, mirroring long-history information transfer between agents.

API-Bank (Li et al., 2023b) provides multi-turn tool-use dialogues in which agents must plan and invoke external APIs. We use it for the real-world workflow setting: Agent 1 analyzes the user request and extracts key information from the dialogue history, while Agent 2 selects the API interface and emits the corresponding invocation from the migrated cache.

GSM8K-5Agents instantiates the serial multiagent setting on GSM8K math problems, where five rolespecific LoRA agents, namely solver, analyst, programmer, inspector, and arbiter, collaborate over a shared backbone across four successive handoffs; see Appendix G.3 for construction details.

Baselines. We compare MULA against four cache-repair methods: CacheBlend (Yao et al., 2026), which selects tokens by estimated cache deviation; RelayCaching (Geng et al., 2026), which fixes a token set at a detection layer; DroidSpeak (Liu et al., 2026b), which recomputes whole critical layers; and LRAgent (Jeon et al., 2026), which shares a base-model cache with low-rank adapter components. *Full Prefill* recomputes the entire context under downstream Agent and serves as the quality upper bound; *Direct Reuse* hands upstream Agent's cache to downstream Agent without correction, quantifying the uncorrected mismatch. For more experiments, please refer to Appendix G.

4.3 MAIN RESULTS

LongBench. Table 1 reports migration results on LongBench subsets, averaging the 10% and 25% repair budgets for sparse methods. Direct reuse collapses on Qwen3-1.7B (25.43 $\rightarrow$ 6.14). MULA is the best repair method on both backbones, reaching 25.13 on Qwen3-1.7B and 27.97 on Llama3.2-3B, which correspond to 98.8% and 98.1% of Full Prefill. This result shows that a small, dynamically selected set of KV positions suffices to recover targetagent quality.

API-Bank. Figure 3 (left) sweeps the repair budget from 5% to 80% on Qwen3-1.7B. MULA approaches Full Prefill with a small ratio and can even surpass it; the sharp early gain and subsequent saturation indicate that repair quality depends on identifying influential KV positions rather than recomputing a large fraction of the cache.

GSM8K-5Agents. Figure 3 (right) reports end-to-end accuracy across hand-offs. Uncorrected reuse accumulates KV errors and degrades accuracy, while MULA approaches full-prefill accuracy at $10\% \sim 25\%$ budgets, confirming that attention-guided repair remains effective under repeated migrations.

Table 1: Main LongBench scores under LoRA KV migration, reported in %. Bold marks the best repair method for each subdataset and average, and underline marks the second-best.

Model	Method	LongBench Subset													Avg.
		NQA	Qasp.	MFQA	HotQA	2Wiki	Mus.	TREC	Trivia	SAM	Cnt.	Ret.	LCC	Repo	
(a) Qwen3-1.7B															
	Full Prefill	17.54	32.72	46.55	38.61	28.07	22.00	4.00	33.80	37.60	0.00	19.00	28.94	21.79	25.43
	Direct Reuse	4.28	6.44	22.01	6.56	9.27	1.79	1.00	2.25	2.43	0.67	1.50	10.79	10.82	6.14
	RelayCaching	16.63	**30.47**	45.17	39.44	27.50	20.40	**2.00**	32.03	32.12	0.50	**24.00**	**28.56**	18.53	24.41
	DroidSpeak	10.25	21.65	36.92	23.70	19.41	12.36	1.07	5.44	12.47	0.72	13.50	20.68	18.99	15.17
	CacheBlend	1.87	28.21	35.36	21.67	15.92	9.76	**2.00**	9.78	20.48	**1.25**	16.00	27.38	**28.05**	16.75
	LRAgent	9.35	14.58	29.73	16.22	16.18	11.65	1.00	3.57	5.68	0.70	7.00	19.38	16.54	11.66
	MuLA	**18.33**	29.55	**45.43**	**39.69**	**27.71**	**22.28**	1.50	**34.12**	**35.63**	0.00	23.00	28.37	21.04	**25.13**
(b) Llama3.2-3B															
	Full Prefill	20.24	20.90	39.33	13.53	13.79	7.96	39.17	11.38	30.68	3.00	63.00	56.07	51.67	28.52
	Direct Reuse	6.39	10.68	32.86	10.23	12.13	7.40	34.50	10.55	29.09	0.00	6.00	58.03	50.83	20.67
	RelayCaching	18.87	**20.73**	**39.47**	13.45	15.76	7.44	38.83	12.76	32.08	1.50	40.50	57.37	50.94	26.90
	DroidSpeak	10.06	10.49	34.72	11.44	13.73	7.61	34.50	10.53	**32.94**	0.00	6.46	**58.86**	50.80	21.70
	CacheBlend	13.97	10.51	36.04	11.23	**15.98**	**8.78**	34.00	10.58	24.86	1.00	6.50	57.77	50.48	21.67
	LRAgent	6.47	10.38	31.74	9.90	14.43	8.20	34.50	10.45	**32.94**	0.00	6.00	58.33	50.31	21.05
	MuLA	**19.32**	20.57	39.23	**13.90**	15.24	7.77	**39.25**	**12.93**	32.91	**2.50**	**50.50**	57.34	**52.20**	**27.97**

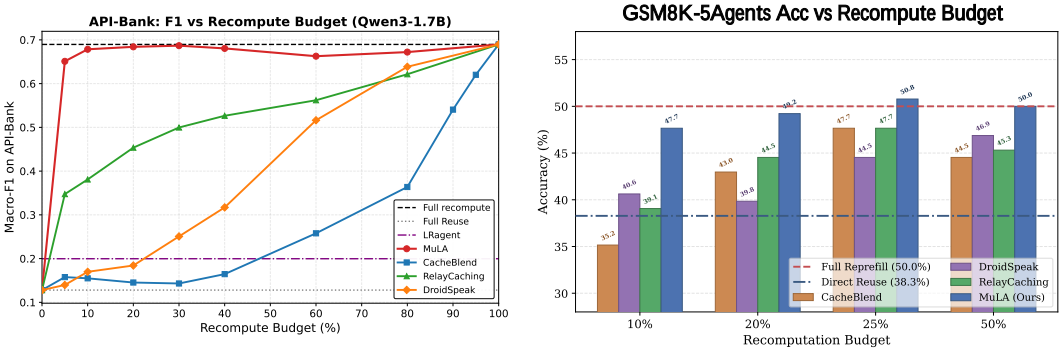


Figure 3: API-Bank Macro-F1 as a function of the KV recompute ratio for Qwen3-1.7B-Base (left) and GSM8K-5Agents accuracy across nominal recomputation budgets in the five-agent pipeline for Qwen3-1.7B-Base (right).

5 Conclusion

Multi-LoRA multi-agent systems enable lightweight role-specialized LLM agents to collaborate over a shared base model, yet sequential context hand-offs incur heavy redundant prefill overhead: downstream agents must recompute the full historical KV cache from scratch even when reusing context produced by upstream agents equipped with different LoRA adapters.

We present MuLA, a sparse KV-cache reuse framework for multi-LoRA agent workflows. MuLA leverages cheap attention-concentration signals collected during the upstream agents forward pass to proactively forecast tokens that will suffer large KV deviations under target-LoRA parameters, without running expensive forward computation on the downstream agent in advance. Following this attention-guided token selection stage, MuLA performs layer-adaptive sparse recomputation only on the identified critical token positions, utilizing an onset-cache mechanism to constrain hidden-state memory overhead. Extensive experiments on LongBench, API-Bank and GSM8K benchmarks demonstrate that MuLA recovers nearly full-prefill generation quality under modest recomputation budgets, yielding substantial inference speed-ups especially for long-sequence agent conversations. Taking LongBench as an example, with only **10%∼25%** token recomputation ratio, MuLA retains nearly full-prefill generation quality. For 20k+ length contexts, MuLA achieves around **3.54**× ∼ **6.96**× speedup on downstream prefill. Even under large LoRA-adapter divergence, MuLA maintains near-full-prefill performance, outperforming competing cache-repair baselines by considerable margins.

AI USE STATEMENT

In this work, generative AI tools were used exclusively for text polishing, grammatical correction, and syntax refactoring of LaTeX writing. In addition, an open-weights LLM (Qwen3.8-27B) was employed via vLLM solely as an offline teacher to synthesize multi-agent trajectory data for downstream GSM8K multi-agent training, as detailed in Section G.3. Generative AI was not used for conceiving core scientific hypotheses or interpreting experimental outcomes. All generated text, code, and analytical arguments have been thoroughly verified and audited by the authors.

ETHICS STATEMENT

This work investigates efficient KV cache reuse in multi-agent LLM systems, focusing purely on algorithmic design, inference latency optimization, and memory efficiency. All experiments rely on standard, publicly available research benchmarks (LongBench, API-Bank, RULER, and GSM8K; Section 4.2, Appendix G) and publicly released open-weights models (Qwen and Llama families; Section 4.1). No private, proprietary, personally identifiable, or human-subject data was gathered or used in this research. The proposed method introduces no specialized algorithmic biases or safety vulnerabilities beyond those inherent to the underlying pre-trained foundation models. We foresee no direct negative societal, ethical, or environmental impacts resulting from this work; rather, improving the inference efficiency of multi-agent LLMs helps reduce energy consumption and computational carbon footprints in long-context model serving (Appendix B).

REPRODUCIBILITY STATEMENT

To ensure full reproducibility, the end-to-end mathematical formulation of MuLA-encompassing observe attention aggregation, sliding-window smoothing, top-k selection, and layer-wise sparse recomputationis rigorously presented in Section 3, with custom Triton prefix attention detailed in Algorithm 1. Complete training recipes, model checkpoints, and inference hyperparameters are tabulated in Appendix E, while benchmark configurations and disaggregated results are reported in Section 4, Appendix G. Full source code of MuLA and custom GPU kernels will be released as open source upon publication.

REFERENCES

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. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

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 preprint arXiv:2604.03143*, 2026. URL https://arxiv.org/abs/2604.03143.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant lora serving. In *Proceedings of Machine Learning and Systems*, volume 6, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/054de805fcceb78a201f5e9d53c85908-Abstract-Conference.html.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Tianyu Fu, Zihan Min, Hanling Zhang, Jichao Yan, Guohao Dai, Wanli Ouyang, and Yu Wang. Cache-to-cache: Direct semantic communication between large language models. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/474ada926b331d78f06d95e8913111cc-Abstract-Conference.html.

Yingsheng Geng, Yuchong Gao, Weihong Wu, Guyue Liu, and Jiang Liu. Relaycaching: Accelerating llm collaboration via decoding kv cache reuse. *arXiv preprint arXiv:2603.13289*, 2026. URL https://arxiv.org/abs/2603.13289.

In Gim, Guojun Chen, Seung seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In *Proceedings of Machine Learning and Systems*, volume 6, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/a66caa1703fe34705a4368c3014c1966-Abstract-Conference.html.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Kristjan Greenewald, Luis Lastras, Thomas Parnell, Vraj Shah, Lucian Popa, Giulio Zizzo, Chulaka Gunasekara, Ambrish Rawat, and David Cox. Activated lora: Fine-tuned llms for intrinsics. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/4d0b6303d4a4811445f69f357bf6def5-Abstract-Conference.html.

Siwei Han, Peng Xia, Ruiyi Zhang, Tong Sun, Yun Li, Hongtu Zhu, and Huaxiu Yao. Mdocagent: A multi-modal multi-agent framework for document understanding. *arXiv preprint arXiv:2503.13964*, 2025. URL https://arxiv.org/abs/2503.13964.

Sirui Hong, Mingchen Zhuge, Jiaqi 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. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/6507b115562bb0a305f1958ccc87355a-Abstract-Conference.html.

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. URL https://arxiv.org/abs/2404.06654.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2106.09685.

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Zhang Qin, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. Epic: Efficient position-independent caching for serving large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 24391–24402, 2025. URL https://proceedings.mlr.press/v267/hu25j.html.

Hyesung Jeon, Hyeongju Ha, and Jae-Joon Kim. Lragent: Efficient kv cache sharing for multi-lora llm agents. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/61572.

Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018. URL https://aclanthology.org/Q18-1023/.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023. doi: 10.1145/3600006.3613165. URL https://dl.acm.org/doi/10.1145/3600006.3613165.

Woongkyu Lee, Junhee Cho, and Jungwook Choi. Mapcoder-lite: Distilling multi-agent coding into a single small llm. In *Findings of the Association for Computational Linguistics: EACL 2026*, 2026. URL https://aclanthology.org/2026.findings-eacl.346/.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. In *Advances in Neural Information Processing Systems*, volume 36, pp. 51991–52008, 2023a. URL https://proceedings.neurips.cc/paper/2023/hash/a3621ee907def47c1b952ade25c67698-Abstract-Conference.html.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-bank: A comprehensive benchmark for tool-augmented LLMs. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3102–3116, Singapore, December 2023b. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.187. URL https://aclanthology.org/2023.emnlp-main.187/.

Suyi Li, Hanfeng Lu, Tianyuan Wu, Minchen Yu, Qizhen Weng, Xusheng Chen, Yizhou Shan, Binhang Yuan, and Wei Wang. Toppings: Cpu-assisted, rank-aware adapter serving for llm inference. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, pp. 613–629, 2025. URL https://www.usenix.org/conference/atc25/presentation/li-suyi-toppings.

Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Yue Liu, Bo Li, Xuming Hu, and Xiaowen Chu. Chunkkv: Semantic-preserving kv cache compression for efficient long-context llm inference. In *Advances in Neural Information Processing Systems*, volume 38, pp. 28728–28778, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/2987f911151b39cd3a1761e212319e8e-Abstract-Conference.html.

Ye Liu, Kevin Qinghong Lin, Chang-Wen Chen, and Mike Zheng Shou. Videomind: A chain-of-lora agent for temporal-grounded video reasoning. In *International Conference on Learning Representations*, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/5e84e4413268b713f0d4a1b23a9dae57-Abstract-Conference.html.

Yuhan Liu, Yuyang Huang, Jiayi Yao, Shaoting Feng, Zhuohan Gu, Kuntai Du, Hanchen Li, Yihua Cheng, Junchen Jiang, Shan Lu, Madan Musuvathi, and Esha Choukse. Droidspeak: Kv cache sharing across fine-tuned model variants. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pp. 319–338, 2026b. URL https://www.usenix.org/conference/nsdi26/presentation/liu-yuhan.

Junru Lu, Jiazheng Li, Guodong Shen, Lin Gui, Siyu An, Yulan He, Di Yin, and Xing Sun. Rolemrc: A fine-grained composite benchmark for role-playing and instruction-following. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 21008–21030, 2025. doi: 10.18653/v1/2025.findings-acl.1082. URL https://aclanthology.org/2025.findings-acl.1082/.

Qwen Team. Qwen3.8-27B model card, 2026. URL https://huggingface.co/Qwen/Qwen3.8-27B.

Pavan C. Shekar and Aswanth Krishnan. Adaptive minds: Empowering agents with lora-as-tools. *arXiv preprint arXiv:2510.15416*, 2025. URL https://arxiv.org/abs/2510.15416.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. S-lora: Serving thousands of concurrent lora adapters. In *Proceedings of Machine Learning and Systems*, volume 6, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/906419cd502575b617cc489a1a696a67-Abstract-Conference.html.

Xiangyu Shi, Marco Chiesa, Gerald Q. Maguire Jr., and Dejan Kostic. Kvcomm: Enabling efficient llm communication through selective kv sharing. In *International Conference on Learning Representations*, 2026. URL https://iclr.cc/virtual/2026/poster/10010626.

Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Danning Ke, Shikuan Hong, Yiwu Yao, and Gongyi Wang. Razorattention: Efficient kv cache compression through retrieval heads. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/2a98af4fea6a24b73af7b588ca95f755-Abstract-Conference.html.

Oguzhan Topsakal and Tahir Cetin Akinci. Creating large language model applications utilizing langchain: A primer on developing llm apps fast. In *International conference on applied engineering and natural sciences*, volume 1, pp. 1050–1056, 2023. URL https://as-proceeding.com/index.php/icaens/article/view/1127.

Jialin Wang and Zhihua Duan. Agent ai with langgraph: A modular framework for enhancing machine translation using large language models. *arXiv preprint arXiv:2412.03801*, 2024. URL https://arxiv.org/abs/2412.03801.

Kangxu Wang, Ze Chen, Chengcheng Wei, Jiewen Zheng, Jiarong He, and Max Gao. Model fusion with multi-lora inference for tool-enhanced game dialogue agents. In *5th Wordplay: When Language Meets Games Workshop*, 2025a. URL https://wordplay-workshop.github.io/pdfs/19.pdf.

Qian Wang, Zahra Yousefijamarani, Morgan Lindsay Heisler, Rongzhi Gu, Bai Xiaolong, Shan Yizhou, Wei Zhang, Wang Lan, Ying Xiong, Yong Zhang, and Zhenan Fan. Mepic: Memory efficient position independent caching for llm serving. *arXiv preprint arXiv:2512.16822*, 2025b. URL https://arxiv.org/abs/2512.16822.

Shao Wang, Rui Ren, and Lin Gui. Forkkv: Scaling multi-lora agent serving via copy-on-write disaggregated kv cache. *arXiv preprint arXiv:2604.06370*, 2026a. URL https://arxiv.org/abs/2604.06370.

Shihao Wang, Jiahao Chen, Yanqi Pan, Hao Huang, Yichen Hao, Xiangyu Zou, Wen Xia, Wentao Zhang, Chongyang Qiu, and Pengfei Wang. Prophetkv: User-query-driven selective recomputation for efficient kv cache reuse in retrieval-augmented generation. *arXiv preprint arXiv:2602.02579*, 2026b. URL https://arxiv.org/abs/2602.02579.

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 257–279, 2024. doi: 10.18653/v1/2024.naacl-long.15. URL https://aclanthology.org/2024.naacl-long.15/.

Sunghyeon Woo, Hoseung Kim, Sunghwan Shim, Minjung Jo, Hyunjoon Jeong, Jeongtae Lee, Joonghoon Kim, Sungjae Lee, Baeseong Park, Se Jung Kwon, and Dongsoo Lee. Prefillshare: A shared prefill module for kv reuse in multi-llm disaggregated serving. *arXiv preprint arXiv:2602.12029*, 2026. URL https://arxiv.org/abs/2602.12029.

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 conversations. In *First Conference on Language Modeling*, 2023. URL https://arxiv.org/abs/2308.08155.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/5e5fd18f863cbe6d8ae392a93fd271c9-Abstract-Conference.html.

Jianxin Yan, Zeheng Qian, Wangze Ni, Zhitao Shen, Zhiping Wang, Haoyang Li, Jia Zhu, Lei Chen, and Kui Ren. Qcfuse: Query-centric cache fusion for efficient rag inference. *arXiv preprint arXiv:2604.08585*, 2026. URL https://arxiv.org/abs/2604.08585.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a. URL https://arxiv.org/abs/2505.09388.

Bin Yang, Qiuyu Leng, Jun Zeng, and Zhenhua Wu. Cacheclip: Accelerating rag with effective kv cache reuse. *arXiv preprint arXiv:2510.10129*, 2025b. URL https://arxiv.org/abs/2510.10129.

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. *ACM Transactions on Computer Systems*, 2026. doi: 10.1145/3790254. URL https://dl.acm.org/doi/10.1145/3790254.

Hancheng Ye, Zhengqi Gao, Mingyuan Ma, Qinsi Wang, Yuzhe Fu, Ming-Yu Chung, Yueqian Lin, Zhijian Liu, Jianyi Zhang, Danyang Zhuo, and Yiran Chen. Kv-comm: Online cross-context kv-cache communication for efficient llm-based multi-agent systems. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/1a074a28c3a6f2056562d00649ae6416-Abstract-Conference.html.

Xiaoyan Yu, Tongxu Luo, Yifan Wei, Fangyu Lei, Yiming Huang, Hao Peng, and Liehuang Zhu. Neeko: Leveraging dynamic lora for efficient multi-character role-playing agent. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024. URL https://aclanthology.org/2024.emnlp-main.697/.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 75834–75852, 2025. URL https://proceedings.mlr.press/v267/zhang25bi.html.

A Propagation Differences: Prefix-Induced versus LoRA-Parameter-Induced KV Deviations

We contrast two fundamentally distinct sources of KV-cache mismatch in LLM inference: *prefix-induced deviations*, which stem from misaligned cross-chunk attention when reusing independently-prefilled text chunks (typical for RAG workloads (Yao et al., 2026)), and *LoRA-parameter-induced deviations*, which occur when switching between different LoRA adapters over the same history sequence in multi-LoRA-agent pipelines. Their layer-wise behaviors are visualized in Figure 4, where mismatch is quantified as $1 -$ cosine similarity for K and V cache entries across transformer layers, averaged over 16 evaluation samples.

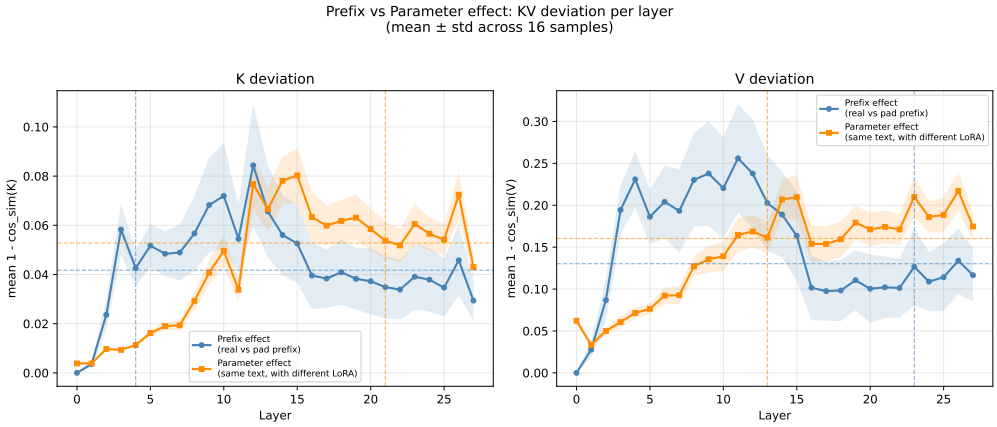


Figure 4: Per-layer KV cache deviation: prefix-effect versus LoRA-parameter-effect. Prefix-triggered deviations peak in shallow layers, whereas LoRA adapter switch introduces perturbations that grow along model depth. Shaded areas show sample-wise standard deviation.

In *prefix-induced deviations*, each chunks KV cache is pre-computed in isolation; when these pre-computed caches are directly concatenated in a new query context, they lack cross-chunk attention signals between chunks, giving rise to prefix-induced deviations. For this type of mismatch, substantial cache mismatch arises already within the shallow transformer layers, as shown by the blue line in Figure 4. *LoRA-parameter-induced deviations* follow a markedly different propagation trajectory.

Swapping LoRA adapters injects small representation perturbations at every single transformer layer, affecting Q/K/V projections as well as attention-output and MLP transformations. These per-layer perturbations do not naturally decay; instead, self-attention aggregates and magnifies discrepancies from preceding layers, as shown by the orange line in Figure 4. As a result, observable large-magnitude KV-cache dissimilarity only builds up in middle-to-deep layers. Shallow layers exhibit only modest deviation magnitudes, meaning shallow-layer KV states between source and target agent remain comparatively close. Consequently, deviation signals collected from early layers cannot reliably pinpoint the error hotspots that will appear later in the network.

This fundamental divergence in layer-wise propagation creates a clear barrier to directly porting existing prefix-oriented cache-repair methods to multi-LoRA-agent scenarios. Prior techniques rest on the working assumption that severe cache corruption manifests early in shallow layers, so shallow-layer observations are sufficient to locate most tokens that need repair. This assumption fails under LoRA adapter switching, where major deviation hotspots are deferred to deeper layers. Applying these existing approaches would either miss those critical faulty tokens, or force us to run the target LoRA forward pass to measure true deviations for token selection which defeats the core goal of cache reuse and creates a circular dependency. This motivates MuLAs core design principle: rather than explicitly measuring KV deviations using target-LoRA execution, we exploit low-cost attention-concentration signals harvested from the upstream agents forward pass to proactively forecast upcoming deep-layer deviation hotspots.

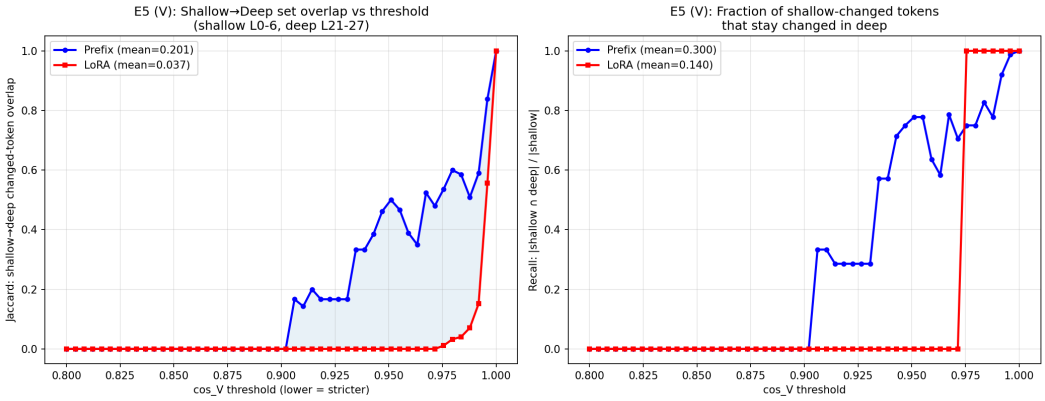


Figure 5: Token-level predictability of deep-layer KV errors from shallow-layer signals for prefix and LoRA-induced deviations on V cache. Left: Jaccard overlap between shallow (L0-6) and deep (L21-27) changed token sets over varying cosine thresholds. Right: Recall of shallow erroneous tokens persisting into deep layers. Prefix deviations exhibit substantially higher overlap and recall, showing shallow tokens reliably predict deep errors; LoRA deviations show weak shallow-to-deep token correspondence.

We further quantify the token-level predictive power of shallow-layer error signals in Figure 5. We select tokens as erroneous when their V-cache cosine similarity falls below a given threshold, and compute two metrics: the Jaccard overlap between shallow-error token set and deep-error token set, as well as the recall of shallow erroneous tokens that remain erroneous at deep layers.

For prefix-induced deviations, the Jaccard overlap and recall rise steadily as the threshold relaxes. A large portion of tokens identified as erroneous in shallow layers remain problematic in deep layers, confirming that shallow-layer observations can effectively locate deep-layer error hotspots. In contrast, LoRA-parameter-induced deviations yield near-zero Jaccard overlap and recall across most threshold values. Only at extremely loose thresholds do the values jump, which means tokens with large KV mismatch in shallow layers rarely correspond to the true deep-layer error tokens under LoRA switching. This token-level evidence reinforces our core argument: prior prefix-based repair methods, which select tokens using shallow-layer KV states, will fail to capture deep error hotspots in multi-LoRA agent handoff.

B EFFICIENCY ANALYSIS

B.1 THEORETICAL OVERHEAD & SPEEDUP OF MULA

We analyze the theoretical computation and memory overhead of MuLA, comparing against the full-prefill baseline for multi-LoRA agent handoff. Let N denote the length of shared historical context to-be-migrated, L the number of transformer layers, d_{model} the model hidden dimension, H_{kv} the number of KV heads, d the head dimension, r the recomputation budget ratio ($|F_l| \leq rN$), and L_{obs} the observation-window size for attention-concentration scoring, where $L_{\text{obs}} \ll N$. The newly-incoming prompt I_2 of downstream agent must perform full prefill for both MuLA and full-prefill baseline, whose cost cancels out in speedup comparison. MuLAs total computation consists of three components: select-stage attention-concentration scoring, layer-wise sparse repair on historical tokens, and mandatory full prefill for new prompt tokens.

Select-stage overhead. Instead of materializing the full $O(N^2)$ attention matrix, MuLA computes attentionconcentration scores only over queries inside the small observation window. The corresponding FLOPs are:

$$\text{FLOPs}_{\text{select}} = O\big(L_{\text{obs}} \cdot N \cdot H_{\text{kv}} \cdot d\big).$$

This is linear in N and avoids quadratic attention cost. Only aggregated scalar scores per token are stored, yielding a per-layer memory cost of $O(N)$. Nevertheless, this introduces a non-negligible constant-factor overhead, which may offset acceleration gains under extremely short-context scenarios. Beyond score computation, MuLA must select the K-th largest concentration scores per layer to form the repair set F_l. With $K \leq rN$, maintaining a size-K heap while scanning all N scores yields a per-layer cost of $O(N \log K)$, or equivalently $O(N \log rN)$; a full sort would give $O(N \log N)$. Across L layers, the total selection overhead is therefore

$$\text{FLOPs}_{\text{select,topk}} = O\big(L \cdot N \log(rN)\big).$$

Unlike the quadratic attention term, this scales linearly in N up to a logarithmic factor.

Repair-stage sparse recomputation overhead. For each layer, only tokens in the selected set F_l execute layer-norm, QKV projection, sparse causal attention, output projection, and feed-forward network. The projection and FFN costs scale linearly with the number of repaired tokens:

$$\text{FLOPs}_{\text{proj+ffn}} = O\big(L \cdot rN \cdot d_{\text{model}}^2\big).$$

For sparse causal attention, every repaired token attends causally over all preceding historical positions. The attention cost becomes $O(rN^2 H_{\text{kv}} d)$. Combining these terms, the total repair-stage FLOPs for historical context is:

$$\text{FLOPs}_{\text{repair}} = O\big(LrNd_{\text{model}}^2 + rN^2 H_{\text{kv}} d\big).$$

Different from layer-granular repair methods such as DroidSpeak, MuLA performs token-level sparse correction per layer. Note that MuLA does **not** eliminate quadratic complexity, but reduces the quadratic term coefficient by a factor of r.

Onset-cache memory overhead. Without optimization, sparse recomputation would require storing hidden states for all $L \times N$ positions. MuLA maintains onset set $O_l = F_l \setminus F_{l-1}$, which records tokens entering the repair set for the first time at layer l. Only hidden states of onset tokens are preserved. The overall memory footprint drops from

$$O(LNd_{\text{model}}) \quad \text{to} \quad \sum_l |O_l| \cdot d_{\text{model}}.$$

When repaired tokens exhibit layer-wise continuity,

$$\sum_l |O_l| \ll L \cdot rN.$$

In the worst case where repaired tokens are fully uncorrelated across layers, the memory benefit degenerates.

Asymptotic speedup analysis for long sequences. Combining the expressions derived above, the total FLOPs of MuLA can be written as:

$$\text{FLOPs}_{\text{MuLA}} = \text{FLOPs}_{\text{select}} + \text{FLOPs}_{\text{select,topk}} + \text{FLOPs}_{\text{repair}} + \text{FLOPs}_{\text{new}},$$

where

$$\begin{aligned}\text{FLOPs}_{\text{select}} &= O\big(L_{\text{obs}} N H_{\text{kv}} d\big),\\ \text{FLOPs}_{\text{topk}} &= O\big(LN \log(rN)\big),\\ \text{FLOPs}_{\text{repair}} &= O\big(LrN d_{\text{model}}^2 + rN^2 H_{\text{kv}} d\big),\\ \text{FLOPs}_{\text{new}} &= O\big(L|I_2| d_{\text{model}}^2 + |I_2|^2 H_{\text{kv}} d\big).\end{aligned}$$

For the full-prefill baseline, the total cost is

$$\text{FLOPs}_{\text{Full}} = O\Big(L(N+|I_2|) d_{\text{model}}^2 + (N+|I_2|)^2 H_{\text{kv}} d\Big).$$

Now consider the asymptotic regime of a very long historical context, i.e., $N \to \infty$, while keeping the new prompt length $|I_2|$, the repair ratio r, the number of layers L, the observation window L_{obs}, and all model dimensions $(d, d_{\text{model}}, H_{\text{kv}})$ constant. In this limit, lower-order terms become negligible:

$$|I_2| \ll N, \quad L_{\text{obs}} N \ll N^2, \quad N \log(rN) \ll N^2.$$

Thus, the dominant contributions are the quadratic attention terms:

$$\text{FLOPs}_{\text{MuLA}} \sim rN^2 H_{\text{kv}} d, \qquad \text{FLOPs}_{\text{Full}} \sim N^2 H_{\text{kv}} d.$$

Consequently, the theoretical FLOPs speedup factor of MuLA over the full prefill baseline is

$$\frac{\text{FLOPs}_{\text{Full}}}{\text{FLOPs}_{\text{MuLA}}} \sim \frac{N^2 H_{\text{kv}} d}{rN^2 H_{\text{kv}} d} = \frac{1}{r}.$$

This result shows that, for sufficiently long sequences, the speedup converges to a constant factor $1/r$, independent of N. Since r is the fraction of tokens selected for repair (typically $r \ll 1$, e.g., 0.050.1), the reduction in FLOPs is roughly an order of magnitude. In other words, MuLA effectively reduces the quadratic coefficient from 1 to r, achieving near-linear scaling in practice, though the actual wall-time improvement may also depend on memory bandwidth, kernel launch overhead, and the efficiency of the sparse attention implementation.

It is worth noting that when N is only moderately large, the logarithmic and linear terms may still contribute noticeably, and the speedup can be somewhat less than $1/r$. However, as N grows further, these sub-quadratic terms become increasingly negligible, and the acceleration approaches the ideal factor $1/r$.

B.2 Empirical Efficiency

In interactive multi-agent applications, token streaming makes the delay to the first token the primary user-visible latency. We therefore use time to first token (TTFT), defined as the wall-clock interval from the completion of the preceding agent's decoding to the emission of the successor agent's first token.We construct a five-agent pipeline in which Agent 1 receives an initial prompt of P tokens and every agent decodes B tokens. Each successor inherits the initial prompt and all preceding agents' outputs, so Agent i receives $P + (i-1)B$ tokens and then decodes B new tokens. We evaluate $P = B \in \{4\text{K}, 8\text{K}, 16\text{K}\}$, under which Agent i's input length is $i \times B$. Accordingly, Full includes history re-prefill and first-token decoding, whereas MuLA includes token selection, cache repair, and first-token decoding; Direct measures first-token decoding with the inherited cache unchanged.

MuLA's runtime pipeline operates as follows. During each upstream decoding step, MuLA incrementally accumulates the attention-concentration statistics using a deque and records the hidden states required by the onset cache; the associated reductions and bookkeeping are pipelined with upstream decoding. Consequently, when the upstream agent finishes decoding, the selector inputs and retained states are already available. The handoff TTFT clock starts at this boundary and includes all non-overlapped work on the handoff critical path—parallel top-k selection across layers, sparse cache repair, and the downstream agent's first-token decoding.

Figure 6 shows that Full re-prefill becomes increasingly expensive as either the per-agent output length or the number of preceding agents grows, while MuLA retains substantially lower TTFT across both model families. The advantage widens for later handoffs and longer histories: at the 16K Agent-5 handoff, MuLA with $r = 0.10$ reduces TTFT from 6.88 s to 1.07 s on Qwen3-1.7B and from

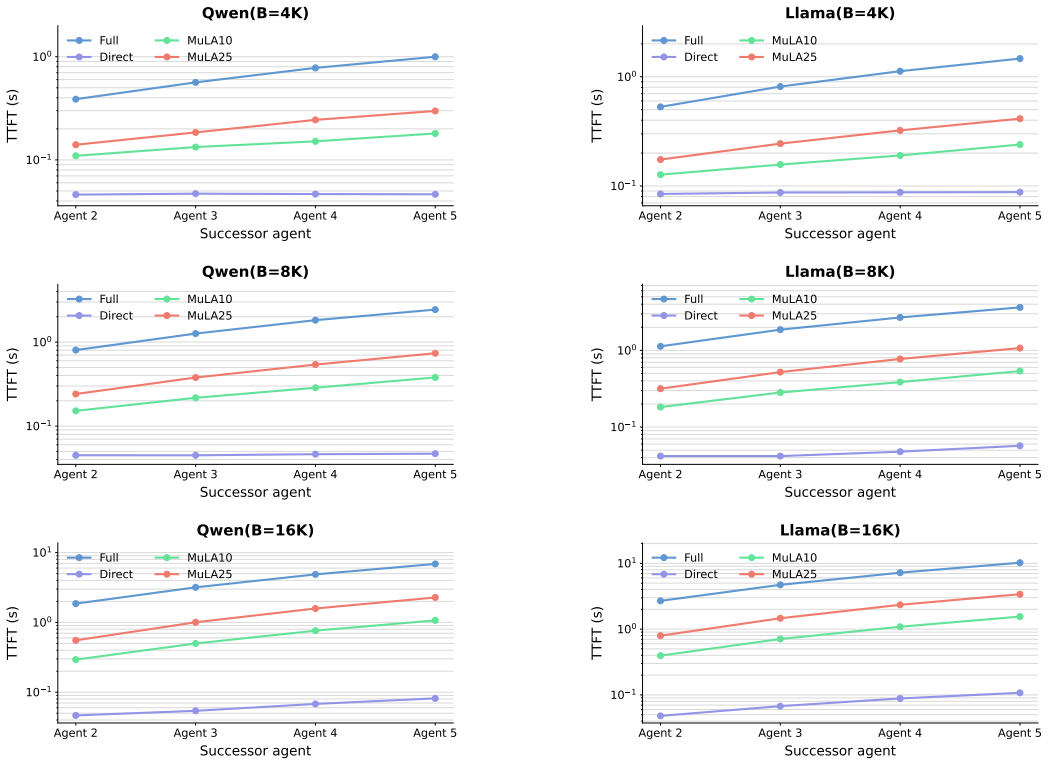


Figure 6: Handoff TTFT across successor agents on Qwen3-1.7B (left) and Llama-3.2-3B (right). The initial prefix and each agent's decoded output have length $P = B \in \{4\text{K}, 8\text{K}, 16\text{K}\}$, so Agent i receives $i \times B$ preceding tokens. TTFT is measured from the end of the preceding agent's decoding to the successor's first-token emission. Curves compare Full re-prefill, Direct cache reuse, and MuLA with repair ratios $r = 0.10$ and 0.25. The y-axis is logarithmic; lower is better.

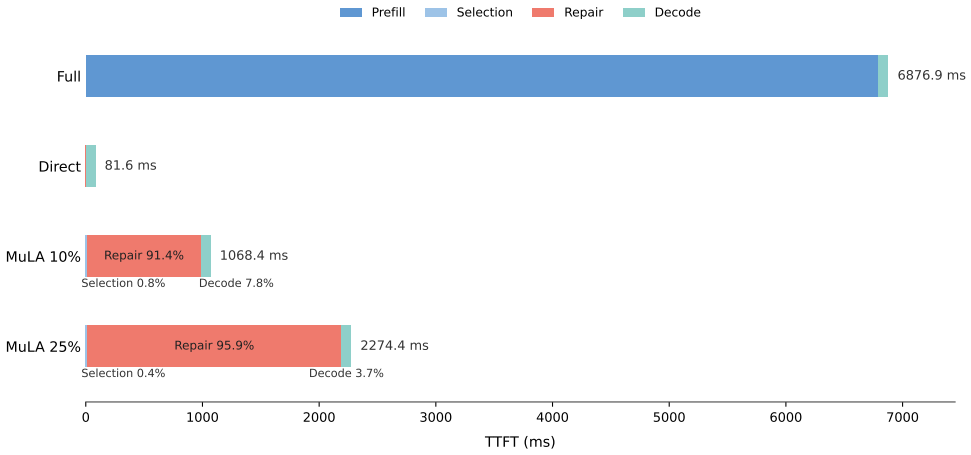


Figure 7: Qwen3-1.7B TTFT breakdown at the Agent-5 handoff with $P = B = 16\text{K}$, corresponding to $5 \times 16\text{K} = 80\text{K}$ preceding tokens. "Decode" denotes the successor's first-token decoding. Bars show TTFT in milliseconds; component labels indicate within-method percentages.

10.23 s to 1.55 s on Llama-3.2-3B, corresponding to a 6.4–6.6× speedup. A smaller repair ratio is consistently faster, whereas Direct provides a latency lower bound by performing no cache correction.

Figure 7 further decomposes the Qwen3-1.7B 16K Agent-5 handoff. The dominant cost of Full is history re-prefill, while MuLA replaces it with a much smaller repair cost; selection and first-token decoding contribute only a minor fraction of its TTFT. Increasing the repair ratio from 10% to 25% raises the repair time, explaining the corresponding TTFT increase.

In practice, the theoretical speedup of $1/r$ is rarely achieved, even with a small repair ratio such as $r = 0.1$. At shorter contexts, selection and repair overhead can dominate; moreover, the FLOP-based analysis does not capture hardware effects such as sparse-kernel utilization, non-contiguous memory access, and launch overhead. These factors keep measured wall-clock gains below the ideal FLOP reduction while preserving their increasing long-context trend.

B.3 Operator optimization implementation

To perform sparse recomputation for selected repair tokens, we require a specialized attention primitive: each repaired query token must attend to its own full causal prefix $[0, p_i]$, where p_i is the position of the repaired token. Queries have ragged, variable-length prefix boundaries, so standard prefill-style attention kernels cannot be directly reused. We implement a Triton-based sparse prefix attention kernel, summarized in Algorithm 1.

The kernel works on chunks of repaired queries. BLOCK_M denotes how many repaired query tokens are processed within one GPU thread-block, enabling parallelism over the set of tokens to-be-repaired. BLOCK_N is the tile size for iterating over the KV sequence: KV entries are streamed into SRAM in tiles of size BLOCK_N, rather than loading the entire sequence at once. An *online-softmax* algorithm incrementally maintains running maximum $\boldsymbol{m}$, running sum of exponentials $\boldsymbol{l}$, and value accumulator $\boldsymbol{A}$ across KV tiles. This avoids materializing the full $O(N^2)$ attention-score matrix and keeps SRAM consumption bounded. Grouped-query-attention (GQA) is handled inside each thread-block by mapping each query head to its corresponding KV head $h_{kv} = \lfloor h_q/G \rfloor$, without explicit KV repetition. For each query, scores for positions beyond its prefix boundary p_i are masked to $-\infty$ inside tile processing.

The kernel only iterates up to $S_{\text{eff}} = \max_i p_i + 1$, the maximum required prefix length for the current query chunk, skipping trailing KV entries that are never accessed. Above the kernel, a Python-level wrapper further splits large numbers of repaired queries via `query_chunk_size` to bound register pressure per kernel launch. When Triton/CUDA is unavailable, a reference fallback implementation invokes per-query scaled dot-product attention for correctness validation.

C Why Attention Concentration Predicts LoRA-Induced KV Deviations

To better understand why attention-concentration scores correlate with LoRA-induced KV deviations, we examine the token-set overlap dynamics throughout LoRA fine-tuning (left panel of Figure 8). We track both training loss and the intersection ratio between the top-30% highest-attention tokens (from the base model) and the top-30% tokens exhibiting largest KV deviations after LoRA adaptation. The overlap ratio rises rapidly in early optimization steps and gradually saturates as training proceeds, accompanied by fluctuating but decreasing training loss. This observation suggests that LoRA weight updates naturally tend to modify representations of tokens that receive high attention mass. Rather than being merely a spurious correlation observed at inference time, the linkage between high attention and large KV deviation originates from the fine-tuning process itself. This provides complementary empirical support for our attention-guided token-selection strategy.

To further validate the effectiveness of attention-driven token selection after LoRA convergence, we conduct controlled sparse-repair ablation experiments, shown in the right panel of Figure 8. Starting from the upstream pre-LoRA KV cache, we perform partial sparse repair with varying per-layer repair budgets, comparing two token-selection schemes: selecting tokens ranked by upstream attention concentration, and selecting random tokens as baseline. We evaluate reconstruction quality via the L2 distance between the top-64 logits produced by sparse-repaired cache and the reference logits from full target-LoRA prefill; smaller distance means better alignment with the gold-standard output.

Algorithm 1 Triton Kernel: MuLA Sparse Prefix Attention Forward

Input: Query chunk $Q_{\text{part}} \in \mathbb{R}^{B\times H_q\times K\times D}$, $K, V \in \mathbb{R}^{B\times H_{kv}\times S\times D}$, prefix boundary vector $\boldsymbol{p} \in \mathbb{N}^K$, scale factor $scale$; constants $\text{BLOCK}_M, \text{BLOCK}_N$, GQA group size $G = H_q/H_{kv}$.
Output: Partial attention output $\text{Out}_{\text{part}} \in \mathbb{R}^{B\times H_q\times K\times D}$.
{GQA head mapping for thread block}
For thread-block assigned batch index b, query head h_q, set $h_{kv} = \lfloor h_q/G \rfloor$.
Let $\mathcal{I} \subseteq \{1, \dots, K\}$ denote the set of query indices processed by this thread-block.
$Q_{\mathcal{I}} \triangleq Q_{\text{part}}[b, h_q, \mathcal{I}, :]$ {select queries for current block}
$\{p_i\}_{i\in\mathcal{I}} \triangleq \boldsymbol{p}[\mathcal{I}]$ {per-query prefix upper bound}
{Initialize online softmax accumulators for numerical stability}
$\boldsymbol{m}^{(0)} \leftarrow -\infty \cdot \mathbf{1}_{|\mathcal{I}|}, \quad \boldsymbol{l}^{(0)} \leftarrow \mathbf{0}_{|\mathcal{I}|}, \quad \boldsymbol{A}^{(0)} \leftarrow \mathbf{0}^{|\mathcal{I}|\times D}$
for $t_{\text{start}} = 0$ **to** S **step** BLOCK_N **do**
  $\mathcal{T} = \{t_{\text{start}}, t_{\text{start}} + 1, \dots, \min(t_{\text{start}} + \text{BLOCK}_N - 1, S - 1)\}$
  $K_{\mathcal{T}} \triangleq K[b, h_{kv}, \mathcal{T}, :], \quad V_{\mathcal{T}} \triangleq V[b, h_{kv}, \mathcal{T}, :]$
  Compute raw inner-product scores:

$$S_{i,t} = scale \cdot \langle Q_{\mathcal{I}}[i, \cdot],\, K_{\mathcal{T}}[t - t_{\text{start}}, \cdot] \rangle, \quad i \in \mathcal{I},\, t \in \mathcal{T}.$$

  Apply per-query prefix constraint:

$$\tilde{S}_{i,t} = \begin{cases} S_{i,t} & t \le p_i, \\ -\infty & t > p_i. \end{cases}$$

  $\boldsymbol{m}_{\text{block}} \leftarrow \max_{t\in\mathcal{T}} \tilde{S}_{i,t},\ \forall i \in \mathcal{I}$
  $\boldsymbol{m}_i^{(\text{new})} = \max\left(\boldsymbol{m}_i^{(\text{prev})}, (\boldsymbol{m}_{\text{block}})_i\right), \quad \forall i \in \mathcal{I}$
  $\alpha_i = \exp\left(\boldsymbol{m}_i^{(\text{prev})} - \boldsymbol{m}_i^{(\text{new})}\right)$
  $\tilde{P}_{i,t} = \exp\left(\tilde{S}_{i,t} - \boldsymbol{m}_i^{(\text{new})}\right), \quad i \in \mathcal{I}, t \in \mathcal{T}$
  $\boldsymbol{l}_i^{(\text{new})} = \alpha_i \cdot \boldsymbol{l}_i^{(\text{prev})} + \sum_{t\in\mathcal{T}} \tilde{P}_{i,t}, \quad \forall i \in \mathcal{I}$
  Update output accumulator:

$$\boldsymbol{A}_i^{(\text{new})} = \alpha_i \cdot \boldsymbol{A}_i^{(\text{prev})} + \sum_{t\in\mathcal{T}} \tilde{P}_{i,t} \cdot V_{\mathcal{T}}[t - t_{\text{start}}, \cdot], \quad \forall i \in \mathcal{I}.$$

  $\boldsymbol{m}^{(\text{prev})} \leftarrow \boldsymbol{m}^{(\text{new})},\ \boldsymbol{l}^{(\text{prev})} \leftarrow \boldsymbol{l}^{(\text{new})},\ \boldsymbol{A}^{(\text{prev})} \leftarrow \boldsymbol{A}^{(\text{new})}$
end for
Compute final output after normalizing accumulated weights:

$$\text{Out}_{\text{part}}[b, h_q, i, :] = \frac{\boldsymbol{A}_i^{(\text{prev})}}{\boldsymbol{l}_i^{(\text{prev})}}, \quad \forall i \in \mathcal{I}.$$

Write Out_{part} to global output tensor.

Repairing top-attention tokens achieves markedly lower logit reconstruction error across the entire budget range. This somewhat counter-intuitive result implies that attention-concentrated tokens exert larger influence over final output logits, justifying MuLAs choice to prioritize these positions for sparse correction.

D MuLA and Streaming KV Cache Compression: Connections, Distinctions, and Design Rationale

MuLA's per-agent local selection, where each agent scores only its own segment ΔC_{m-1} and contributes $S_l^{(m-1)}$ to a monotonically growing union $\mathcal{F}_l^{(m)} = \mathcal{F}_l^{(m-1)} \cup S_l^{(m-1)}$, bears a deliberate

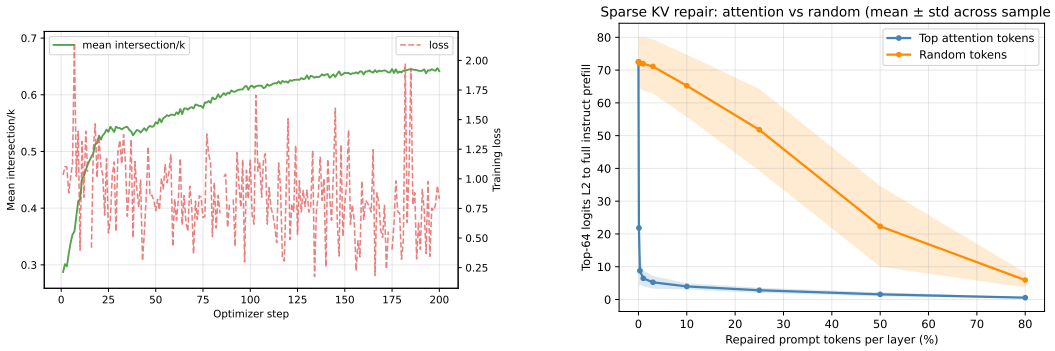


Figure 8: Left: Token-set overlap during LoRA fine-tuning. Green: intersection ratio between top-30% high-attention tokens and top-30% KV-deviation hot-spots; red dashed: training loss (right y-axis). Rising overlap shows LoRA preferentially modifies highly-attended tokens. Right: Logit-space reconstruction error for token-selection heuristics, measured by Top-64 logit L2 distance (lower is better) over repair budgets. The performance of repairing high-attention tokens is far superior to that of baseline random tokens. Collectively, both plots validate attention concentration as MuLAs effective token-selection predictor.

resemblance to streaming KV cache compression. We clarify here what is shared, what differs, and why this design is principled for multi-agent migration.

Similarities. Our per-agent design mirrors the chunk-by-chunk processing of streaming long-generation compression (e.g., StreamingLLM (Xiao et al., 2024) and ChunkKV (Liu et al., 2025)), as well as generation-time block-boundary pruning methods (e.g., RazorAttention (Tang et al., 2025)). In those streaming systems, generating or ingesting an unbounded stream cannot afford repeated full-sequence memory usage; instead, the stream accumulates KV state chunk by chunk, dynamic pruning or eviction is triggered at block boundaries, and only critical positions are persistently retained in the long-term cache alongside initial attention sinks and a local rolling window. MuLA adopts this exact structural philosophy across agent boundaries:

(i) **Segment-local scoring**: instead of rescanning the accumulated context C_m after every hand-off, each agent evaluates attention exclusively over its own segment ΔC_{m-1} using its trailing observation window; (ii) **Structural invariants**: the very first segment forcibly preserves the initial sink tokens to anchor attention sinks, while every segment unconditionally retains its final observation window tokens; (iii) **Monotonic retention**: selected critical positions enter a cumulative, non-decreasing union $\mathcal{F}_l^{(m)} = \mathcal{F}_l^{(m-1)} \cup S_l^{(m-1)}$, paralleling the permanent promotion policy of streaming chunked caches.

The rationale behind this structural resemblance is that in multi-agent pipelines, each agent's execution turn forms a natural, semantically coherent chunk. In streaming systems, arbitrary chunk boundaries are often artificially imposed on a continuous stream; in contrast, the turn boundaries $\Delta C_{m-1} = I_{m-1} \oplus D_{m-1}$ between agents correspond directly to distinct functional roles (e.g., plan, execute, inspect). Performing selection independently within each semantic chunk aligns directly with the core insight of recent streaming compression: local self-contained segments exhibit stable intra-chunk attention structures. MuLA leverages this intrinsic modularity by treating each agent's contribution as a self-contained unit for token selection, harmonizing the multi-agent workflow structure with modern chunked streaming principles.

Differences. The objectives are opposite. Streaming compression *discards* tokens to bound memory at the cost of approximate attention, whereas MuLA *preserves* the full sequence and *repairs* a small subset under the target adapter to restore fidelity. Concretely, an evicted token never participates in later attention, while a MuLA-selected token is recomputed through the full target block (QKV projection, causal prefix attention over $[0{:}p_i{+}1]$, FFN) and written back in place, with unselected positions untouched. Moreover, streaming assumes a single model and a single pass, while MuLA operates across adapter hand-offs: the scorer (upstream agent) and the beneficiary of repair (downstream agent) are different models, and the union accumulates across hand-offs rather than being decided once.

Table 2: LoRA training configurations for APIBank, LongBench, RULER, and GSM8K.

Setting	APIBank	LongBench	RULER	GSM8K
LoRA rank / scaling α	16/32	32/64	32/64	32/64
LoRA dropout	0.05	0.03	0.08	0.05

Note: Shared across all datasets: LoRA targets $\{W_q, W_k, W_v, W_o, W_{\text{gate}}, W_{\text{up}}, W_{\text{down}}\}$, learning rate 3×10^{-4}, 3 epochs, AdamW with cosine decay, gradient checkpointing enabled.

E EXPERIMENT SETTINGS AND DETAILS

We compare against deviation-based selection methods by extending CacheBlend (Yao et al., 2026) and RelayCaching (Geng et al., 2026) to our setting. We also include DroidSpeak (Liu et al., 2026b), which identifies critical KV-cache layers across fine-tuned model variants and selectively recomputes these layers while reusing the remaining cache. Finally, we compare with LRAgent (Jeon et al., 2026), which decomposes a Multi-LoRA agent cache into a shared base-model KV component and a LoRA-induced low-rank component, then shares the base cache while storing or reconstructing the adapter component in low-rank form.

For a fair comparison, we control the repaired KV-cache ratio across methods. Except for LRAgent, we evaluate each repair method with 10% and 25% KV-cache recomputation budgets over all KV cache. For LRAgent, we follow its original setting and apply the full KV-cache update described in the original paper.

E.1 Training Setting

For LongBench and RULER, we train two role-specific LoRA adapters for the hand-off workflow: a comprehension-and-extraction adapter that identifies the evidence required by the downstream agent, and an answer-generation adapter that produces the final response from the transferred context. The LongBench roles are trained with NarrativeQA (Kočiský et al., 2018), while the RULER agents are trained using a 1k-item train set synthesized from various sub-tasks of the RULER official code (Hsieh et al., 2024). For API-Bank, the first agent analyzes the dialogue, selects the appropriate API, and generates its arguments; the second agent converts this decision into the API invocation instruction. We use Qwen3.8-27B to generate analysis targets for the API-Bank training examples and use these targets to supervise the analysis agent. For GSM8K, we use the same model served through vLLM to generate training trajectories for five distinct agent roles, each of which is used to train its corresponding adapter (Section G.3). All the above LoRA fine-tuning is conducted via supervised fine-tuning (SFT) with standard next-token cross-entropy loss. Table 2 summarizes the LoRA training configurations.

E.2 Validation and Hyperparameter Selection

For each benchmark, we hold out a subset of the training data as a validation set and select the best MuLA hyperparameters on this split. Under the same benchmark-specific protocol, we perform offline analysis for CacheBlend, DroidSpeak, and RelayCaching to determine their critical repair layers before test-time evaluation. The LongBench configurations used for Qwen and Llama are reported in Table 3. For MuLA, L_{obs} is the number of recent query positions used to compute attention scores, and w is the sliding-average pooling width used to smooth these scores along the token dimension. For RelayCaching, r denotes the starting layer for recomputation from cached hidden states, d the detection layer where tokens are selected, and k the final layer of the repair range. We set k to the model's last transformer layer for both backbones. Following RelayCaching's combination of influence- and deviation-based selection, we allocate the token-selection budget between these two criteria in a 2:8 ratio. For CacheBlend, l denotes the layer at which cache deviations are measured to select tokens for subsequent sparse recomputation. For DroidSpeak, L denotes the inclusive interval of layers whose KV caches are fully recomputed under the specified budget.

Table 3: LongBench hyperparameters for Qwen and Llama.

Method	Parameter	Qwen	Llama
MuLA	L_{obs}	4	6
	w	31	7
RelayCaching	r	1	1
	d	1	1
	k	28	28
CacheBlend	l	1	1
DroidSpeak (10%)	L	$[15, 17]$	$[6, 8]$
DroidSpeak (25%)	L	$[16, 22]$	$[17, 23]$

E.3 DISAGGREGATED LONGBENCH RESULTS

Table 4 disaggregates the results summarized in Table 1 and reports the two evaluation budgets separately on the same LongBench subsets.

Table 4: Disaggregated LongBench results for Qwen3-1.7B-Base and Llama-3.2-3B, reported in %. Avg. is the arithmetic mean over the displayed subsets.

Method	**Budget**	**LongBench Subset**													**Avg.**
		NQA	Qasp.	MFQA	HotQA	2Wiki	Mus.	TREC	Trivia	SAM	Cnt.	Ret.	LCC	Repo	
(a) Qwen															
Full Prefill	–	17.54	32.72	46.55	38.61	28.07	22.00	4.00	33.80	37.60	0.00	19.00	28.94	21.79	25.43
Direct Reuse	–	4.28	6.44	22.01	6.56	9.27	1.79	1.00	2.25	2.43	0.67	1.50	10.79	10.82	6.14
RelayCaching	10%	17.08	29.72	44.48	39.74	26.49	20.36	2.00	29.18	27.62	1.00	25.00	28.42	18.23	23.79
RelayCaching	25%	16.18	31.22	45.86	39.14	28.50	20.43	2.00	34.88	36.62	0.00	23.00	28.70	18.83	25.03
DroidSpeak	10%	8.51	14.30	32.56	18.64	18.19	11.01	1.00	3.72	8.09	1.18	10.00	19.28	16.28	12.52
DroidSpeak	25%	11.98	28.99	41.28	28.76	20.62	13.71	1.14	7.15	16.84	0.25	17.00	22.08	21.69	17.81
CacheBlend	10%	1.34	20.24	25.29	4.89	3.94	0.41	1.00	4.71	17.10	1.50	2.00	23.33	26.84	10.20
CacheBlend	25%	2.40	36.17	45.42	38.44	27.90	19.10	3.00	14.84	23.86	1.00	30.00	31.42	29.26	23.29
LRAgent	–	9.35	14.58	29.73	16.22	16.18	11.65	1.00	3.57	5.68	0.70	7.00	19.38	16.54	11.66
MuLA	10%	18.29	28.90	45.35	40.13	27.32	22.72	2.00	34.37	35.20	0.00	25.00	26.94	21.38	**25.20**
MuLA	25%	18.36	30.19	45.51	39.24	28.09	21.84	1.00	33.86	36.05	0.00	21.00	29.79	20.70	**25.05**
(b) Llama															
Full Prefill	–	20.24	20.90	39.33	13.53	13.79	7.96	39.17	11.38	30.68	3.00	63.00	56.07	51.67	28.52
Direct Reuse	–	6.39	10.68	32.86	10.23	12.13	7.40	34.50	10.55	29.09	0.00	6.00	58.03	50.83	20.67
RelayCaching	10%	18.56	20.68	39.75	13.22	16.58	7.61	39.33	14.33	32.53	0.00	33.00	58.20	50.96	26.52
RelayCaching	25%	19.17	20.77	39.18	13.67	14.94	7.27	38.33	11.18	31.62	3.00	48.00	56.54	50.91	27.28
DroidSpeak	10%	5.72	10.24	33.55	9.89	13.91	7.59	34.50	10.53	33.11	0.00	7.25	58.60	50.55	21.19
DroidSpeak	25%	14.40	10.73	35.89	12.99	13.55	7.62	34.50	10.52	32.76	0.00	5.67	59.12	51.05	22.22
CacheBlend	10%	11.66	10.12	33.31	10.33	14.83	8.11	34.50	10.59	24.10	1.00	6.00	57.72	49.54	20.91
CacheBlend	25%	16.27	10.89	38.77	12.12	17.13	9.45	33.50	10.56	25.61	1.00	7.00	57.82	51.42	22.43
LRAgent	–	6.47	10.38	31.74	9.90	14.43	8.20	34.50	10.45	32.94	0.00	6.00	58.33	50.31	21.05
MuLA	10%	19.27	21.03	39.31	14.31	15.80	7.73	39.33	14.15	33.63	2.00	49.00	57.08	51.25	**27.99**
MuLA	25%	19.36	20.10	39.15	13.49	14.67	7.81	39.17	11.70	32.18	3.00	52.00	57.59	53.15	**27.95**

LRAgent performs strongly in its original setting, where LoRA updates are confined to the Q/V projections and its decomposition into a shared base-model cache and a low-rank adapter component closely matches the adapter structure. In our setting, however, the role-specific LoRAs also modify the output projection and the MLP projections, so the resulting hidden-state changes cannot be fully represented by this restricted decomposition. RelayCaching also allocates part of its repair budget to identifying and repairing important tokens, which makes it competitive on LongBench, where long contexts tend to contain sparse relevant evidence. However, only part of its budget targets these important tokens, and the selected token set remains fixed across subsequent layers. These constraints limit its ability to adapt to layer-dependent changes and lead to weaker performance on API-Bank and GSM8K than MuLA. These results indicate that reliable cache migration must account for adapter effects throughout the transformer block and adjust its repair targets across layers, as MuLA does.

F Ablation Analysis

To isolate the contributions of MuLA's selection strategy, we remove layer-wise dynamic selection by fixing the token set from its onset layer, and remove attention guidance by replacing attention-based ranking with uniform random selection while retaining the same sparse-recomputation pipeline. As summarized in Table 5, the fixed-set ablation confirms that repair targets must adapt across layers as LoRA-induced deviations propagate through depth, while the random-selection ablation shows that sparse recomputation is effective only when attention concentration identifies the influential token positions to repair.

Table 5: Ablation of layer-wise dynamic selection and attention-guided token selection on API-Bank using Qwen3-1.7B-Base. Results are parameter-level Macro-F1.

Method	10% budget	25% budget
Full Prefill	0.6890	
MuLA	**0.6799**	**0.6840**
w/o Dynamic Selection (Fixed-onset)	0.5272	0.6478
w/o Attention Selection (Random)	0.1280	0.1200
Direct Reuse	0.1280	

We perform ablation studies on two core hyper-parameters of MuLA: the smoothing kernel size and the attention observation window size. All experiments are conducted on the API-Bank dataset using Qwen3-1.7B-Base, with a fixed 25% recomputation budget and average-pool token scoring. The full-prefill setup serves as the upper-bound reference, achieving Macro-F1 $= 0.6890$.

Table 6 shows that performance is relatively stable across a broad range of smoothing kernels and observation windows. An overly narrow window provides insufficient attention evidence for reliable token selection, whereas enlarging either parameter beyond the effective range yields no consistent benefit. The best setting remains close to full-prefill performance on both Macro-F1 and exact match, indicating that MuLA is not highly sensitive to precise hyper-parameter choices.

Table 6: Hyper-parameter sweep under a fixed 25% recompute budget (avg-pool, API-Bank, Qwen3-1.7B). Prec. = Precision, Rec. = Recall, Macro-F1: macro averaged F1 on API parameter matching. EM = Exact Match Rate. The best configuration is highlighted with gray background and bold values.

Best / Fixed kernel $= 25$					*Reference / Fixed window* $= 32$				
k/w	Prec.	Rec.	Macro-F1	EM	Method / k/w	Prec.	Rec.	Macro-F1	EM
27/28	0.6812	0.6914	**0.6840**	**0.5696**	Full-prefill	–	–	0.6890	0.5780
25/24	0.6694	0.6793	0.6722	0.5634	21/32	0.6701	0.6807	0.6730	0.5613
25/26	0.6733	0.6845	0.6765	0.5655	27/32	0.6733	0.6835	0.6758	0.5613
25/28	0.6781	0.6890	0.6811	0.5676	29/32	0.6778	0.6880	0.6803	0.5676
25/30	0.6788	0.6876	0.6809	0.5676	41/32	0.6689	0.6786	0.6713	0.5593
25/32	0.6778	0.6869	0.6800	0.5696					

G More Experiments

G.1 LoRA Divergence Sweep

In this section, we vary the initialization schemes and learning rates of the source and target LoRA adapters to control their discrepancy and measure how the resulting cache-migration difficulty affects each method. Figure 9 defines LoRA divergence as the percentage-point (pp) difference between Full Prefill and Direct Reuse, and reports the corresponding pp gap between Full Prefill and each repair method; thus, a curve closer to zero indicates more effective recovery. All partial-recompute methods use a fixed 25% repair budget, while DroidSpeak is given its best 25% layer window for

each configuration and LRAgent uses its base-shared configuration. When the two adapters are nearly identical, all methods remain close to Full Prefill because the reused cache is already compatible with the target model. As divergence increases, however, the gaps of CacheBlend, DroidSpeak, and LRAgent grow rapidly and largely follow the degradation of Direct Reuse, indicating that their repaired states do not sufficiently correct the adapter-induced mismatch. RelayCaching is more robust and recovers a substantial portion of the lost quality, but its gap still reaches about 20.6 pp at the largest divergence, while CacheBlend, DroidSpeak, and LRAgent remain roughly 52–59 pp below Full Prefill. In contrast, MuLA keeps the gap near zero throughout the sweep: it exceeds Full Prefill by as much as 1.6 pp at intermediate divergence and is only 0.5 pp below Full Prefill at the largest divergence. These results show that, under the same limited repair budget, MuLA's advantage becomes more pronounced as the source–target LoRA discrepancy grows, demonstrating that its token- and layer-adaptive repair targets the cache entries most sensitive to model adaptation.

G.2 RULER TEST

We evaluate MuLA on the RULER benchmark (Hsieh et al., 2024), a comprehensive long-context evaluation suite designed to rigorously measure the long-context reasoning and retrieval capacity of large language models. RULER comprises four families of challenging long-context tasks: needle-in-haystack retrieval, multi-hop variable tracking, information aggregation, and open-ended question answering. Each task family targets distinct capabilities that become increasingly difficult for models as context windows expand: locating sparse embedded facts, maintaining consistent bindings for variables across long passages, summarizing and combining scattered pieces of evidence, and answering open questions grounded in extensive source material. All tasks can be procedurally generated with fully configurable context lengths ranging from several thousand to tens of thousands of tokens. This flexible setup makes RULER well-suited for stress-testing KV cache reuse strategies under lengthy multi-agent conversation histories, where accumulated context grows rapidly and accurate preservation of earlier information across sequential agent hand-offs is critical.

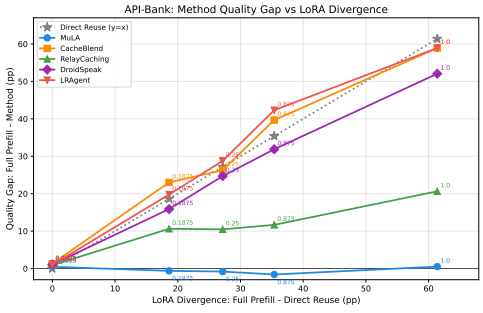


Figure 9: Quality gap to Full Prefill, measured in percentage points (pp), as LoRA divergence increases. Partial-recompute methods use a 25% repair budget; lower is better.

We adapt the original single-model evaluation protocol to simulate a two-agent hand-off scenario for multi-LoRA systems. Both agents share the same base LLM backbone but are fine-tuned with distinct role-specific LoRA adapters:

- **Agent 1 (search-oriented agent)**: processes the full long-context input, performs comprehension and intermediate reasoning, and collects all the relevant contextual clues related to answering the questions.
- **Agent 2 (answer-oriented agent)**: takes over the contextual clues produced by Agent 1 to generate final task outputs. Instead of executing expensive full prefill from scratch, Agent 2 reuses Agent 1s KV cache and applies MuLA to selectively repair corrupted cache entries.

This two-agent setup mimics realistic multi-agent workflows: one agent ingests and processes long raw context, then hands off its cached KV state to a second specialized agent for answer synthesis. Directly re-using Agent 1s unmodified KV cache leads to severe performance drops, caused by representation discrepancies introduced by LoRA fine-tuning between the two agents. We report per-task accuracy across all four RULER task categories.

As shown in Table 7, direct KV reuse suffers severe accuracy degradation across most challenging long-context tasks. Even under 25% recomputation budget, MuLA stays nearly on par with or slightly surpasses the full-prefill upper-bound.

Table 7: MuLA accuracy on RULER8K full-test for Qwen3-0.6B-Base and Llama-3.2-3B.

	Qwen3-0.6B-Base				Llama-3.2-3B			
Task	F	D	10%	25%	F	D	10%	25%
CWE								
CWE	71.2	4.0	68.8	69.6	92.8	0.0	93.6	93.6
CWE-hard	48.0	0.0	45.6	45.6	82.4	0.0	81.6	81.6
Counting & Factuality								
Entity cnt. (1)	99.2	97.6	99.2	99.2	100.0	97.6	100.0	100.0
Entity cnt. (2)	100.0	100.0	99.2	99.2	100.0	98.4	100.0	100.0
Fact hop (1)	100.0	52.0	98.4	98.4	64.0	48.0	64.8	66.4
Fact hop (2)	97.6	60.0	98.4	98.4	69.6	43.2	61.6	71.2
FWE	98.4	64.0	97.6	98.4	100.0	98.4	100.0	100.0
FWE-hard	100.0	80.8	100.0	100.0	99.2	97.6	99.2	99.2
NIAH								
NIAH multi-key (1)	99.2	67.2	99.2	98.4	99.2	94.4	99.2	99.2
NIAH multi-query	100.0	84.8	100.0	99.2	100.0	85.6	100.0	100.0
NIAH multi-value	99.2	93.6	100.0	100.0	100.0	87.2	100.0	100.0
NIAH needle-noise	62.4	52.8	63.2	63.2	72.0	74.4	72.8	72.8
NIAH single (1)	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
NIAH single (2)	100.0	97.6	100.0	100.0	100.0	100.0	100.0	100.0
Numerical Aggregation								
Num. agg. (1)	2.4	0.0	0.8	2.4	3.2	0.0	2.4	1.6
Num. agg. (2)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Question Answering								
QA Hotpot	44.8	37.6	41.6	43.2	54.4	52.8	56.0	56.0
QA SQuAD	49.6	44.8	45.6	48.0	56.0	55.2	52.8	53.6
Temporal & Variable Tracking								
Temporal (1)	96.8	69.6	96.0	96.0	98.4	97.6	97.6	98.4
Temporal (2)	97.6	72.0	96.0	96.0	98.4	97.6	98.4	99.2
VT (1)	99.2	20.8	99.2	98.4	98.4	40.0	99.2	99.2
VT (2)	97.6	8.8	97.6	97.6	97.6	21.6	99.2	99.2
VT (3)	97.6	5.6	98.4	97.6	96.0	18.4	96.8	99.2

F = Full-prefill accuracy, D = Direct-reuse accuracy.
10% / 25% = MuLA accuracy at the respective repair budget.

G.3 SERIAL MULTI-HANDOFF EVALUATION WITH A FIVE-AGENT PIPELINE

In this section, we construct a multi-agent training and test suite for GSM8K by extending the CAMEL multi-agent framework (Li et al., 2023a). We instantiate a sequential multi-agent pipeline upon Qwen3-1.7B-Base (Yang et al., 2025a), where five role-specific LoRA adapters are attached to the shared base model. All agents collaboratively analyze, reason, argue through, solve, and verify the same mathematical problem, creating realistic repeated context hand-offs along the pipeline.

The training data is generated by Qwen3.8-27B (Qwen Team, 2026) served through vLLM. For every GSM8K (Cobbe et al., 2021) training question, the teacher plays five agent roles in sequence, yielding one multi-turn trajectory with the schema {`math_solver`, `mathematical_analyst`, `programming_expert`, `inspector`, `final_refer`}. Each trajectory is then re-rendered into one transcript with common system prompt, where the user turn of agent i carries that agent's role instruction (Table 8). Agent i is fine-tuned with standard next-token cross-entropy loss applied *only* to its own agent segment. All earlier tokens in the upstream prefix are masked to -100, so each LoRA observes the full upstream context for conditioning while being optimized exclusively for its role-specific output contract. At inference, agent $i{+}1$ delta-prefills only its new user prompt on top of agent i's KV cache.

Table 8: Per-agent role instructions. Each adapter is trained on the shared-prefix transcript with loss restricted to its own assistant turn.

#	Agent	Prompt (user-turn template)
1	MATH SOLVER	Solve the given math problem step-by-step. Organize each reasoning step using `<step>` and `</step>` tags. Enclose your full solution and final answer within `<sol>...<solver_ans>XX</solver_ans></sol>`.
2	MATHEMATICAL ANALYST	Perform formal mathematical analysis over the solution trajectory from the solver. Declare variables, state constraints, and write logical deductions. Do not restate descriptive narratives. Wrap your formal analysis and derived result inside `<analyst_ans>XX</analyst_ans>`.
3	PROGRAMMING EXPERT	Translate the mathematical reasoning into runnable executable code. Output only structured content wrapped inside `<py_exec>...</py_exec>`, utilizing `<ast>`, `<py_fn>`, and `<stdout>`. Do not generate any extra natural-language text outside these tags.
4	INSPECTOR	Conduct a multi-path audit to examine outputs from the Math Solver, Mathematical Analyst, and Programming Expert. Summarize your audit outcome and verdict inside `<audit_dossier>...<inspector_verdict>VALID (XX)</inspector_verdict>`.
5	FINALREFER	Cross-examine all audit dossiers produced by previous agents. Keep your response within 1k words. Wrap your consensus result and boxed answer in `<consensus_arbiter>...\boxed{XX}`.

We further analyze the memory overhead incurred by MULA across repeated agent hand-offs over 10 conversation rounds, where each round appends roughly 8k tokens to the cumulative context. Figure 10 visualizes the simulated peak memory breakdown and relative overhead dynamics under a fixed 10% recomputation budget. On the left subplot, we decompose total GPU memory consumption into four components: model weights, standard dense KV cache, sparse onset cache, and the turn-wise hidden-state pool.

We clarify these four memory components explicitly.

1. *Model weights*: correspond to the fixed footprint of backbone model parameters, independent of conversation length.
2. *Standard dense KV cache*: the conventional key-value cache holding attention states for every token in the full conversation context during generation.
3. *Sparse onset cache*: persistently stores hidden states of selected onset tokens for each upstream agent. After an agent completes decoding and executes the onset-token SELECT

procedure, only hidden states of selected onset tokens are kept here for later sparse layer-wise recomputation by downstream agents.

4. *Turn-wise hidden-state pool*: a per-turn temporary buffer that exists only during the current agents decoding. Critically, this pool only holds hidden states for the **newly generated context of the current agent**, and does not store tokens inherited from earlier conversation rounds. Since the SELECT operation cannot run until decoding terminates, we cannot identify onset tokens on-the-fly during generation. All candidate hidden states produced for this turns new tokens are buffered here until decoding finishes and SELECT completes. After populating sparse onset cache from these candidates, the turn-wise hidden-state pool is entirely freed. The observed GPU memory peak occurs when this temporary buffer co-resides with sparse onset cache inherited from previous rounds.

The total peak memory gradually rises as the conversation context grows, yet remains safely below the 24 GB memory limit of an NVIDIA 4090 GPU throughout all ten rounds. The right subplot quantifies MuLAs relative overhead: the overhead ratio measured against raw KV cache size drops rapidly in early rounds and converges asymptotically to approximately 21% as the context length accumulates. Meanwhile, the overhead relative to overall peak GPU memory stays consistently low, hovering around 15% across all conversation turns. This demonstrates that MuLA introduces only modest, bounded memory cost for enabling repeated KV-cache migration in multi-agent sequential pipelines, and the relative overhead diminishes as the context window expands.

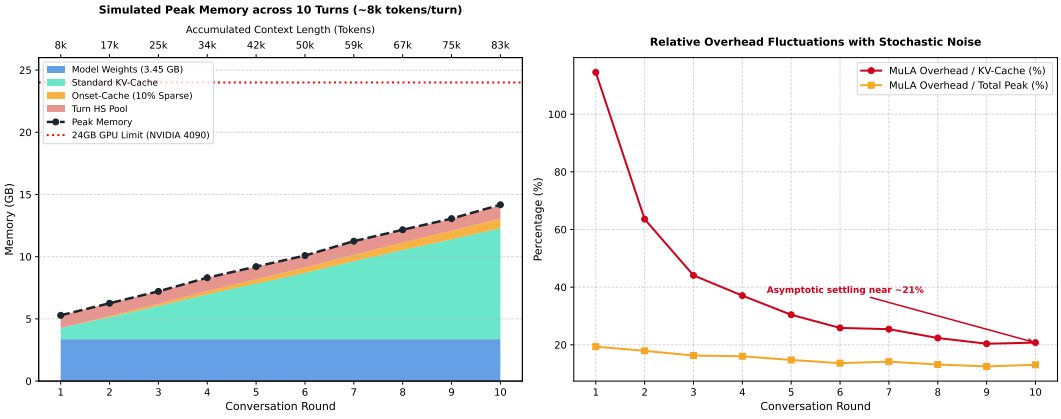


Figure 10: Simulated peak-memory breakdown over 10 conversation rounds (∼8k tokens/turn, 10% repair ratio). Left: stacked area decomposes peak usage into model weights, standard KV-cache, sparse onset-cache, and per-turn hidden-state pool; the dashed line tracks total peak against the 24 GB GPU limit, with accumulated context length on the top axis. Right: MuLA's extra overhead stays near ∼21% of the KV-cache as context accumulates.

G.3.1 Multi-Handoff Verification of the Core Attention Predictor

In Section 3.2, we presented our key observation: upstream shallow attention concentration anticipates downstream deep-layer LoRA KV cache deviation hotspots. This finding was established based on a single representative agent transition. The 5-Agent GSM8K multi-agent cascade offers four distinct sequential context hand-offs:

1. **Hand-off 1**: Agent 1 (Math Solver) → Agent 2 (Mathematical Analyst)
2. **Hand-off 2**: Agent 2 (Mathematical Analyst) → Agent 3 (Programming Expert)
3. **Hand-off 3**: Agent 3 (Programming Expert) → Agent 4 (Inspector)
4. **Hand-off 4**: Agent 4 (Inspector) → Agent 5 (FinalRefer)

To rigorously verify that our observation holds across diverse role semantics, we compute the layer-lag dynamics and token-level error predictability for all four hand-offs over full multi-turn conversational

trajectories. Figure 11 provides token-level 1D overlay traces across all four hand-off pairs, showing that upstream shallow attention spikes consistently pinpoint the exact downstream deep-layer KV error bursts across all agent roles.

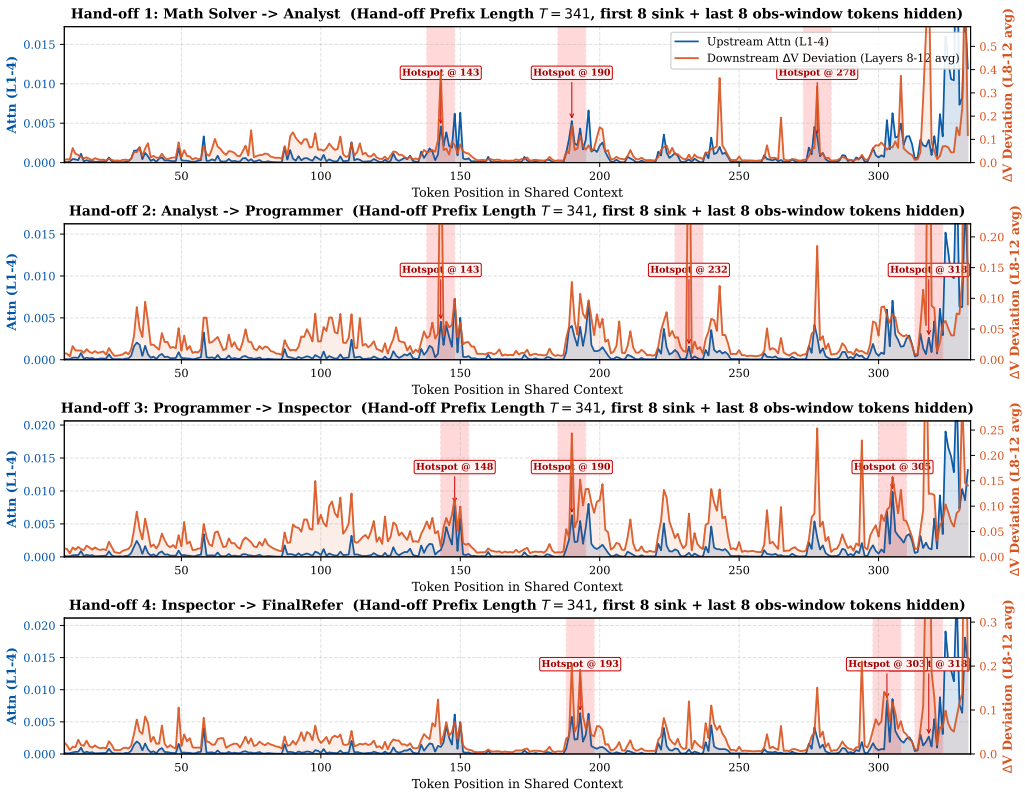


Figure 11: Token-level alignment between upstream shallow attention concentration score (blue line and shading) and downstream deep-layer LoRA V-cache cosine distance ($1-\cos(V_1, V_2)$, red line and shading) across all four sequential hand-offs in the 5-Agent GSM8K multi-agent cascade. Highlighted red intervals pinpoint representative error hotspots along token positions where upstream attention spikes anticipate downstream deep angular deviation peaks across diverse semantic transitions (from word-problem solving, formal analysis, and code synthesis to verification and arbitration.).

H Limitations and Future Works

MuLA currently focuses on sequential chain-structured hand-offs, where each transition inherits from a single predecessor cache: $C_m = C_{m-1} \oplus D_{m-1} \oplus I_m$ with monotonically growing repair union $\mathcal{F}_l^{(m)} = \mathcal{F}_l^{(m-1)} \cup S_l^{(m-1)}$. This sequential setting already covers a broad class of multi-agent pipelines, from Chain-of-LoRA role-specialized workflows (Liu et al., 2026a; Lee et al., 2026; Yu et al., 2024; Topsakal & Akinci, 2023) to sequential multi-agent inference with cache reuse (Geng et al., 2026; Liu et al., 2026b), including the two-agent migration and five-agent cascade evaluated in this work. Parallel multi-agents fan-out/fan-in workflows (e.g., the LangGraph Router paradigm (Wang & Duan, 2024; Topsakal & Akinci, 2023)), where a shared router context C_0 branches to K agents producing $D^{(1)}, \ldots, D^{(K)}$ concurrently before a merger step, are beyond the current scope and suggest a promising direction for future extension. Such parallel orchestration is also widely adopted in multi-agent frameworks (Wu et al., 2023; Li et al., 2023a; Hong et al., 2024), making it a natural next step for cache-migration support.

Adapting MuLA to parallel fan-in would invite several interesting extensions. First, parallel branches naturally diverge in content and length, so the merger observes multiple candidate caches rather than a single $\mathrm{KV}_{m-1}^{\mathrm{gen}}$, together with additional cross-branch attention to be established. Recent work on

collective KV sharing (Bian et al., 2026; Shi et al., 2026; Ye et al., 2025) and disaggregated multi-agents serving (Wang et al., 2026a; Woo et al., 2026) suggests that managing multiple concurrent cache sources is increasingly practical. Second, positional bookkeeping (RoPE offsets, causal prefix boundaries p_i) and the onset cache $O_l^{(m)} = \mathcal{F}_l^{(m)} \setminus \mathcal{F}_{l-1}^{(m)}$ could be generalized to DAG-structured lineage.

Encouragingly, these extensions appear largely complementary to the present design. The intuition is simple: keep using MuLA wherever the cache still flows one-to-one or one-to-many, namely when the router hands the shared context to each parallel branch. Only the final merge step, where the short outputs $D^{(1)}, \ldots, D^{(K)}$ of all branches are stitched together, needs an exact re-prefill. In practice, the expensive part of the computation lives in the long shared prefix, which MuLA already accelerates, while the merge only processes short branch outputs and stays relatively cheap. This yields a practical recipe for mixed serial-parallel pipelines: accelerate the long stretches with MuLA and spend exact compute only where branches meet. Generalizing the formalism and kernels to arbitrary DAG-structured workflows is left to future work.