KVCMAS: EFFICIENT KV CACHE CORRECTION FOR SHARED CONTEXT IN MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Prompt-specialized multi-agent systems enable multiple agents to share a model while performing complementary roles to solve complex tasks. However, agent-specific prefixes change the KV cache generated for the same shared context, causing each agent to repeatedly prefill the growing context and construct a separate cache with high computation and memory overhead. Selective recomputation reduces this redundancy but still retains substantial model execution, while existing delta correction methods either support only recurring context relations or maintain memory-intensive online correction states for dynamically changing context. For first seen shared context, these methods also construct a reference cache outside the agent workflow, and an approximate correction at the first agent affects the outputs passed to subsequent agents. We present **KVCMAS**, an online KV cache correction framework that represents cross-agent cache deviations using compact low-rank states and seamlessly chains corrections along the agent workflow without an additional reference prefill. This design supports dynamically changing shared context while preserving an exact first-agent cache. Across multiple language and vision-language workloads, KVCMAS matches or improves the accuracy of prior KV cache sharing methods while achieving the lowest TTFT under highly concurrent serving. Under controlled serving traces, it provides a 2.0× TTFT speedup over inference without KV cache sharing and reduces peak GPU memory by up to 3.7× relative to a prior KV cache correction method. These results establish KVCMAS as an accurate and scalable KV cache sharing approach for prompt-specialized multi-agent serving.

1 INTRODUCTION

LLM-based multi-agent systems (MAS) improve performance on complex tasks by coordinating agents specialized for complementary roles, such as planning, tool execution, critique, and orchestration (Li et al., 2023; Zhu et al., 2025; Dong et al., 2025; Yu et al., 2026; Zhang et al., 2024; 2026b; Hong et al., 2024; Li et al., 2025; 2026b; Zong et al., 2024). Such specialization is implemented by heterogeneous models, role-specific adapters, or distinct role prompts over a shared model (Chen et al., 2024; Lee et al., 2026; Kong et al., 2024). Among these designs, prompt-based specialization over a shared model is particularly practical for serving because agents share model weights and new roles require no additional training (Kong et al., 2024; Wang et al., 2024; Li et al., 2025; Wang et al., 2026). During execution, user queries, retrieved information, tool observations, and agent outputs propagate across agents, forming trajectories that interleave agent-specific prefixes with shared context (Tang et al., 2025; Li et al., 2024; Konstantinova & Grosenick, 2026). As these trajectories grow through multi-turn interactions, additional agents, and multimodal inputs, repeated shared context processing becomes a major serving overhead (Kim et al., 2026; Zhu et al., 2026; Bian et al., 2026).

Although prompt-specialized agents share model weights, their different prefixes change the hidden states and KV caches generated for the same shared context (Yao et al., 2025; Liu et al., 2026; Li et al., 2026a). Consequently, each agent repeatedly prefills the overlapping context and retains a separate KV cache. Directly reusing a cache constructed under another prefix avoids this redundancy but causes substantial accuracy degradation (Ye et al., 2025; Geng et al., 2026; Ma et al., 2026). Selective recomputation mitigates this error by rebuilding selected layers or tokens through model execution, but recovering more accuracy requires additional prefill computation (Yao et al., 2025; Liu et al., 2026; Geng et al., 2026). Delta correction instead estimates the cache difference induced

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.

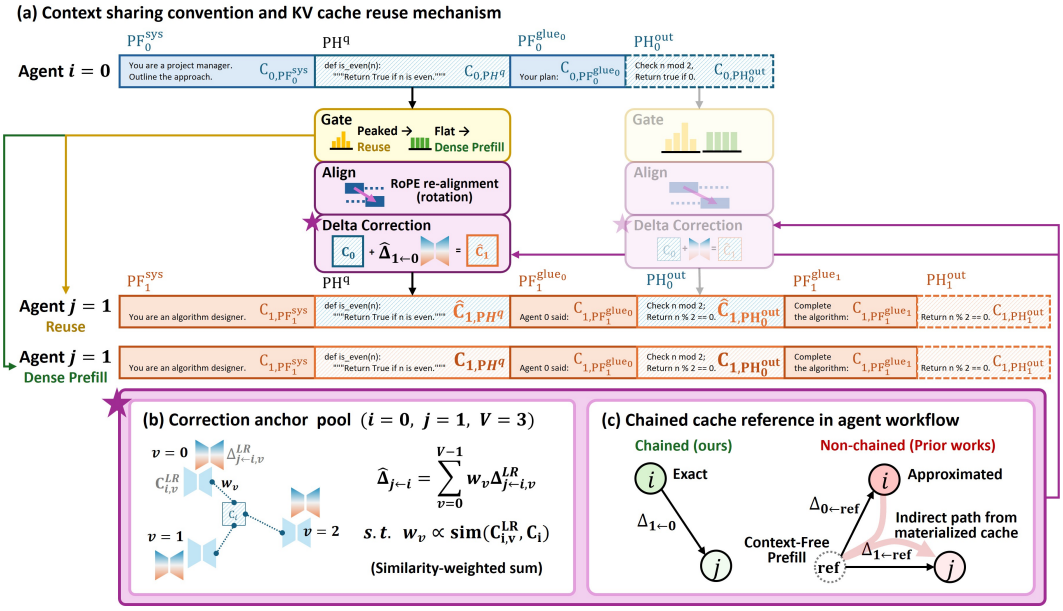

Figure 1: Overview of the context sharing convention and KVCMAS on an agent workflow edge from agent i to agent j. (a) Agent trajectories interleave agent-specific prefix segments (PF) with shared context segments (PH). KVCMAS reuses the preceding agent's KV cache for a PH segment when its correction is reliable and otherwise falls back to dense prefill. (b) KVCMAS matches the current cache representation to the online anchors of the corresponding shared segment and estimates the required KV cache correction by combining their low-rank corrections with similarity-based weights. (c) The corrected cache becomes the reference for the next workflow edge, chaining corrections along the agent workflow without constructing a separate context-free reference cache.

by the target context and applies it without executing the model for the reconstruction (Li et al., 2026a; Ma et al., 2026; Ye et al., 2025). Most existing correction methods construct relation-specific corrections for recurring context relations (Li et al., 2026a; Ma et al., 2026), but these corrections remain tied to previously observed context, making accurate correction challenging for new user requests and agent outputs. KVComm supports dynamically changing context through an online anchor pool, but its full-dimensional base caches and agent-specific corrections cause memory usage to grow with the shared context length (Ye et al., 2025). Existing delta correction methods also construct agent-specific caches from a separate context-free or position-independent reference. For first seen shared context, constructing this reference requires an additional prefill outside the actual workflow. Moreover, correction error at the first agent produces an inaccurate output that propagates to subsequent agents as shared context.

To address these limitations, we propose **KVCMAS**, an online KV cache correction framework for dynamically shared context in prompt-specialized multi-agent systems. One of our key ideas is that cross-agent KV cache deviations are concentrated in a low-dimensional feature space, allowing KVCMAS to store online correction states in compact low-rank form. We further observe that using an exact first-agent cache and workflow-relative references reduces aggregate correction error while eliminating a separate context-free reference prefill. Based on this observation, KVCMAS densely prefills the first agent and uses each corrected shared context cache as the reference for the next workflow edge. Across text and vision workloads, KVCMAS retains competitive multi-agent task accuracy while reducing correction memory and improving serving efficiency.

2 Background

2.1 Context Sharing Convention

We consider a MAS in which multiple agents share the same model weights but use different agent-specific prefixes. As illustrated in Figure 1(a), the trajectory of agent i interleaves agent-specific

prefix segments (PF) with shared context segments (PH):

$$x_i = \left[\mathrm{PF}_i^{\mathrm{sys}},\, \mathrm{PH}^{\mathrm{q}},\, \mathrm{PF}_i^{\mathrm{glue}_0},\, \mathrm{PH}_0^{\mathrm{out}},\, \mathrm{PF}_i^{\mathrm{glue}_1},\, \mathrm{PH}_1^{\mathrm{out}},\, \ldots\right] \tag{1}$$

Here, $\mathrm{PF}_i^{\mathrm{sys}}$ denotes the system prompt of agent i, while $\mathrm{PF}_i^{\mathrm{glue}_k}$ denotes the agent-specific glue prompt placed before the output of agent k. In contrast, PH^{q} contains the user query and task observations, while $\mathrm{PH}_k^{\mathrm{out}}$ contains the output of agent k. These PH segments form the shared context propagated across downstream agents. A downstream agent j follows the same structure with its own PF segments, and its shared context additionally includes $\mathrm{PH}_i^{\mathrm{out}}$ after an edge $i \to j$. Such an edge represents an arbitrary agent transition, including those in sequential, loop, fan-in, and fan-out workflows.

For a segment seg in the trajectory of agent i, we denote its preceding context by $c_{i,\mathrm{seg}}$ and its resulting KV cache by $C_{i,\mathrm{seg}} \in \mathbb{R}^{L_{\mathrm{seg}} \times D}$, where L_{seg} is the segment length and D is the KV cache feature dimension:

$$C_{i,\mathrm{seg}} = \mathrm{KV}(\mathrm{seg} \mid c_{i,\mathrm{seg}}) \tag{2}$$

Here, $\mathrm{KV}(\mathrm{seg} \mid c)$ denotes the KV cache constructed for segment seg under preceding context c. An agent-specific PF segment differs in content across agents and is therefore not a target for cross-agent KV cache sharing. In contrast, a PH segment contains the same content across agents and provides the main opportunity for KV cache sharing, while its preceding context differs because of their agent-specific PF segments. Consequently, the same PH segment generally produces different KV caches across agents, i.e., $C_{i,\mathrm{seg}} \neq C_{j,\mathrm{seg}}$. This cross-agent cache deviation prevents direct reuse of otherwise overlapping PH caches. We next describe how existing KV cache sharing methods address this deviation.

2.2 KV Cache Sharing Methods

KV cache sharing reuses a cache previously constructed for an overlapping PH segment, avoiding repeated prefill. However, the reused cache must account for cross-agent cache deviation to preserve accuracy. Existing methods address this deviation through selective recomputation or delta correction. Here, we note that all evaluated methods, including KVCMAS, re-align cached keys to their target positions before cross-agent reuse. We omit this deterministic alignment from the notation below.

Selective recomputation. Selective recomputation replaces a selected subset of the reused KV cache with the corresponding cache recomputed under the target agent's context. Let $C_{i,\mathrm{seg}}$ be the reused cache, $C_{j,\mathrm{seg}}$ be the target cache, and $\mathcal{T}$ be the cache subset selected for recomputation, which corresponds to layers, tokens, or their combination. The resulting cache $\widehat{C}_{j,\mathrm{seg}}$ is

$$\widehat{C}_{j,\mathrm{seg}}[u] = \begin{cases} C_{j,\mathrm{seg}}[u], & u \in \mathcal{T} \\ C_{i,\mathrm{seg}}[u], & u \notin \mathcal{T} \end{cases} \tag{3}$$

where u denotes a generic cache index. DroidSpeak identifies critical layer groups by profiling layer-wise KV cache deviation offline and recomputes the full shared context within the selected layers (Liu et al., 2026). CacheBlend instead selects tokens based on KV cache deviation measured in early layers and recomputes the selected tokens through subsequent layers (Yao et al., 2025). RelayCaching further combines KV cache deviation with attention scores to select important tokens and restricts their recomputation to critical middle layers (Geng et al., 2026). However, selective recomputation corrects only the selected cache subset, leaving cache deviations outside $\mathcal{T}$ in the reused cache. Reducing this remaining deviation requires recomputing a larger subset, which increases prefill computation and produces a trade-off between accuracy and efficiency. We provide a detailed analysis of the limited reconstruction coverage of selective recomputation in Appendix A.1.

Delta correction. Delta correction estimates the KV cache deviation from a reference cache to the target agent cache and applies it without executing the model over the corrected entries. Let $C_{\mathrm{ref,seg}}$ denote the reference cache for segment seg and $C_{j,\mathrm{seg}}$ denote its target cache for agent j. In the correction methods considered in this work, $C_{\mathrm{ref,seg}}$ is generated from the shared segment without an agent-specific prefix and serves as a context-free reference. The corrected cache is given by

$$\widehat{C}_{j,\mathrm{seg}} = C_{\mathrm{ref,seg}} + \widehat{\Delta}_{j \leftarrow \mathrm{ref,seg}} \tag{4}$$

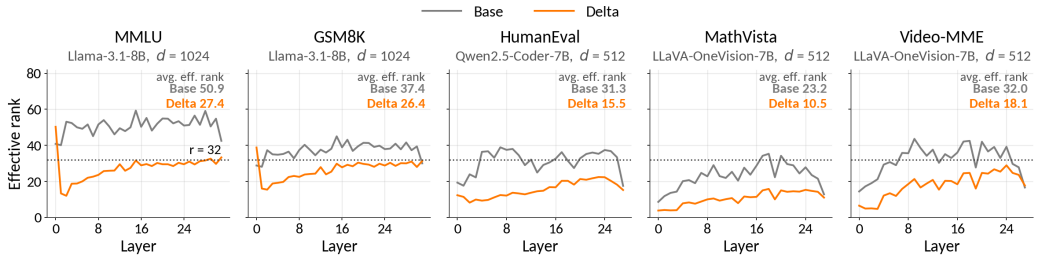

Figure 2: Effective rank of the base cache and delta correction across layers and workloads. Effective rank is defined as the minimum rank retaining 90% of the singular-value energy. The annotated values report the effective rank averaged across layers, and the dotted line marks rank 32.

where $\widehat{\Delta}_{j\leftarrow\mathrm{ref,seg}}$ estimates the exact cache deviation $\Delta_{j\leftarrow\mathrm{ref,seg}} = C_{j,\mathrm{seg}} - C_{\mathrm{ref,seg}}$. When the estimated correction is accepted, delta correction avoids the model execution required by selective recomputation, while an unreliable correction falls back to dense prefill.

GraphFlow constructs and reuses corrections for recurring transitions between agent operations (Li et al., 2026a). Kamera similarly constructs corrections for repeated multimodal content based on the context that precedes it (Ma et al., 2026). However, because the task-specific context carried by PH changes across requests and interactions, a correction constructed for a previously observed context relation does not represent new user requests or agent outputs. Therefore, in dynamic context settings, correction reuse is limited, requiring a newly constructed correction or dense prefill. KVComm supports dynamically changing shared context using an online anchor pool (Ye et al., 2025). Each anchor stores a context-free base cache and the corresponding agent-specific delta corrections. For a new segment, KVComm compares its base cache with the stored anchors and estimates the correction as their weighted combination:

$$\widehat{\Delta}_{j\leftarrow\mathrm{ref,seg}} = \sum_{v=0}^{V-1} w_v \Delta^{(v)}_{j\leftarrow\mathrm{ref,seg}}, \qquad \sum_{v=0}^{V-1} w_v = 1 \tag{5}$$

where V is the number of anchors and w_v is the normalized weight determined by the similarity between the current and stored base caches. An unreliable match falls back to dense prefill and provides a new observed correction. However, storing both the base caches and delta corrections in full-dimensional form makes memory usage grow with the shared context length, the number of anchors, and the number of agents. Furthermore, the delta correction methods above construct their corrected caches relative to a separately constructed context-free or position-independent reference. For first seen shared context, constructing this reference adds a prefill outside the agent workflow. Moreover, correction error at the first agent produces an inaccurate output that propagates to subsequent agents as shared context. This early error substantially increases the aggregate error across the workflow. We also note that works using specialized transfer mechanisms for KV cache sharing require additional training or calibration (Fu et al., 2026; Heo et al., 2026; Yang et al., 2025b), while system-level cache optimizations are complementary to cross-agent cache correction (Yang et al., 2025a; Gim et al., 2024; Bian et al., 2026; Zhang et al., 2026a; Pan et al., 2025).

3 METHODOLOGY

In this section, we present **KVCMAS**, an online KV cache correction framework for dynamically shared context in prompt-specialized multi-agent systems. As illustrated in Figure 1, KVCMAS consists of two major components. First, it stores the base representations and delta corrections of the online anchor pool in low-rank form, reducing the memory cost of online correction. Second, it defines each correction relative to the source agent cache already produced along the current workflow edge, avoiding a separately constructed context-free reference cache. We first present the observations behind these designs and then describe the KVCMAS correction procedure. Unless otherwise noted, the observations use the workloads and models described in Section 4.

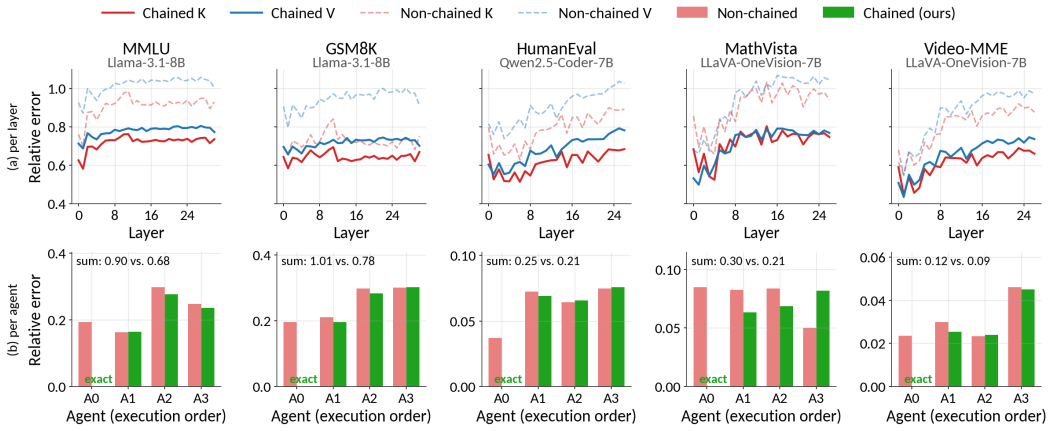

Figure 3: KV cache reconstruction error normalized by the Frobenius norm of the target tensor, using a context-free reference (non-chained) and the source agent cache (chained). The error is measured against the exact cache obtained through full prefill. (a) reports the error across layers, and (b) reports the error across agents in execution order A0→A1→A2→A3, with an exact A0 cache under chained correction.

3.1 Compact Online Correction via Low-Rank Anchor Pools

Online delta correction incurs substantial memory overhead because each anchor stores a full-dimensional base cache and agent-specific correction states. Our key observation is that these online correction states occupy a low-dimensional feature space. KVCMAS exploits this structure to compress the anchor pool while leaving the active KV cache used by attention full-dimensional. For a shared context of length L and KV cache feature dimension D, storing these states across V anchors and N consuming agents requires $\mathcal{O}(VNLD)$ memory for each pool. For example, with Llama-3.1-8B in BF16, a 4K-token segment pool with $V = 20$ and corrections for $N = 4$ agents requires approximately 50 GiB for the full-dimensional anchor states alone.

Specifically, we observe that agent-specific delta corrections have an effective rank below 32 across the various workloads, substantially below the full feature dimension (Figure 2). The base caches exhibit a similar structure, although their effective ranks are slightly higher, consistent with prior observations on KV caches (Chang et al., 2025; Saxena et al., 2024). KVCMAS therefore stores both base caches and delta corrections in low-rank form. The reconstructed base representations are used only for anchor matching, while the reconstructed correction is applied to the source-agent cache to materialize a full-dimensional target cache. Appendix A.2 further shows that PF-induced deviations have substantially lower effective rank than PH-induced deviations, highlighting the greater challenge of correcting dynamically changing PH.

3.2 Chained Correction along the Agent Workflow

Existing delta correction methods construct each agent-specific cache relative to a context-free reference $C_{\mathrm{ref,seg}}$. We refer to this reference structure as *non-chained* correction. Because $C_{\mathrm{ref,seg}}$ is not produced by an executed agent, constructing it for first seen shared context requires a separate prefill outside the actual workflow. Moreover, when only the materialized agent cache $\widehat{C}_{i,\mathrm{seg}}$ is carried forward, transitioning to agent j requires either retaining $C_{\mathrm{ref,seg}}$ separately or recovering it before applying $\widehat{\Delta}_{j\leftarrow\mathrm{ref,seg}}$. This produces the additional reference construction and indirect correction path illustrated in Figure 1(c). Non-chained correction also applies an approximate correction to the first agent, and the resulting cache error produces an inaccurate output that propagates to subsequent agents as another shared context.

In contrast, the source agent cache $C_{i,\mathrm{seg}}$ is already produced by the actual workflow and reflects the preceding interaction trajectory. The exact correction from source agent i to target agent j is

$$\Delta_{j\leftarrow i,\mathrm{seg}} = C_{j,\mathrm{seg}} - C_{i,\mathrm{seg}} = \Delta_{j\leftarrow\mathrm{ref,seg}} - \Delta_{i\leftarrow\mathrm{ref,seg}} \tag{6}$$

We refer to correction relative to $C_{i,\mathrm{seg}}$ as *chained* correction because its reference follows the agent workflow. With exact corrections, the context-free and source agent references are algebraically

equivalent. However, with empirical corrections obtained from actual model runs, the selected reference produces different approximation errors. KVCMAS densely prefills the first agent and chains correction from each materialized shared context cache. As shown in Figure 3, this structure reduces aggregate correction error across the evaluated layers and agents.

3.3 KVCMAS

Building on the two core ideas introduced above, KVCMAS combines low-rank online correction with chained correction along the agent workflow, as illustrated in Figure 1. KVCMAS processes agent-specific PF separately and applies online correction to shared segments. As shown in Figure 1(a), the first agent densely prefills its input and produces an exact cache. For a shared segment transferred from source agent i to target agent j, KVCMAS re-aligns the key positions in $C_{i,\text{seg}}$, matches the source cache to the online anchors, and estimates the required correction. Since $C_{i,\text{seg}}$ is already produced by the source agent, anchor matching requires no additional model forward pass.

KVCMAS maintains an anchor pool for each shared placeholder slot. Each anchor stores a base cache representation together with corrections for the consuming agents observed for that slot. Figure 1(b) shows the low-rank states stored in each anchor. For compact notation, let $Z \in \{C^{(v)}_{i,\text{seg}}, \Delta^{(v)}_{j\leftarrow i,\text{seg}}\}$ denote either the base cache or delta correction stored in anchor v. For each key or value matrix $Z \in \mathbb{R}^{L_{\text{seg}}\times D}$, KVCMAS applies truncated SVD:

$$Z = U_Z \Sigma_Z R_Z^\top \approx U_{Z,r}\Sigma_{Z,r}R_{Z,r}^\top = A_Z B_Z, \quad A_Z = U_{Z,r}\Sigma_{Z,r}, \;\; B_Z = R_{Z,r}^\top \tag{7}$$

where the subscript r denotes the components corresponding to the largest r singular values, $A_Z \in \mathbb{R}^{L_{\text{seg}}\times r}$, and $B_Z \in \mathbb{R}^{r\times D}$. Each anchor v stores the low-rank factors of its base cache $C^{(v)}_{i,\text{seg}}$ and the available corrections $\Delta^{(v)}_{j\leftarrow i,\text{seg}}$ for its consuming agents. KVCMAS compares $C_{i,\text{seg}}$ with the reconstructed base representation of each anchor and assigns larger weights to more similar anchors. Following Eq. 5, it estimates the transition correction directly from the stored low-rank factors:

$$\widehat{\Delta}_{j\leftarrow i,\text{seg}} = \sum_{v=0}^{V-1} w_v A_{\Delta,v} B_{\Delta,v}, \quad \sum_{v=0}^{V-1} w_v = 1 \tag{8}$$

where $A_{\Delta,v}B_{\Delta,v}$ represents the delta correction stored in anchor v and w_v denotes its normalized similarity weight. For a pool containing corrections for N consuming agents, this reduces its anchor memory from $\mathcal{O}(VNLD)$ to $\mathcal{O}(VNr(L+D))$.

Following KVComm (Ye et al., 2025), KVCMAS determines correction reliability from the normalized entropy of the anchor matching scores. A peaked distribution indicates a distinctive anchor match, whereas a flat distribution triggers dense prefill. A correction is accepted when its normalized entropy does not exceed the threshold τ, with a higher τ yielding a higher reuse ratio ρ. For an accepted correction, KVCMAS applies $\widehat{\Delta}_{j\leftarrow i,\text{seg}}$ to the aligned source agent cache and materializes the full-dimensional target cache $\widehat{C}_{j,\text{seg}}$. This cache is used by ordinary attention and becomes the shared context reference for the next workflow edge, realizing the chained correction in Eq. 6 without returning to a context-free reference cache. Otherwise, agent j densely prefills the segment and produces an exact cache. KVCMAS factorizes the current source cache and its observed transition correction when updating the corresponding anchor pool. Further details on the anchor pool management are provided in Appendix A.3.

4 Experiments

4.1 Experimental Setup

Workloads and Baselines. We evaluate KVCMAS using the multi-agent framework of KVComm (Ye et al., 2025), which builds on AgentPrune (Zhang et al., 2025) and GPTSwarm (Zhuge et al., 2024). We use Llama-3.1-8B-Instruct for MMLU and GSM8K, Qwen2.5-Coder-7B-Instruct for HumanEval, and LLaVA-OneVision-7B for MathVista and Video-MME. The LLM workloads follow the agent prompts of KVComm, while MathVista and Video-MME adapt the prompt structures of GSM8K and MMLU, respectively. Each accuracy workload uses three task-specific agents

Table 1: Accuracy (Acc., %), end-to-end latency (Lat., s), TTFT (s), peak GPU memory (Mem., GB), and KV cache reuse ratio (ρ). Excluding NonShared, the highest and second-highest accuracy among KV cache sharing methods are shown in **bold** and underlined, respectively.

(a) LLM benchmarks

	MMLU					GSM8K					HumanEval				
Method	**Acc.↑**	**Lat.↓**	**TTFT↓**	**Mem.↓**	ρ	**Acc.↑**	**Lat.↓**	**TTFT↓**	**Mem.↓**	ρ	**Pass@1↑**	**Lat.↓**	**TTFT↓**	**Mem.↓**	ρ
NonShared	67.54	18.59	0.06	16.78	0.00	85.60	20.82	0.12	16.88	0.00	86.96	8.80	0.05	15.28	0.00
FullShared	36.45	10.52	0.04	15.92	1.00	29.42	23.87	0.12	16.14	1.00	80.75	7.90	0.04	15.04	1.00
DroidSpeak	41.61	17.10	0.06	16.07	0.91	37.83	20.50	0.13	16.30	0.91	82.61	8.84	0.05	15.08	0.89
CacheBlend	55.39	18.49	0.07	16.15	0.90	77.03	19.49	0.12	16.29	0.90	82.61	8.34	0.05	15.02	0.90
RelayCaching	60.68	17.76	0.05	16.23	0.90	77.94	20.75	0.12	16.27	0.90	83.23	8.45	0.05	15.01	0.90
GraphFlow	36.38	21.36	0.09	16.12	0.82	31.46	30.17	0.18	16.46	0.79	60.87	8.82	0.05	15.10	0.83
KVComm	62.57	15.37	0.08	52.40	0.76	**80.14**	21.49	0.16	59.18	0.80	**85.09**	8.55	0.05	30.54	0.80
KVCMAS	**65.45**	14.21	0.05	16.06	0.79	79.53	19.71	0.12	16.10	0.76	84.47	8.32	0.05	15.04	0.78

(b) VLM benchmarks

	MathVista					Video-MME				
Method	**Acc.↑**	**Lat.↓**	**TTFT↓**	**Mem.↓**	ρ	**Acc.↑**	**Lat.↓**	**TTFT↓**	**Mem.↓**	ρ
NonShared	55.90	4.56	0.55	17.43	0.00	53.00	183.66	37.76	17.60	0.00
FullShared	50.20	0.50	0.26	16.70	1.00	37.25	22.65	0.48	16.74	1.00
DroidSpeak	53.00	3.49	0.46	16.86	0.79	48.84	311.84	29.53	16.88	0.79
CacheBlend	50.80	3.98	0.49	16.86	0.80	46.36	237.25	38.76	16.89	0.80
RelayCaching	50.50	3.96	0.49	16.89	0.80	44.92	243.64	40.72	16.86	0.80
GraphFlow	34.60	2.30	0.41	16.87	0.64	28.22	190.46	25.76	16.64	0.62
KVComm	53.50	2.43	0.42	59.19	0.63	51.56	163.06	24.30	53.27	0.62
KVCMAS	**53.80**	1.72	0.34	16.77	0.63	**52.36**	151.67	24.06	16.61	0.64

followed by a reflection agent. We include execution without KV cache sharing, which repeatedly prefills the shared context (**NonShared**), and direct cross-agent cache reuse without correction (**FullShared**) as reference baselines, together with DroidSpeak (Liu et al., 2026), CacheBlend (Yao et al., 2025), RelayCaching (Geng et al., 2026), GraphFlow (Li et al., 2026a), and KVComm (Ye et al., 2025). Since Kamera (Ma et al., 2026) similarly relies on corrections constructed for previously observed context relations, we use GraphFlow as the representative baseline for this setting; its results demonstrate the limitation of preconstructed corrections for dynamically changing shared context. Detailed workload and baseline configurations are provided in Appendix B.1.

Accuracy Evaluation. We denote ρ by the fraction of shared cache reused, corresponding to entries excluded from selective recomputation or served through accepted delta correction. Equal ρ does not imply equal computation because selective recomputation executes the model over recomputed entries, whereas delta correction directly updates accepted entries Ye et al. (2025); Ma et al. (2026). We evaluate selective recomputation at $\rho = 0.9$ and 0.8 and delta correction at approximately $\rho = 0.8$ and 0.6 for the LLM and VLM workloads, respectively. For methods with online anchor matching, we select the reliability threshold τ to reach these operating points. Unless otherwise stated, KVCMAS uses rank $r = 32$ for both base caches and delta corrections and maintains up to $V = 10$ anchors per shared context slot. Accuracy is averaged over three runs, while latency, TTFT, and peak memory are measured on an NVIDIA A100 80 GB GPU at 1 QPS, with TTFT averaged across each trajectory.

Serving Efficiency. We implement the methods upon vLLM (Kwon et al., 2023) and evaluate them using controlled agent traces across a QPS sweep. The traces fix a four-agent schedule and incorporate 128-token agent outputs into subsequent inputs while varying the non-generated shared context. We report request-level median (p50) and tail (p90) TTFT. We compare selective recomputation at $\rho = 0.9$ and 0.8 with delta correction at $\rho = 0.8$ and 0.6, respectively. These concurrent serving experiments run on an NVIDIA A100 80 GB GPU. To examine per-request single-batched efficiency, we measure memory usage and throughput under single-stream execution on an NVIDIA A6000 48 GB GPU. Using the same controlled setting, we further vary the rank, anchor pool size, number of agents, and number of interaction rounds in Appendix C.

4.2 Benchmark Accuracy

Table 1 reports accuracy and system performance across all workloads. KVCMAS achieves the highest accuracy among KV cache sharing methods on three of the five benchmarks and ranks second to KVComm on GSM8K and HumanEval within 0.7% points. It also provides the lowest

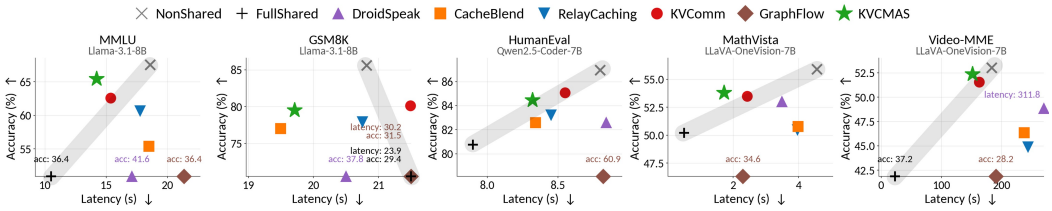

Figure 4: Benchmark accuracy and latency trade-off across the evaluated workloads. The shaded region spans FullShared and NonShared, with the upper left indicating a better trade-off. Out-of-range results are placed on the plot boundaries and annotated with their values. KVCMAS consistently remains near the upper-left Pareto frontier.

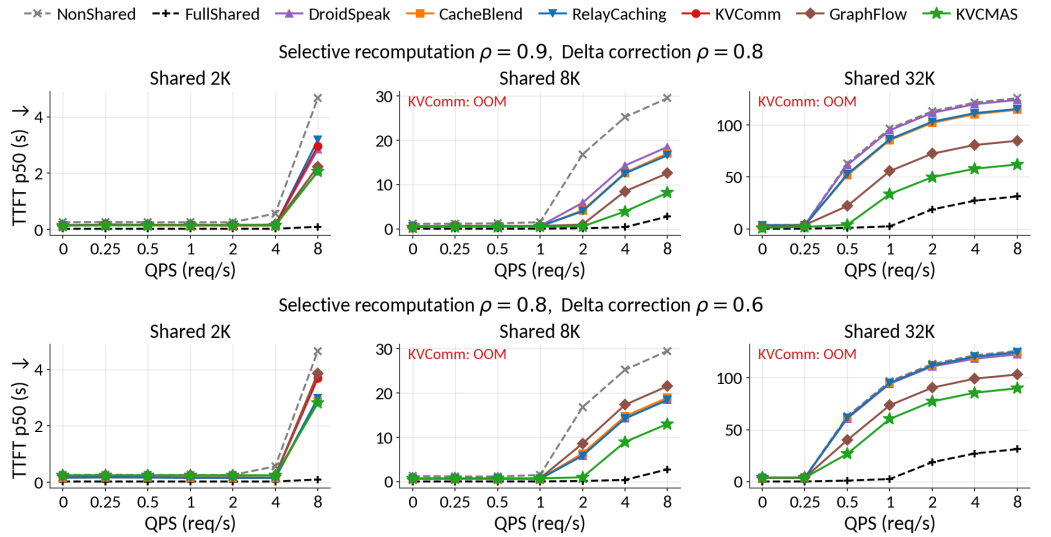

Figure 5: Median (p50) TTFT under concurrent serving across shared context lengths and QPS. The top row compares selective recomputation at $\rho = 0.9$ with delta correction at $\rho = 0.8$, while the bottom row compares $\rho = 0.8$ with $\rho = 0.6$, respectively. KVComm runs out of memory at 8K and 32K shared context.

end-to-end latency and TTFT among the delta correction methods across all workloads and achieves lower end-to-end latency than the selective recomputation methods except on GSM8K, despite using lower reuse ratios. Consequently, KVCMAS remains near the upper-left Pareto frontier in Figure 4, providing the strongest overall accuracy and latency trade-off.

Selective recomputation partially recovers the accuracy loss of FullShared by rebuilding a selected cache subset. However, deviations outside the selected subset remain uncorrected, while increasing its coverage requires additional model execution. This trade-off is particularly evident on Video-MME and aligns with the reconstruction analysis in Appendix A.1. Delta correction instead updates the reused cache without model execution. GraphFlow applies corrections constructed for previously observed context relations without online matching to the current shared context, leading to substantial accuracy degradation on dynamic requests. KVComm recovers much of this loss through online anchor matching, but its full-dimensional anchor pool uses up to 3.7× more peak GPU memory than KVCMAS. KVCMAS retains online matching while preserving an exact first-agent cache, chaining corrections along the workflow, and storing anchor states in low-rank form. It provides accuracy comparable to or higher than that of KVComm, with lower latency and memory usage. Because benchmark latency also reflects differences in generated responses and agent trajectories, Section 4.3 evaluates serving efficiency using controlled traces. In addition, Appendices B.2 and B.3 report run-to-run variation, trajectory lengths, and ablations of the reliability threshold.

4.3 Serving Efficiency

KVCMAS achieves the lowest median TTFT among the KV cache sharing works, with larger advantages at longer shared contexts and higher request rates, as shown in Figure 5. At 32K shared tokens

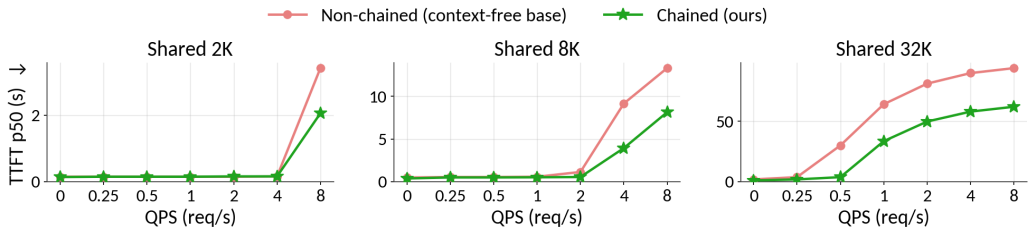

Figure 6: Median (p50) TTFT of chained and non-chained correction at $\rho = 0.8$ across shared context lengths and QPS. Non-chained correction uses a context-free reference, while chained correction follows the preceding agent cache.

and 8 QPS under the $\rho = 0.8$ setting, KVCMAS provides a 2.0× TTFT speedup over NonShared and 27% lower TTFT than GraphFlow, the next-fastest KV cache sharing work. NonShared repeatedly prefills the shared context, while FullShared avoids this computation through direct reuse at the cost of substantial accuracy degradation. DroidSpeak propagates hidden states for all shared context through the selected layers to update their KV projections, whereas CacheBlend and RelayCaching restrict model execution to selected tokens. Delta correction instead updates corrected cache entries without model execution, resulting in lower TTFT for GraphFlow, KVComm, and KVCMAS.

The TTFT gain of KVCMAS compared to previous delta correction methods becomes more pronounced as the shared context grows. For first seen shared context, non-chained correction constructs a separate context-free reference outside the agent workflow. KVComm additionally stores its online anchor states in full-dimensional form and runs out of memory at 8K and 32K shared contexts. KVCMAS stores these states in low-rank form and chains correction from a cache already produced by the workflow, avoiding the reference prefill and reducing anchor memory. The results at $\rho = 0.6$ follow the same scaling trend as those at $\rho = 0.8$. At shorter contexts and lower request rates, where prefill contributes less to serving time, the differences among methods are smaller.

Figure 6 compares chained correction with a non-chained variant while keeping the remaining procedure fixed. At 32K shared tokens and 8 QPS, chained correction reduces median TTFT by 34% by removing the reference prefill that would otherwise compete with concurrent requests for batching and compute resources. Appendices B.4 and B.5 provide the complete p50 values and additional p90 results for Figures 5 and 6, respectively. Appendix B.6 extends the evaluation to shared context accumulated from agent outputs, where KVCMAS retains its serving efficiency advantage.

Under single-stream execution, KVCMAS uses substantially less memory than KVComm while maintaining peak memory comparable to other methods, and its TTFT and throughput gains increase with shared-context length (Appendix C.1). The rank and anchor pool size ablations show that $r = 32$ and $V = 10$ capture most of the accuracy gains, while larger settings provide only marginal improvements with higher memory usage (Appendices C.2 and C.3). KVCMAS also retains the lowest TTFT and highest throughput among the KV cache sharing works as the number of agents and interaction rounds increases (Appendices C.4 and C.5).

5 Conclusion

We introduced KVCMAS, an online KV cache correction framework for dynamically changing shared context in prompt-specialized multi-agent systems. KVCMAS stores cross-agent cache deviations in compact low-rank form and chains corrections using caches naturally produced along the workflow. This provides an exact first agent cache, improves correction accuracy, and removes the reference prefill required by non-chained correction. Across language and vision-language workloads, KVCMAS achieves the highest overall accuracy and the lowest serving latency among KV cache sharing works at high request rates. At 32K shared tokens and 8 QPS, it provides a 2.0× TTFT speedup over NonShared, while chained correction reduces TTFT by 34% compared with non-chained correction. KVCMAS also reduces peak GPU memory by up to 3.7× relative to KVComm and retains the lowest TTFT and highest throughput as the number of agents and interaction rounds increases. These results establish KVCMAS as an accurate and scalable KV cache sharing approach for multi-agent serving.

AI USE STATEMENT

In this work, we used generative AI tools to aid and polish writing, to draft parts of the paper, to search for and identify related work, and for research ideation. We have not used generative AI tools for generating synthetic datasets or for proving mathematical claims, and the remaining tasks with required disclosure are not applicable to this work. All AI-assisted text was reviewed and revised by the authors, and all identified literature was verified against the original sources. We take responsibility for the final content of this work, including text produced with the aid of AI.

REPRODUCIBILITY STATEMENT

To ensure reproducibility, we provide the source code as an anonymous, downloadable package in the supplementary materials, including detailed instructions for environment setup and commands to run the accuracy benchmark and the efficiency measurements. All datasets used in our experiments are publicly available. The experimental settings are described in Section 4.1 and Appendix B.1, the anchor pool management and correction procedure are specified in Appendix A.3, and the complete numerical results are reported in Appendices B and C.

REFERENCES

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.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S. Abdelfattah, and Kai-Chiang Wu. Palu: Kv-cache compression with low-rank projection. In *International Conference on Learning Representations*, 2025.

Justin Chen, Swarnadeep Saha, and Mohit Bansal. ReConcile: Round-table conference improves reasoning via consensus among diverse LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7066–7085, 2024.

Yuhao Dong, Zuyan Liu, Hai-Long Sun, Jingkang Yang, Winston Hu, Yongming Rao, and Ziwei Liu. Insight-V: Exploring long-chain visual reasoning with multimodal large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9062–9072, 2025.

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://arxiv.org/abs/2510.03215.

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.

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.

Taekyung Heo, Rasoul Shafipour, Ritchie Zhao, Maximilian Golub, Mohammad Mahdi Kamani, Ritika Borkar, Makesh Tarun Chandran, Pantea Zardoshti, and Bita Darvish Rouhani. Cross-model KV cache transfer in LLM families: A closed-form linear mapping for prefill reuse. *arXiv preprint arXiv:2608.03893*, 2026. URL https://arxiv.org/abs/2608.03893.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, 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.

Donghwan Kim, Prakhar Singh, Younghoon Min, Jongryool Kim, Jongse Park, and Kiwan Maeng. Characterization of multi-model agentic AI systems on general tasks via trace-driven simulation. *arXiv preprint arXiv:2606.01725*, 2026.

Aobo Kong, Shiwan Zhao, Hao Chen, Qicheng Li, Yong Qin, Ruiqi Sun, Xin Zhou, Enzhi Wang, and Xiaohang Dong. Better zero-shot reasoning with role-play prompting. 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. 4099–4113, 2024.

Olha Konstantinova and Scott Grosenick. Passing context between agents in multi-agent A2A systems. Microsoft ISE Developer Blog, June 2026. URL https://devblogs.microsoft.com/ise/a2a-context-passing-multi-agent-systems/.

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

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*, pp. 6569–6596, 2026.

Ao Li, Shangpeng Yang, Fahao Chen, Tianheng Xu, Peng Li, and Zhou Su. GraphFlow: A graph-based workflow management for efficient LLM-agent serving. *arXiv preprint arXiv:2605.22566*, 2026a.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large scale language model society. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Haoran Li, Ziyi Su, Yun Xue, Zhiliang Tian, Yiping Song, and Minlie Huang. Advancing collaborative debates with role differentiation through multi-agent reinforcement learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22655–22666, 2025.

Peiwen Li, Shiyang Zhang, Yangtian Zhang, Sizhuang He, David van Dijk, and Rex Ying. MoRSE: Task-oriented multi-agent system with mixture of role-subtask experts. *arXiv preprint arXiv:2608.09251*, 2026b.

Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, Yi Yang, et al. A survey on LLM-based multi-agent systems: Workflow, infrastructure, and challenges. *Vicinagearth*, 1(1), 2024.

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, 2026.

Bole Ma, Jan Eitzinger, Harald Koestler, and Gerhard Wellein. Kamera: Unified position-invariant multimodal KV cache for training-free reuse. *arXiv preprint arXiv:2606.23581*, 2026.

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

Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. Eigen attention: Attention in low-rank space for KV cache compression. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 15332–15344. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-emnlp.899.

Yichen Tang, Weihang Su, Yujia Zhou, Yiqun Liu, Min Zhang, Shaoping Ma, and Qingyao Ai. Augmenting multi-agent communication with state delta trajectory. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025.

Yimeng Wang, Jiaxing Zhao, Hongbin Xie, Hexing Ma, Yuzhen Lei, Shuangxue Liu, Xuan Song, Zichen Zhang, and Haoran Zhang. MetaGen: Self-evolving roles and topologies for multi-agent LLM reasoning. *arXiv preprint arXiv:2601.19290*, 2026.

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.

Huan Yang, Renji Zhang, Mingzhe Huang, Weijun Wang, Yin Tang, Yuanchun Li, Yunxin Liu, and Deyu Zhang. KVShare: An LLM service system with efficient and effective multi-tenant KV cache reuse. *arXiv preprint arXiv:2503.16525*, 2025a. URL https://arxiv.org/abs/2503.16525.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. KVLink: Accelerating large language models via efficient KV cache reuse. *arXiv preprint arXiv:2502.16002*, 2025b. URL https://arxiv.org/abs/2502.16002.

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

Hancheng Ye, Zhengqi Gao, Mingyuan Ma, Qinsi Wang, Yuzhe Fu, Ming-Yu Chung, Yueqian Lin, Zhijian Liu, Jianyi Zhang, Danyang Zhuo, and Yiran Chen. KVCOMM: Online cross-context KV-cache communication for efficient LLM-based multi-agent systems. *arXiv preprint arXiv:2510.12872*, 2025.

Xinlei Yu, Chengming Xu, Zhangquan Chen, Yudong Zhang, Shilin Lu, Cheng Yang, Jiangning Zhang, Shuicheng Yan, and Xiaobin Hu. Visual document understanding and reasoning: A multi-agent collaboration framework with agent-wise adaptive test-time scaling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12300–12311, 2026.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for LLM-based multi-agent systems. In *International Conference on Learning Representations*, 2025.

Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language models. In *International Conference on Learning Representations*, 2024.

Rui Zhang, Chaeeun Kim, Shaoting Feng, Kuntai Du, Yuhan Liu, Yi Zhong, Cheng-Wei Ching, Junchen Jiang, and Liting Hu. Learning agent execution for KV-cache management in agentic serving. *arXiv preprint arXiv:2608.14624*, 2026a.

Zaibin Zhang, Junlan Xiao, Zhongbo Zhang, Yifan Wang, Li Kang, Yiran Qin, Changxing Xia, Heng Zhou, Talas Fu, Enshen Zhou, Ruimao Zhang, Zhenfei Yin, Huchuan Lu, and Lijun Wang. MA-VLA: Multi-arm vision-language-action model for collaboration and compositional generalization. *arXiv preprint arXiv:2608.25864*, 2026b.

Kan Zhu, Mathew Jacob, Chenxi Ma, Yi Pan, Stephanie Wang, Arvind Krishnamurthy, and Baris Kasikci. TraceLab: Characterizing coding agent workloads for LLM serving. *arXiv preprint arXiv:2606.30560*, 2026.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. MultiAgentBench: Evaluating the collaboration and competition of LLM agents. *arXiv preprint arXiv:2503.01935*, 2025.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62743–62767, 2024.

Chang Zong, Yuchen Yan, Weiming Lu, Jian Shao, Yongfeng Huang, Heng Chang, and Yueting Zhuang. Triad: A framework leveraging a multi-role LLM-based agent to solve knowledge base question answering. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 1698–1710, 2024.

A Analysis of KV Cache Reuse

A.1 Reconstruction Coverage of Selective Recomputation

Section 4.2 shows that selective recomputation incurs accuracy degradation relative to dense prefill (NonShared). To examine this difference, we measure how much KV cache reuse error is removed by representative layer and token selection criteria. We use the operating points from the accuracy evaluation: $\rho = 0.9$ for the LLM workloads and $\rho = 0.8$ for the VLM workloads. Throughout this work, KV cache reuse error is measured as the Frobenius norm of the cache difference normalized by the Frobenius norm of the corresponding full target tensor.

We analyze three criteria used as core components of existing selective recomputation methods. Layer selection recomputes layers with large KV cache deviation, following the offline profiling criterion of DroidSpeak (Liu et al., 2026). Deviation selection chooses tokens with large KV cache deviation, as used by CacheBlend (Yao et al., 2025). Attention selection chooses tokens with high attention importance, one of the criteria used by RelayCaching (Geng et al., 2026). We note that this analysis isolates each selection criterion rather than reproducing the complete execution procedure of each method. For each criterion, *Coverage* denotes the fraction of the total KV cache reuse error removed by recomputation. *Relative Coverage* normalizes this value by random selection under the same recomputation budget.

Table 2: Reconstruction coverage of selective recomputation across datasets and models. Coverage measures the fraction of KV cache reuse error removed by recomputation, and Relative Coverage normalizes it by random selection under the same budget. Higher is better for both.

	Layer Selection		**Deviation Selection**		**Attention Selection**	
Dataset / Model	**Coverage**	**Relative Coverage**	**Coverage**	**Relative Coverage**	**Coverage**	**Relative Coverage**
GSM8K Llama-3.1-8B	0.13	1.33×	0.16	1.61×	0.15	1.45×
MMLU Llama-3.1-8B	0.03	0.87×	0.27	2.66×	0.17	1.65×
HumanEval Qwen2.5-Coder-7B	0.29	2.67×	0.32	3.24×	0.13	1.33×
MathVista LLaVA-OneVision-7B	0.50	2.35×	0.22	1.10×	0.20	1.02×
Video-MME LLaVA-OneVision-7B	0.39	1.82×	0.21	1.03×	0.20	0.99×

Table 2 shows that the evaluated criteria generally identify more informative cache subsets than random selection. However, at least half of the reuse error remains outside the recomputed subset across all workloads. On MathVista and Video-MME, deviation and attention selection remove only about 20% of the error and remain close to random selection. This limited coverage is consistent with the larger accuracy gap of token selective recomputation on the VLM workloads in Section 4.2. Although coverage does not directly determine task accuracy because layers and token positions affect model outputs differently, these results show the difficulty of correcting cross-agent cache deviation under a limited recomputation budget. KVCMAS instead estimates corrections for the full reused cache without model execution over individual entries. This broader correction coverage leads to its higher accuracy with lower latency in long-context, high-load regimes despite its lower reuse ratio.

A.2 Low-Rank Structure of PF-Induced Delta

Section 3.1 shows that cross-agent corrections for dynamically shared PH admit compact low-rank representations. We further examine the cache deviations induced when recurring PF segments, including system and glue prompts, precede a shared segment. This analysis concerns corrections to the shared cache under different PF contexts rather than reuse of the agent-specific PF caches themselves. Before computing each delta, we re-align the key positions to remove deterministic differences introduced by RoPE. Unlike PH, which changes across requests and grows along the workflow, the PF relation for a given agent recurs across requests. KVCMAS therefore also applies to recurring PF relations: following the preconstructed correction setting of GraphFlow and Kamera (Li et al., 2026a; Ma et al., 2026), the corresponding delta is profiled once, stored in low-rank form, and positionally re-aligned when reused.

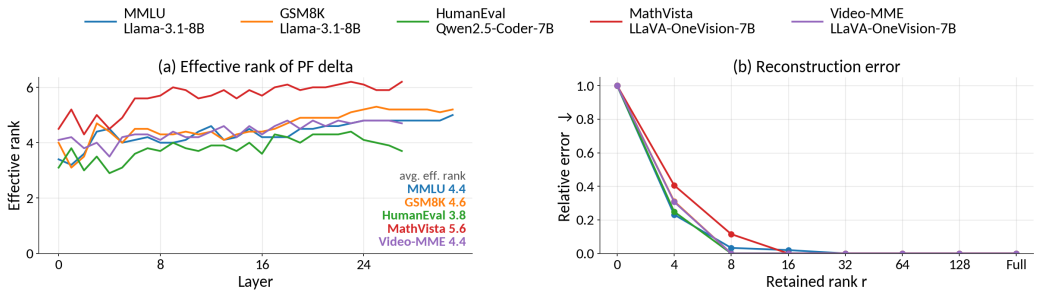

Figure 7: Effective rank and reconstruction error of PF-induced delta corrections. (a) reports the layer-wise effective rank, defined as the minimum rank retaining 90% of the singular-value energy, together with the average across layers for each workload. (b) reports the relative reconstruction error as the retained rank increases.

As shown in Figure 7, PF-induced cache deviations exhibit a more compact low-rank structure than deviations from dynamically changing PH. Their effective rank remains around 4–6 across the evaluated workloads, and rank 8 retains over 90% of the singular-value energy. We note that the lengths of both PF and PH segments are greater than their effective ranks, which reflects their actual low-rank structure. These results support applying the low-rank correction of KVCMAS to recurring PF without online anchor matching. This work focuses on the challenging dynamically changing PH, for which KVCMAS estimates the required correction online through the anchor pool.

A.3 ANCHOR POOL MANAGEMENT

Section 3.3 presents the low-rank online correction used by KVCMAS. Here, we detail its anchor matching, reliability control, and pool updates. Each shared context slot maintains a process-wide pool of up to V anchors, each storing low-rank source cache representations and corrections for its consuming workflow nodes. Following KVComm, KVCMAS uses normalized entropy as a confidence score. Low entropy indicates that one compatible anchor is distinctly preferred, whereas high entropy indicates that no candidate is clearly preferred.

Reliability Gate. For a comparison prefix of length k, let $\mathcal{A}_k$ contain the anchors whose stored length is at least k. KVCMAS reconstructs their value states layer by layer and computes:

$$d_v^{\text{gate}} = \left(\sum_{\ell=1}^{N_{\text{layer}}} \left\| V_{i,\text{seg}}^{(\ell)}[:k] - \widetilde{V}_{i,\text{seg}}^{(v,\ell)}[:k] \right\|_F^2 \right)^{1/2}, \qquad v \in \mathcal{A}_k. \tag{9}$$

The distances are converted into normalized gate weights:

$$g_v = \frac{\exp(-d_v^{\text{gate}}/T)}{\sum_{u \in \mathcal{A}_k} \exp(-d_u^{\text{gate}}/T)}, \qquad T = 1. \tag{10}$$

Reliability is determined from their normalized entropy:

$$\overline{H}(\mathbf{g}) = \frac{-\sum_{v \in \mathcal{A}_k} g_v \log_2 g_v}{\log_2 |\mathcal{A}_k|}. \tag{11}$$

Low entropy indicates a distinctive anchor match, whereas high entropy indicates no reliable match. KVCMAS accepts correction when $\overline{H}(\mathbf{g}) \leq \tau$; thus, a higher τ increases the realized reuse ratio ρ. Fewer than two compatible anchors trigger dense prefill. Appendix B.3 evaluates the resulting accuracy and reuse trade-off.

Correction Weighting and Positional Alignment. After accepting reuse, KVCMAS computes token-wise interpolation weights from the mean absolute source cache distance, separately for keys and values, and applies a softmax with $T = 1$. Equation 8 omits these token and key/value indices for readability. For key matching, the source and anchor keys are represented in the same canonical positional coordinates before their distances are computed. The corrected keys are then aligned to their target positions, while values require no rotation. These operations proceed layer by layer without an additional full-cache copy.

Variable Segment Lengths. Anchors retain their original lengths without padding or length-specific pools. The gate compares eligible anchors over the prefix of length k, while correction of a length-S segment uses anchors covering at least S tokens. If no anchor covers the full segment, KVCMAS corrects the longest covered prefix and retains the position re-aligned source cache for the remaining rows, avoiding extrapolation beyond observed anchor coverage.

Cache Materialization and Pool Updates. An accepted correction materializes the full-dimensional target cache, which becomes the source cache for the next workflow edge. Because interpolation quality varies across transitions, KVCMAS applies the reliability gate independently at every edge. An unreliable match triggers dense prefill, producing an exact target cache and resetting propagated cache error. After dense prefill, KVCMAS computes:

$$\Delta_{j \leftarrow i,\text{seg}} = C_{j,\text{seg}} - \text{Align}(C_{i,\text{seg}}), \tag{12}$$

where $C_{i,\text{seg}}$ is the materialized source cache carried by the workflow. It factorizes the source cache and observed correction and inserts their low-rank factors as a new anchor. Accepted corrections are not reinserted, so pool updates use target caches obtained through dense model execution. Following KVComm, the least frequently used anchor is evicted among the five oldest entries when the pool is full.

B MAIN EXPERIMENTS

B.1 EXPERIMENTAL SETUP

As described in Section 4.1, we build our evaluation on the multi-agent framework of KVComm (Ye et al., 2025). Table 3 summarizes the agent family, number of evaluated samples, and reliability threshold used for each workload.

Table 3: Detailed workload configuration used in the accuracy evaluation. MathVista and Video-MME inherit the agent prompt structures of GSM8K and MMLU, respectively.

	MMLU	**GSM8K**	**HumanEval**	**MathVista**	**Video-MME**
Agent Family	AnalyzeAgent	MathSolver	CodeWriting	MathSolver	AnalyzeAgent
# Samples	1,531	1,319	161	1,000	1,296
Threshold τ	0.29	0.32	0.34	0.50	0.60

Agent Roles. Following KVComm (Ye et al., 2025), we use three task-specific agents followed by a *FinalRefer* reflection agent, which reflects on the preceding agent outputs and produces the final answer. MMLU uses a *Knowledgeable Expert*, *Wiki Searcher*, and *Critic* to identify relevant entities, reason over retrieved information, and examine preceding analyses. GSM8K uses a *Math Solver*, *Mathematical Analyst*, and *Programming Expert* for complementary mathematical solutions, while HumanEval uses a *Project Manager*, *Algorithm Designer*, and *Programming Expert* for code planning and implementation. The *FinalRefer* agent produces the final answer from their outputs. MathVista and Video-MME retain the GSM8K and MMLU role structures, respectively, with prompts adapted to the provided image or video.

Workflow Structures. The correction procedure applies to sequential, loop, fan-in, and fan-out transitions. At fan-in, each incoming segment retains its source cache and is corrected separately. Dense fallback produces an exact target cache for the current segment and resets its propagated cache error for subsequent transitions. Appendices C.4 and C.5 evaluate varying numbers of agents and interaction rounds.

Benchmark Samples and VLM Inputs. Each configuration is evaluated three times, with accuracy variation reported in Appendix B.2. For MathVista, LLaVA-OneVision uses its any-resolution input path, producing visual context lengths that vary with image dimensions. For Video-MME, we sample 32 frames, resize each to 384×384, and pool each frame to 196 visual tokens.

Reliability Threshold. We select τ to obtain approximately $\rho = 0.8$ for the LLM workloads and $\rho = 0.6$ for the VLM workloads. Starting from the $\tau = 0.3$ default of KVComm (Ye et al., 2025), we use $\tau = 0.29$, 0.32, and 0.34 for MMLU, GSM8K, and HumanEval, respectively, and $\tau = 0.50$ and 0.60 for MathVista and Video-MME. Appendix B.3 reports the resulting accuracy and reuse trade-off.

Anchor Pool Size. KVComm uses $V = 20$ as its original balance between accuracy and efficiency. KVCMAS uses $V = 10$, which captures most of the accuracy benefit with a smaller pool; larger pools provide only marginal gains with additional memory overhead (Appendix C.3).

Factorization and Measurement. After dense fallback, KVCMAS computes a rank-r factorization for each layer's keys and values. Anchor construction occurs after the corresponding hop generates its first token and is therefore excluded from that hop's TTFT by definition. Its execution time is included in E2E latency and throughput, and peak GPU memory includes the transient factorization state.

Baseline Implementations. We implement all baselines within the same evaluation framework and apply the same key position alignment before cache reuse. GraphFlow is implemented as the representative preconstructed correction baseline: each correction is calibrated on the corresponding recurring context relation relative to a context-free reference and applied to subsequent requests without online matching. This preserves its correction construction and application while excluding graph-specific scheduling components outside the scope of cross-agent KV cache correction. KVComm follows its original online anchor matching procedure with $V = 20$, while the selective recomputation baselines follow the layer and token selection criteria described in Section 2.2.

B.2 ACCURACY EVALUATION

Section 4.2 reports the mean accuracy and latency measured on the trajectories generated by each method. Here, we examine the trajectory lengths covered by the evaluation and the accuracy variation across three runs. Table 4 reports the average and maximum accumulated trajectory lengths in model tokens, including visual tokens for VLM workloads.

Table 4: Average and maximum trajectory lengths during the accuracy evaluation, reported in model tokens. Visual tokens are included for MathVista and Video-MME.

	MMLU		GSM8K		HumanEval		MathVista		Video-MME	
Method	Average	Maximum	Average	Maximum	Average	Maximum	Average	Maximum	Average	Maximum
NonShared	2,324	3,176	2,267	3,460	1,809	3,277	4,216	9,032	6,594	7,659
FullShared	1,986	2,947	2,597	3,911	1,774	3,563	4,199	8,904	7,666	9,194
DroidSpeak	2,115	3,116	2,312	3,965	1,738	3,434	4,223	9,033	7,415	9,150
CacheBlend	2,345	3,432	2,275	3,919	1,774	3,104	4,213	9,033	6,943	9,206
RelayCaching	2,169	3,851	2,105	3,994	1,672	3,155	4,282	9,414	8,745	9,276
GraphFlow	2,461	3,201	2,670	4,438	1,866	3,519	4,230	8,904	6,645	7,657
KVComm	2,350	3,362	2,394	3,975	1,802	3,283	4,215	8,971	6,601	7,763
KVCMAS	2,253	3,362	2,312	3,525	1,818	3,404	4,219	9,036	6,733	8,078

The maximum lengths substantially exceed the averages for several workloads, particularly MathVista, showing that the evaluation includes inputs with long accumulated context. These samples cover the long-context regime in which the serving advantage of KVCMAS increases, as shown in Section 4.3.

Table 5 reports the standard deviation of accuracy in percentage points. KVCMAS shows standard deviations between 0.3% and 0.9% points, comparable to the other correction methods.

Table 5: Standard deviation of accuracy across three runs, reported in percentage points.

Method	MMLU	GSM8K	HumanEval	MathVista	Video-MME
NonShared	0.26	0.22	0.59	0.35	0.37
FullShared	0.27	0.28	0.66	0.35	0.36
DroidSpeak	0.43	0.46	1.03	0.54	0.58
CacheBlend	0.43	0.40	1.03	0.55	0.58
RelayCaching	0.43	0.40	1.01	0.55	0.57
GraphFlow	0.37	0.38	1.14	0.46	0.45
KVComm	0.37	0.33	0.80	0.47	0.50
KVCMAS	0.36	0.35	0.86	0.49	0.47

B.3 THRESHOLD

Section 4.1 selects the reliability threshold τ to obtain the target reuse regime for each workload. Here, we vary τ on representative LLM and VLM workloads, MMLU and Video-MME, and measure accuracy and the realized reuse ratio ρ. As defined in Appendix A.3, τ is applied to the normalized entropy of the anchor weights. A higher τ accepts a broader set of corrections and generally increases ρ.

Table 6: Effect of τ on accuracy (%) and reuse ratio. The settings used in the main evaluation are shown in **bold**.

MMLU			Video-MME		
τ	Accuracy	ρ	τ	Accuracy	ρ
0.10	67.47	0.623	0.20	52.98	0.590
0.20	66.82	0.705	0.40	52.88	0.610
0.25	66.62	0.720	0.50	52.80	0.630
0.27	66.82	0.762	0.55	52.64	0.630
0.29	**65.45**	**0.790**	0.58	52.44	0.630
0.30	64.66	0.820	**0.60**	**52.36**	**0.641**
0.31	64.47	0.875	0.62	52.14	0.641
0.33	64.01	0.886	0.65	52.04	0.650
0.35	61.92	0.905	0.70	51.96	0.650
0.40	59.96	0.939	0.80	51.95	0.649

Table 6 shows that τ controls the accuracy and reuse trade-off. Increasing the threshold generally raises ρ while gradually lowering accuracy. The selected thresholds reach the target reuse regimes without entering the more aggressive range where additional reuse causes larger accuracy loss.

On Video-MME, one of the three reuse-eligible transitions exhibits consistently large KV cache deviation and is routed to dense prefill throughout the threshold sweep. Consequently, ρ saturates near 0.65, close to the attainable reuse level when the other two transitions reuse their caches. Unlike fixed-budget selective recomputation, KVCMAS evaluates each transition independently and avoids forcing reuse when anchor correction is unreliable.

B.4 SERVING EFFICIENCY EVALUATIONS

Figure 5 in Section 4.3 summarizes median TTFT under concurrent serving. Here, we report the complete p50 and p90 values across shared context lengths and request rates, together with the p90 trends in Figure 8. The experiments use Llama-3.1-8B on an NVIDIA A100 80 GB GPU. We replay agent inputs with accumulated 128-token outputs and execute each request through its first generated token to measure TTFT. We compare selective recomputation at $\rho = 0.9$ and 0.8 with delta correction at $\rho = 0.8$ and 0.6, respectively. QPS 0 denotes single-stream request execution without overlapping arrivals.

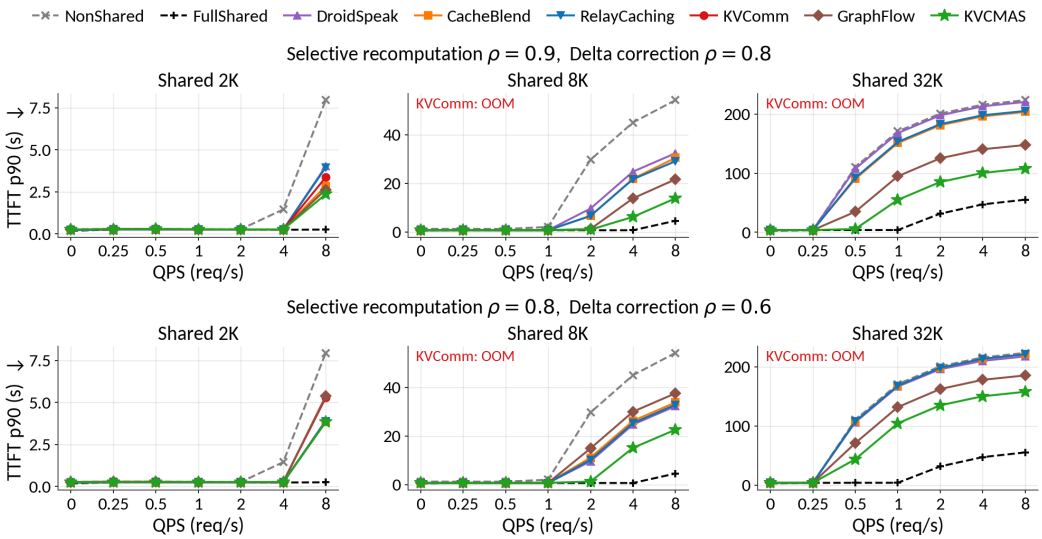

Figure 8: Tail (p90) TTFT under concurrent serving across shared context lengths and QPS. The reuse ratio configurations follow Figure 5.

Table 7 reports the corresponding numerical results. At 32K shared tokens and 8 QPS under the $\rho = 0.8$ delta correction setting, KVCMAS provides 2.0× and 2.1× p50 and p90 TTFT speedups over NonShared, respectively. It also reduces both p50 and p90 TTFT by 27% relative to GraphFlow, the next-fastest KV cache sharing work. The advantage increases with shared context length, showing the efficiency of KVCMAS for long-context workloads under high request rates. KVComm exceeds GPU memory capacity at 8K and 32K because it stores full-dimensional anchor states.

Table 7: Median (p50) and tail (p90) TTFT in seconds under concurrent serving across shared context lengths, request rates, and reuse ratios ρ.

(a) Shared context of 2K

		$\rho = 0.9$							$\rho = 0.8$							$\rho = 0.6$						
Pct.	**Method**	0	.25	.5	1	2	4	8	0	.25	.5	1	2	4	8	0	.25	.5	1	2	4	8
p50	NonShared	0.26	0.27	0.26	0.26	0.26	0.57	4.66	0.26	0.27	0.26	0.26	0.26	0.57	4.66	0.26	0.27	0.26	0.26	0.26	0.57	4.66
	FullShared	0.03	0.03	0.03	0.03	0.03	0.03	0.10	0.03	0.03	0.03	0.03	0.03	0.03	0.10	0.03	0.03	0.03	0.03	0.03	0.03	0.10
	DroidSpeak	0.17	0.17	0.17	0.16	0.16	0.16	2.84	0.16	0.17	0.17	0.16	0.16	0.16	2.93	0.16	0.18	0.17	0.16	0.16	0.16	3.99
	CacheBlend	0.15	0.16	0.15	0.14	0.14	0.14	2.07	0.17	0.18	0.17	0.17	0.17	0.16	2.87	0.17	0.18	0.18	0.17	0.17	0.17	3.11
	RelayCaching	0.15	0.16	0.16	0.15	0.15	0.15	3.18	0.17	0.18	0.17	0.16	0.16	0.16	2.99	0.17	0.18	0.18	0.17	0.17	0.17	3.30
	KVComm	0.14	0.16	0.16	0.16	0.16	0.16	1.49	0.14	0.18	0.17	0.17	0.16	0.17	2.96	0.25	0.25	0.25	0.25	0.25	0.25	3.69
	GraphFlow	0.14	0.16	0.16	0.16	0.16	0.16	1.12	0.14	0.18	0.17	0.17	0.16	0.18	2.24	0.25	0.25	0.25	0.25	0.25	0.25	3.87
	KVCMAS	0.14	0.14	0.14	0.14	0.14	0.14	0.28	0.14	0.15	0.15	0.15	0.16	0.16	2.06	0.25	0.25	0.25	0.25	0.25	0.25	2.81
p90	NonShared	0.27	0.29	0.29	0.27	0.27	1.45	7.95	0.27	0.29	0.29	0.27	0.27	1.45	7.95	0.27	0.29	0.29	0.27	0.27	1.45	7.95
	FullShared	0.18	0.25	0.26	0.26	0.25	0.24	0.25	0.18	0.25	0.26	0.26	0.25	0.24	0.25	0.18	0.25	0.26	0.26	0.25	0.24	0.25
	DroidSpeak	0.23	0.26	0.25	0.25	0.25	0.25	4.02	0.22	0.27	0.25	0.25	0.25	0.25	3.93	0.22	0.27	0.25	0.25	0.25	0.25	5.23
	CacheBlend	0.22	0.26	0.26	0.25	0.25	0.25	2.84	0.23	0.25	0.25	0.25	0.25	0.25	3.84	0.23	0.25	0.25	0.25	0.25	0.25	4.17
	RelayCaching	0.22	0.25	0.25	0.25	0.25	0.25	3.95	0.23	0.25	0.25	0.25	0.25	0.25	3.89	0.23	0.25	0.25	0.25	0.25	0.25	4.38
	KVComm	0.25	0.28	0.26	0.26	0.25	0.25	1.91	0.25	0.28	0.28	0.26	0.25	0.26	3.36	0.27	0.28	0.27	0.27	0.26	0.27	5.29
	GraphFlow	0.25	0.27	0.27	0.26	0.25	0.25	1.82	0.25	0.27	0.28	0.25	0.25	0.26	2.63	0.26	0.28	0.28	0.27	0.26	0.27	5.38
	KVCMAS	0.26	0.28	0.27	0.26	0.26	0.25	1.68	0.25	0.27	0.28	0.27	0.26	0.26	2.35	0.26	0.26	0.26	0.27	0.27	0.27	3.84

(b) Shared context of 8K

		$\rho = 0.9$							$\rho = 0.8$							$\rho = 0.6$						
Pct.	**Method**	0	.25	.5	1	2	4	8	0	.25	.5	1	2	4	8	0	.25	.5	1	2	4	8
p50	NonShared	1.15	1.18	1.25	1.51	16.81	25.24	29.52	1.15	1.18	1.25	1.51	16.81	25.24	29.52	1.15	1.18	1.25	1.51	16.81	25.24	29.52
	FullShared	0.03	0.04	0.04	0.04	0.14	0.39	2.78	0.03	0.04	0.04	0.04	0.14	0.39	2.78	0.03	0.04	0.04	0.04	0.14	0.39	2.78
	DroidSpeak	0.64	0.65	0.64	0.64	5.89	14.31	18.51	0.65	0.65	0.64	0.64	5.87	14.28	18.46	0.65	0.65	0.65	0.65	5.99	14.38	18.62
	CacheBlend	0.56	0.57	0.56	0.56	4.16	12.66	17.04	0.65	0.66	0.66	0.66	6.36	14.75	18.97	0.67	0.67	0.66	0.67	6.48	14.81	19.00
	RelayCaching	0.57	0.58	0.56	0.57	3.98	12.54	16.62	0.65	0.66	0.65	0.65	5.87	14.25	18.45	0.66	0.67	0.66	0.67	6.55	14.85	19.07
	GraphFlow	0.38	0.65	0.64	0.65	4.55	5.96	10.09	0.38	0.67	0.66	0.66	0.98	8.49	12.56	0.75	0.75	0.74	0.75	8.59	17.35	21.55
	KVCMAS	0.25	0.26	0.26	0.28	0.28	1.47	5.65	0.38	0.50	0.50	0.52	0.55	3.94	8.15	0.72	0.73	0.73	0.73	0.98	8.96	13.04
p90	NonShared	1.29	1.27	1.27	2.08	29.70	45.03	54.19	1.29	1.27	1.27	2.08	29.70	45.03	54.19	1.29	1.27	1.27	2.08	29.70	45.03	54.19
	FullShared	0.51	0.72	0.72	0.73	0.72	0.72	4.61	0.51	0.72	0.72	0.73	0.72	0.72	4.61	0.51	0.72	0.72	0.73	0.72	0.72	4.61
	DroidSpeak	0.70	0.74	0.72	0.72	9.64	24.82	32.37	0.70	0.74	0.72	0.72	9.63	24.75	32.29	0.70	0.72	0.73	0.73	9.84	24.95	32.57
	CacheBlend	0.68	0.75	0.72	0.72	6.59	21.89	30.49	0.71	0.75	0.73	0.74	11.15	26.24	33.83	0.71	0.74	0.72	0.72	10.81	25.84	33.40
	RelayCaching	0.68	0.73	0.72	0.72	6.64	21.71	29.09	0.70	0.73	0.72	0.72	10.24	25.34	32.89	0.70	0.73	0.72	0.72	10.91	25.91	33.50
	GraphFlow	0.74	0.75	0.74	0.75	6.55	9.49	17.09	0.74	0.77	0.75	0.76	1.23	13.84	21.65	0.76	0.78	0.76	0.76	14.95	30.02	37.58
	KVCMAS	0.74	0.74	0.74	0.74	0.75	1.93	9.15	0.74	0.75	0.74	0.75	0.96	6.21	13.84	0.76	0.76	0.76	0.78	1.31	15.16	22.63

(c) Shared context of 32K

		$\rho = 0.9$							$\rho = 0.8$							$\rho = 0.6$						
Pct.	**Method**	0	.25	.5	1	2	4	8	0	.25	.5	1	2	4	8	0	.25	.5	1	2	4	8
p50	NonShared	3.76	3.77	62.98	96.55	113.28	121.66	125.89	3.76	3.77	62.98	96.55	113.28	121.66	125.89	3.76	3.77	62.98	96.55	113.28	121.66	125.89
	FullShared	0.07	0.08	0.94	2.41	18.69	27.07	31.25	0.07	0.08	0.94	2.41	18.69	27.07	31.25	0.07	0.08	0.94	2.41	18.69	27.07	31.25
	DroidSpeak	3.66	3.69	61.32	94.92	111.71	120.13	124.17	3.65	3.66	60.63	94.23	110.86	118.35	122.50	3.65	3.66	60.12	93.64	110.58	119.00	123.28
	CacheBlend	3.30	3.31	51.59	85.37	102.11	110.43	114.72	3.68	3.70	61.73	95.00	111.81	120.28	124.64	3.73	3.73	61.37	94.94	111.89	120.29	124.69
	RelayCaching	3.35	3.35	52.60	86.33	103.17	111.51	115.40	3.67	3.69	61.50	94.99	111.83	120.32	124.55	3.72	3.73	61.42	95.15	112.57	120.09	124.61
	GraphFlow	0.91	3.70	9.62	43.77	60.95	69.78	74.10	1.90	3.70	21.96	55.62	72.40	80.76	84.84	3.73	3.75	39.95	73.52	90.45	99.15	103.14
	KVCMAS	0.91	0.92	2.27	18.05	36.02	44.44	48.62	1.02	1.90	3.78	33.52	49.75	57.93	61.95	3.75	3.74	26.70	60.29	77.20	85.63	89.84
p90	NonShared	3.79	3.80	110.37	170.76	200.93	216.00	223.59	3.79	3.80	110.37	170.76	200.93	216.00	223.59	3.79	3.80	110.37	170.76	200.93	216.00	223.59
	FullShared	2.64	3.75	3.81	3.89	31.59	47.30	55.20	2.64	3.75	3.81	3.89	31.59	47.30	55.20	2.64	3.75	3.81	3.89	31.59	47.30	55.20
	DroidSpeak	3.73	3.81	107.55	167.86	197.98	213.25	220.54	3.73	3.77	106.22	166.66	196.07	210.09	217.64	3.72	3.77	105.23	165.67	196.04	211.18	219.40
	CacheBlend	3.63	3.80	90.42	151.05	181.16	196.14	203.79	3.75	3.81	108.09	168.09	198.31	213.46	221.31	3.75	3.75	107.51	168.02	198.48	213.60	221.80
	RelayCaching	3.64	3.79	92.19	152.91	183.05	198.00	205.23	3.74	3.80	107.76	168.04	198.35	213.63	221.17	3.74	3.75	107.89	168.77	199.22	213.26	221.72
	GraphFlow	3.78	3.79	13.57	72.88	103.17	118.78	126.39	3.75	3.78	34.80	94.83	125.30	140.51	147.71	3.78	3.81	71.03	131.88	162.49	178.02	185.51
	KVCMAS	3.77	3.77	4.10	29.01	59.53	74.82	82.44	3.77	3.78	5.64	54.89	85.18	100.48	108.05	3.79	3.80	43.67	104.45	134.98	150.22	157.82

B.5 Chained Correction Efficiency Evaluations

Figure 6 in Section 4.3 compares chained correction with a non-chained variant using a separate context-free reference while keeping the remaining KVCMAS configuration fixed. Here, we report accuracy and numerical p50 and p90 TTFT across shared context lengths and request rates. Both variants use rank $r = 32$ and $V = 10$ anchors. Serving efficiency is evaluated at $\rho = 0.8$.

Table 8 shows that chained correction improves accuracy on both evaluated workloads, by 6.44% points on MMLU and 1.53 points on Video-MME. Chaining preserves an exact first-agent cache and uses each materialized cache for the next transition, preventing first-agent correction error from entering downstream shared context.

Table 8: Accuracy of chained and non-chained KVCMAS. The correction representation, anchor pool, reliability threshold, and reuse ratio follow the main accuracy evaluation settings.

Reference structure	MMLU	Video-MME
Chained	**65.45**	**52.36**
Non-chained	59.01	50.83

Figure 9 reports p90 TTFT, and Table 9 provides the numerical p50 and p90 results. By removing the separate reference prefill, chained correction reduces both median and tail TTFT. At 32K shared tokens and 8 QPS, it reduces both by 34%, with larger benefits at longer shared contexts and higher request rates.

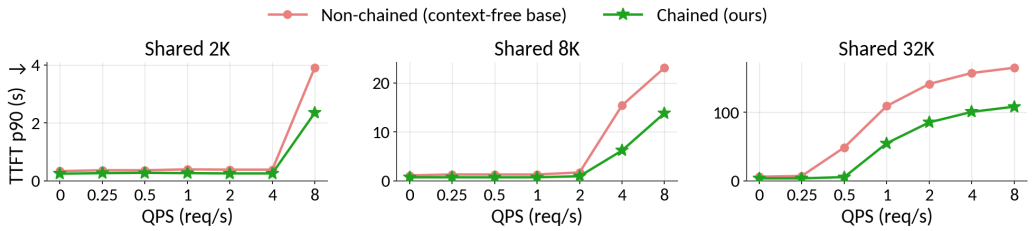

Figure 9: Tail (p90) TTFT of chained and non-chained correction across shared context lengths and request rates at $\rho = 0.8$. Chained correction reduces tail latency by avoiding the separate context-free reference prefill.

Table 9: Median (p50) and tail (p90) TTFT of chained and non-chained correction across shared context lengths and request rates at $\rho = 0.8$. All values are in seconds.

Pct.	Shared	Method	QPS						
			0	**0.25**	**0.5**	**1**	**2**	**4**	**8**
p50	2K	Chained	0.14	0.15	0.15	0.15	0.16	0.16	2.06
		Non-chained	0.15	0.15	0.15	0.15	0.15	0.16	3.42
	8K	Chained	0.38	0.50	0.50	0.52	0.55	3.94	8.15
		Non-chained	0.49	0.55	0.55	0.58	1.14	9.16	13.32
	32K	Chained	1.02	1.90	3.78	33.52	49.75	57.93	61.95
		Non-chained	1.92	3.78	29.99	64.07	81.07	89.77	93.86
p90	2K	Chained	0.25	0.27	0.28	0.27	0.26	0.26	2.35
		Non-chained	0.34	0.37	0.37	0.40	0.39	0.39	3.91
	8K	Chained	0.74	0.75	0.74	0.75	0.96	6.21	13.84
		Non-chained	1.13	1.33	1.32	1.32	1.74	15.41	23.09
	32K	Chained	3.77	3.78	5.64	54.89	85.18	100.48	108.05
		Non-chained	6.17	7.28	48.76	109.05	141.09	156.99	164.61

B.6 Serving Efficiency with Long Agent Outputs

Section 4.3 varies the non-generated shared context with 128-token agent outputs incorporated into subsequent inputs. Here, we extend the TTFT evaluation to 0.375K, 1.5K, and 6K accumulated output tokens. In the four-agent workflow, the final agent receives outputs from the three preceding agents, so the accumulated shared output is three times the per-agent generation length. The 0.375K results appear in Table 7(a). Figures 10 and 11 show the p50 and p90 trends, while Table 10 reports the numerical results for 1.5K and 6K.

At 6K shared generated tokens and 8 QPS under the $\rho = 0.8$ setting, KVCMAS provides 2.3× p50 and 2.4× p90 TTFT speedups over NonShared. It shows the same trend at 0.375K and 1.5K, retaining the lowest TTFT among the KV cache sharing works under high request rates. KVComm exceeds GPU memory capacity at 6K shared generated tokens because of its full-dimensional anchor pool.

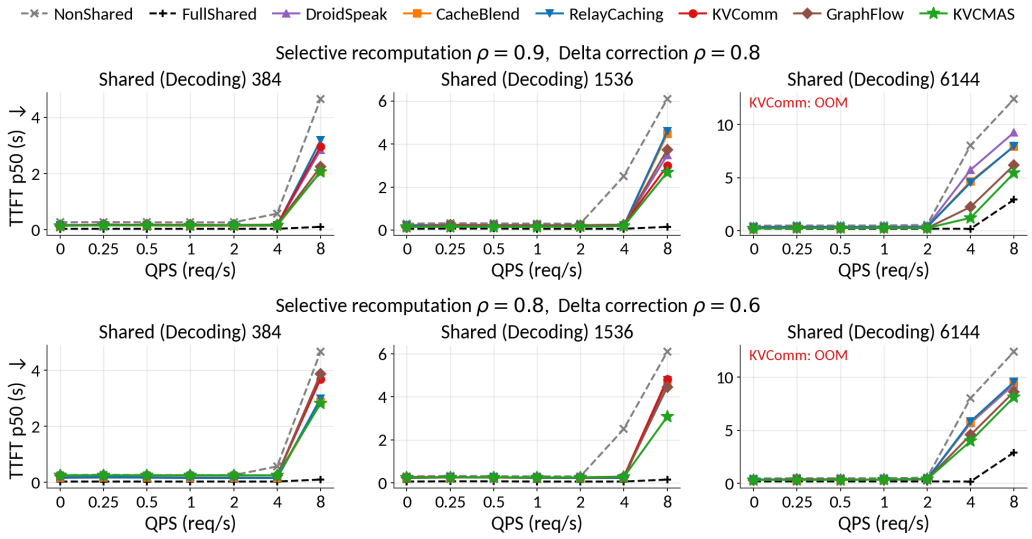

Figure 10: Median (p50) TTFT across request rates with 0.375K, 1.5K, and 6K shared generated tokens. The non-generated shared context is fixed to 2K tokens. Selective recomputation at $\rho = 0.9$ and 0.8 is matched with delta correction at $\rho = 0.8$ and 0.6, respectively.

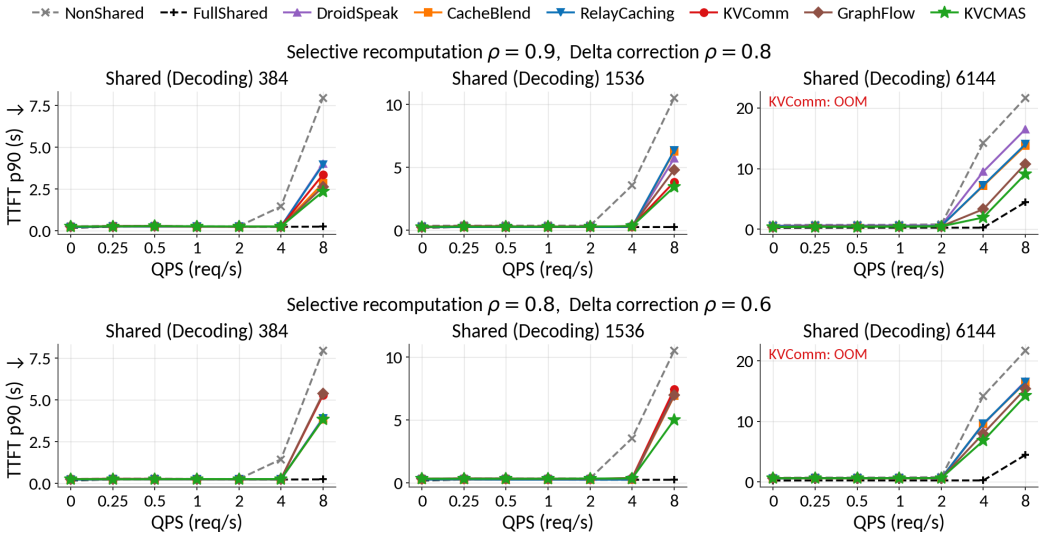

Figure 11: Tail (p90) TTFT under the same accumulated-output configurations as Figure 10.

Table 10: Median (p50) and tail (p90) TTFT in seconds with 1.5K and 6K shared generated tokens. The non-generated shared context is fixed to 2K tokens.

(a) Shared generated context: 1.5k tokens

		$\rho = 0.9$							$\rho = 0.8$							$\rho = 0.6$						
Pct.	**Method**	0	.25	.5	1	2	4	8	0	.25	.5	1	2	4	8	0	.25	.5	1	2	4	8
p50	NonShared	0.30	0.32	0.31	0.30	0.30	2.51	6.10	0.30	0.32	0.31	0.30	0.30	2.51	6.10	0.30	0.32	0.31	0.30	0.30	2.51	6.10
	FullShared	0.06	0.07	0.07	0.06	0.06	0.06	0.15	0.06	0.07	0.07	0.06	0.06	0.06	0.15	0.06	0.07	0.07	0.06	0.06	0.06	0.15
	DroidSpeak	0.23	0.25	0.24	0.23	0.23	0.23	3.48	0.23	0.25	0.25	0.23	0.23	0.23	4.67	0.23	0.25	0.24	0.23	0.23	0.23	4.40
	CacheBlend	0.20	0.21	0.21	0.20	0.20	0.20	4.47	0.23	0.25	0.24	0.23	0.23	0.23	4.58	0.24	0.25	0.25	0.23	0.23	0.24	3.60
	RelayCaching	0.21	0.22	0.22	0.21	0.21	0.21	4.60	0.23	0.25	0.25	0.23	0.23	0.23	4.77	0.24	0.25	0.25	0.23	0.23	0.24	5.12
	KVComm	0.16	0.19	0.19	0.19	0.19	0.19	2.19	0.15	0.24	0.22	0.22	0.22	0.25	2.99	0.27	0.26	0.25	0.25	0.25	0.29	4.83
	GraphFlow	0.15	0.19	0.19	0.19	0.19	0.19	2.15	0.15	0.24	0.22	0.22	0.22	0.25	3.74	0.26	0.26	0.26	0.25	0.25	0.28	4.46
	KVCMAS	0.10	0.10	0.09	0.09	0.09	0.17	1.25	0.15	0.15	0.15	0.15	0.16	0.19	2.69	0.25	0.25	0.25	0.25	0.25	0.27	3.08
p90	NonShared	0.35	0.36	0.36	0.36	0.36	3.56	10.52	0.35	0.36	0.36	0.36	0.36	3.56	10.52	0.35	0.36	0.36	0.36	0.36	3.56	10.52
	FullShared	0.19	0.27	0.26	0.26	0.25	0.25	0.25	0.19	0.27	0.26	0.26	0.25	0.25	0.25	0.19	0.27	0.26	0.26	0.25	0.25	0.25
	DroidSpeak	0.26	0.28	0.27	0.26	0.26	0.26	5.71	0.26	0.28	0.27	0.26	0.26	0.26	6.93	0.26	0.27	0.26	0.26	0.26	0.26	6.63
	CacheBlend	0.24	0.25	0.25	0.25	0.25	0.25	6.27	0.26	0.28	0.26	0.26	0.26	0.26	6.93	0.26	0.28	0.27	0.26	0.26	0.27	5.96
	RelayCaching	0.24	0.25	0.26	0.25	0.25	0.25	6.35	0.26	0.28	0.26	0.26	0.26	0.26	7.16	0.26	0.29	0.27	0.26	0.26	0.27	7.59
	KVComm	0.27	0.28	0.28	0.28	0.26	0.26	2.63	0.27	0.32	0.31	0.28	0.28	0.32	3.84	0.34	0.36	0.33	0.33	0.33	0.39	7.46
	GraphFlow	0.27	0.28	0.28	0.28	0.26	0.26	2.71	0.27	0.32	0.31	0.28	0.28	0.32	4.81	0.34	0.36	0.36	0.33	0.33	0.37	6.98
	KVCMAS	0.26	0.26	0.27	0.27	0.27	0.29	1.73	0.27	0.28	0.28	0.28	0.28	0.32	3.44	0.33	0.33	0.33	0.33	0.35	0.36	5.02

(b) Shared generated context: 6k tokens

		$\rho = 0.9$							$\rho = 0.8$							$\rho = 0.6$						
Pct.	**Method**	0	.25	.5	1	2	4	8	0	.25	.5	1	2	4	8	0	.25	.5	1	2	4	8
p50	NonShared	0.47	0.49	0.49	0.49	0.55	8.03	12.40	0.47	0.49	0.49	0.49	0.55	8.03	12.40	0.47	0.49	0.49	0.49	0.55	8.03	12.40
	FullShared	0.19	0.19	0.19	0.19	0.19	0.18	2.92	0.19	0.19	0.19	0.19	0.19	0.18	2.92	0.19	0.19	0.19	0.19	0.19	0.18	2.92
	DroidSpeak	0.37	0.38	0.38	0.36	0.41	5.73	9.25	0.37	0.38	0.36	0.36	0.41	5.67	9.32	0.37	0.38	0.38	0.36	0.40	5.46	9.21
	CacheBlend	0.31	0.32	0.32	0.31	0.33	4.65	7.93	0.37	0.38	0.38	0.36	0.40	5.75	9.50	0.38	0.40	0.40	0.38	0.42	5.66	9.62
	RelayCaching	0.32	0.34	0.33	0.32	0.34	4.54	7.96	0.37	0.39	0.38	0.36	0.41	5.83	9.55	0.38	0.40	0.40	0.38	0.43	5.74	9.66
	GraphFlow	0.22	0.25	0.25	0.25	0.24	1.24	4.79	0.22	0.25	0.25	0.25	0.25	2.24	6.17	0.32	0.39	0.37	0.37	0.40	4.57	8.62
	KVCMAS	0.19	0.19	0.20	0.21	0.22	0.29	4.04	0.22	0.24	0.25	0.25	0.25	1.21	5.42	0.32	0.33	0.33	0.35	0.39	3.99	8.12
p90	NonShared	0.68	0.72	0.72	0.73	0.80	14.20	21.66	0.68	0.72	0.72	0.73	0.80	14.20	21.66	0.68	0.72	0.72	0.73	0.80	14.20	21.66
	FullShared	0.23	0.25	0.25	0.25	0.25	0.25	4.46	0.23	0.25	0.25	0.25	0.25	0.25	4.46	0.23	0.25	0.25	0.25	0.25	0.25	4.46
	DroidSpeak	0.57	0.62	0.61	0.62	0.62	9.51	16.51	0.57	0.63	0.61	0.62	0.62	9.50	16.64	0.57	0.62	0.61	0.61	0.62	9.20	16.43
	CacheBlend	0.48	0.53	0.53	0.53	0.53	7.17	13.81	0.57	0.62	0.61	0.61	0.62	9.61	16.40	0.59	0.64	0.64	0.64	0.64	9.53	17.12
	RelayCaching	0.49	0.53	0.53	0.53	0.53	7.26	14.02	0.57	0.64	0.62	0.62	0.62	9.61	16.49	0.59	0.66	0.64	0.64	0.64	9.65	17.20
	GraphFlow	0.35	0.38	0.36	0.36	0.39	1.85	7.88	0.35	0.49	0.46	0.46	0.47	3.30	10.79	0.63	0.63	0.60	0.60	0.61	8.00	15.38
	KVCMAS	0.35	0.37	0.37	0.37	0.39	0.66	6.42	0.35	0.39	0.40	0.41	0.47	1.93	9.10	0.61	0.61	0.61	0.62	0.63	6.80	14.22

C Ablation Studies

C.1 Per-Request Efficiency

Section 4.3 evaluates agent-request TTFT under concurrent serving. Here, we measure memory and per-request efficiency under single-stream execution on an NVIDIA A6000 48 GB GPU using Llama-3.1-8B, three agents, and 128 generated tokens per agent. The workflow contains an initial dense-prefill request followed by two reuse requests. TTFT is averaged over the reuse requests, while throughput divides the total trajectory tokens by E2E latency.

Memory Usage. Table 11 reports peak GPU memory across shared context lengths and reuse ratios. KVComm exceeds the A6000 capacity because its full-dimensional anchor pool is stored alongside the model and active KV cache, so we estimate its requirement from the base caches and delta corrections. At 2K shared tokens, KVComm runs out of memory due to memory fragmentation. At 2K, 8K, and 32K, these estimates are 2.5×, 6.2×, and 14.9× those of KVCMAS, respectively. KVCMAS maintains peak memory comparable to the selective recomputation methods and nearly unchanged across reuse ratios because it maintains a fixed number of low-rank anchors rather than full-dimensional recomputation states.

Table 11: Peak GPU memory in GB under single-stream execution across shared context lengths and reuse ratios. KVComm† exceeds the A6000 capacity, so its requirement is estimated from its full-dimensional anchor states.

Shared context	Method	Reuse ratio ρ				
		0.95	**0.90**	**0.80**	**0.70**	**0.60**
2K	NonShared	16.9	16.9	16.9	16.9	16.9
	FullShared	16.4	16.4	16.4	16.4	16.4
	DroidSpeak	17.0	17.0	17.0	17.1	17.2
	CacheBlend	17.2	17.2	17.2	17.3	17.4
	RelayCaching	17.4	17.4	17.4	17.5	17.5
	KVComm†	42.3	42.3	42.3	42.3	42.3
	GraphFlow	16.8	16.8	16.8	16.8	16.8
	KVCMAS	**17.1**	**17.1**	**17.1**	**17.1**	**17.1**
8K	NonShared	19.8	19.8	19.8	19.8	19.8
	FullShared	17.9	17.9	17.9	17.9	17.9
	DroidSpeak	19.3	19.3	19.5	19.8	20.0
	CacheBlend	19.4	19.5	19.7	19.9	20.1
	RelayCaching	19.6	19.7	19.8	19.9	20.0
	KVComm†	120.9	120.9	120.9	120.9	120.9
	GraphFlow	19.1	19.1	19.1	19.1	19.1
	KVCMAS	**19.5**	**19.5**	**19.5**	**19.5**	**19.5**
32K	NonShared	31.5	31.5	31.5	31.5	31.5
	FullShared	24.0	24.0	24.0	24.0	24.0
	DroidSpeak	28.6	28.7	29.5	30.6	31.4
	CacheBlend	28.5	28.9	29.7	30.5	31.3
	RelayCaching	28.5	28.7	29.1	29.5	29.9
	KVComm†	435.6	435.6	435.6	435.6	435.6
	GraphFlow	28.9	28.9	28.9	28.9	28.9
	KVCMAS	**29.2**	**29.2**	**29.2**	**29.2**	**29.2**

TTFT and Throughput. Table 12 reports the numerical results, while Figure 12 compares the methods at matched reuse ratios. At 8K and 32K, KVCMAS achieves the lowest TTFT and highest throughput among the KV cache sharing works, with larger gains as the shared context grows. At 2K, where prefill contributes less to execution, it remains comparable to the fastest work. These results demonstrate its efficiency in single-batch serving, including settings relevant to edge deployments, particularly for context-heavy workloads.

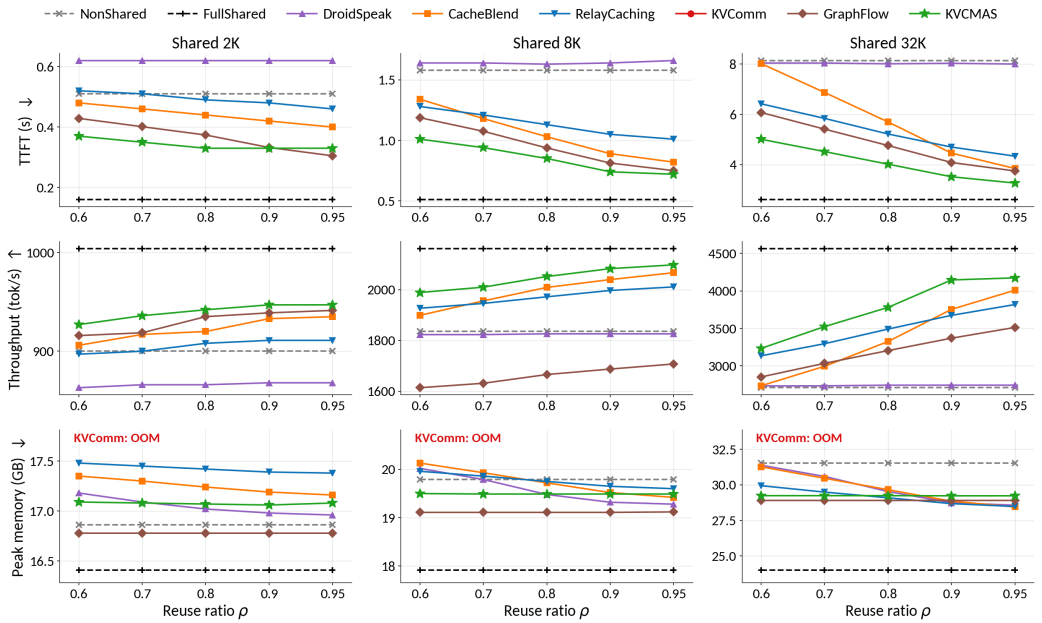

Figure 12: Single-stream TTFT, E2E throughput, and peak GPU memory across shared context lengths at matched reuse ratios. Lower TTFT and memory and higher throughput indicate better efficiency. KVComm exceeds the NVIDIA A6000 48 GB memory capacity.

Table 12: Average reuse-request TTFT and E2E trajectory throughput under single-stream execution. TTFT is reported in seconds, and throughput in tokens/s.

Shared context	Method	TTFT (s)					Throughput (tokens/s)				
		ρ					ρ				
		0.95	**0.90**	**0.80**	**0.70**	**0.60**	**0.95**	**0.90**	**0.80**	**0.70**	**0.60**
2K	NonShared	0.51	0.51	0.51	0.51	0.51	900	900	900	900	900
	FullShared	0.16	0.16	0.16	0.16	0.16	1,004	1,004	1,004	1,004	1,004
	DroidSpeak	0.62	0.62	0.62	0.62	0.62	868	868	866	866	863
	CacheBlend	0.40	0.42	0.44	0.46	0.48	935	933	920	917	906
	RelayCaching	0.46	0.48	0.49	0.51	0.52	911	911	908	900	897
	GraphFlow	0.31	0.33	0.37	0.40	0.43	941	939	935	919	916
	KVCMAS	**0.33**	**0.33**	**0.33**	**0.35**	**0.37**	**947**	**947**	**942**	**936**	**927**
8K	NonShared	1.58	1.58	1.58	1.58	1.58	1,836	1,836	1,836	1,836	1,836
	FullShared	0.51	0.51	0.51	0.51	0.51	2,162	2,162	2,162	2,162	2,162
	DroidSpeak	1.66	1.64	1.63	1.64	1.64	1,826	1,826	1,826	1,823	1,823
	CacheBlend	0.82	0.89	1.03	1.18	1.34	2,067	2,040	2,009	1,956	1,899
	RelayCaching	1.01	1.05	1.13	1.21	1.28	2,011	1,997	1,972	1,946	1,927
	GraphFlow	0.75	0.81	0.94	1.08	1.19	1,707	1,687	1,666	1,631	1,614
	KVCMAS	**0.72**	**0.74**	**0.85**	**0.94**	**1.01**	**2,098**	**2,083**	**2,052**	**2,010**	**1,989**
32K	NonShared	8.14	8.14	8.14	8.14	8.14	2,709	2,709	2,709	2,709	2,709
	FullShared	2.61	2.61	2.61	2.61	2.61	4,564	4,564	4,564	4,564	4,564
	DroidSpeak	8.00	8.03	8.01	8.04	8.04	2,743	2,743	2,743	2,734	2,733
	CacheBlend	3.85	4.46	5.70	6.87	8.02	4,009	3,754	3,325	2,995	2,735
	RelayCaching	4.34	4.70	5.22	5.84	6.42	3,818	3,674	3,491	3,296	3,135
	GraphFlow	3.75	4.09	4.76	5.41	6.08	3,513	3,370	3,204	3,034	2,853
	KVCMAS	**3.27**	**3.52**	**4.02**	**4.52**	**5.01**	**4,174**	**4,146**	**3,780**	**3,524**	**3,234**

C.2 Ranks of Base Cache and Delta Correction

Section 3.1 represents the source cache and delta correction using low-rank factors with ranks r_C and r_Δ, respectively. We vary them independently and evaluate accuracy on MMLU and Video-MME using their benchmark-specific reliability thresholds, adjusting τ for each rank configuration to maintain the target reuse regime of around $\rho = 0.8$ and 0.6 for the LLM and VLM tasks, respectively. In addition, we measure TTFT, throughput, and peak GPU memory with Llama-3.1-8B under the 32K single-stream setting with three agents and $\rho = 0.8$. Figure 13 reports the accuracy and efficiency results, where "Full" denotes the corresponding uncompressed KV feature dimension.

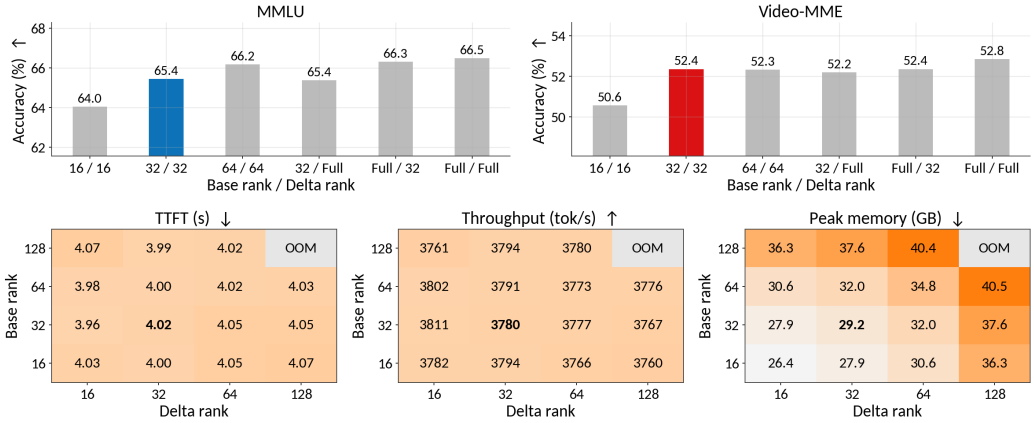

Figure 13: Effect of the base cache and delta correction ranks on accuracy and single-stream efficiency. The upper panels report accuracy on MMLU and Video-MME, while the lower panels show TTFT, throughput, and peak GPU memory across rank combinations. "Full" denotes the corresponding uncompressed KV feature dimension, and the default $(r_C, r_\Delta) = (32, 32)$ is highlighted.

Increasing both ranks from 16 to 32 improves accuracy on MMLU and Video-MME. Beyond rank 32, accuracy changes marginally while memory usage continues to increase, and the $(128, 128)$ configuration exceeds GPU capacity. TTFT and throughput remain nearly unchanged among configurations that fit in memory. We therefore use $r_C = r_\Delta = 32$, which captures most of the full-dimensional accuracy while maintaining a compact anchor pool.

C.3 ANCHOR POOL SIZE

KVCMAS maintains a process-wide anchor pool for each shared placeholder slot. Each anchor stores a low-rank base cache and consumer-specific corrections, and its capacity V is applied independently to each slot. While Section 3.3 describes one source–target transition for clarity, the implementation shares these slot-specific pools across requests. We vary V under the single-stream setting with 32K shared tokens, three agents, rank $r = 32$, and $\rho = 0.8$. KVComm uses $V = 20$ as its original balance between accuracy and efficiency, while KVCMAS uses $V = 10$ by default. For accuracy, we adjust τ for each pool capacity to maintain approximately $\rho = 0.8$ on MMLU and $\rho = 0.6$ on Video-MME. Because entropy-based reliability requires at least two compatible anchors, the $V = 1$ configuration bypasses the reliability gate and directly uses the sole anchor.

Table 13: Effect of anchor pool capacity V on single-stream efficiency and accuracy. TTFT is reported in seconds, throughput in tokens/s, peak GPU memory in GB, and accuracy in percent.

V	**TTFT (s)**	**Throughput**	**Memory**	**MMLU**	**Video-MME**
1	2.69	4,050	26.21	62.51	49.40
5	3.93	3,821	26.68	64.42	51.89
10	4.02	3,780	29.24	65.45	52.36
20	4.05	3,769	31.94	65.80	53.15

A larger pool provides more representative anchors and improves correction accuracy, with $V = 10$ capturing most of the gain. Increasing V from 10 to 20 provides only marginal additional accuracy with higher memory usage, while TTFT and throughput change little beyond $V = 5$. Even at the matched capacity of $V = 20$, KVCMAS uses 31.94 GB, compared with the estimated 435.6 GB required by KVComm under the same 32K configuration, isolating the low-rank memory reduction from the difference in their default pool capacities. We therefore use $V = 10$ as a balanced default.

C.4 Scaling with the Number of Agents

Using Llama-3.1-8B on an NVIDIA A6000 48 GB GPU, we vary the number of agents $N \in \{2, 4, 8\}$ with 32K shared tokens, one interaction round, and 128 generated tokens per agent under single-stream execution. TTFT is averaged across all N agent requests, while throughput covers the full trajectory. Selective recomputation uses $\rho = 0.9$, and delta correction uses $\rho = 0.8$. Because the fixed 32K context dominates the trajectory length, increasing N primarily creates more opportunities to reuse the same context.

Table 14: Single-stream efficiency as the number of agents increases. Selective recomputation uses $\rho = 0.9$, and delta correction uses $\rho = 0.8$. KVComm is omitted because its anchor pool exceeds GPU capacity at 32K shared tokens. OOM denotes out of memory.

	TTFT (s)			**Throughput (token/s)**			**Memory (GB)**		
Method	$N=2$	$N=4$	$N=8$	$N=2$	$N=4$	$N=8$	$N=2$	$N=4$	$N=8$
NonShared	8.25	8.37	OOM	2,729	2,760	OOM	27.25	35.63	OOM
FullShared	3.87	2.01	1.08	3,830	4,922	5,904	23.06	23.06	23.06
DroidSpeak	6.82	6.19	5.91	2,657	2,852	3,021	27.12	27.93	29.64
CacheBlend	5.63	4.49	3.90	**3,627**	3,900	4,153	27.03	27.79	29.54
RelayCaching	5.53	4.28	3.65	3,480	3,883	4,093	26.61	27.22	28.44
GraphFlow	5.71	3.97	3.10	3,075	3,860	4,244	28.80	29.82	31.86
KVCMAS	**5.11**	**3.50**	**2.70**	3,546	**4,327**	**4,776**	29.07	29.39	30.03

KVCMAS achieves the lowest TTFT among the KV cache sharing works across all agent counts, with a larger advantage as more agents reuse the shared context. It also provides the highest throughput at $N = 4$ and $N = 8$ and remains close to the highest at $N = 2$. More agents amortize the initial dense prefill across additional reuse requests, reducing average TTFT. Peak memory increases only marginally because corrections for additional agent transitions are stored as compact low-rank anchor states.

C.5 Scaling with the Number of Rounds

Using Llama-3.1-8B on an NVIDIA A6000 48 GB GPU, we vary the number of interaction rounds $R \in \{2, 4, 8\}$ with three agents, 32K initial shared tokens, and 128 generated tokens per agent under single-stream execution. TTFT is averaged across all agent requests, while throughput covers the full trajectory. The fixed 32K context remains dominant as additional rounds create more opportunities to reuse the accumulated context. Selective recomputation uses $\rho = 0.9$, and delta correction uses $\rho = 0.8$.

Table 15: Single-stream efficiency as the number of interaction rounds increases. Selective recomputation uses $\rho = 0.9$, and delta correction uses $\rho = 0.8$. KVComm is omitted because its anchor pool exceeds GPU capacity at 32K shared tokens.

	TTFT (s)			**Throughput (token/s)**			**Memory (GB)**		
Method	$R = 2$	$R = 4$	$R = 8$	$R = 2$	$R = 4$	$R = 8$	$R = 2$	$R = 4$	$R = 8$
NonShared	8.22	8.33	8.19	2,728	2,747	2,769	31.44	31.44	31.44
FullShared	1.39	0.74	0.43	5,421	6,445	6,948	23.06	23.06	23.06
DroidSpeak	5.87	5.61	5.54	2,756	2,777	2,805	27.66	27.75	28.03
CacheBlend	4.03	3.57	3.36	3,927	4,110	4,278	27.50	27.65	27.90
RelayCaching	3.75	3.37	3.08	3,824	3,972	4,138	27.00	27.16	27.38
GraphFlow	3.04	2.48	2.21	3,926	4,248	4,582	29.36	29.52	29.70
KVCMAS	**2.87**	**2.30**	**2.02**	**4,198**	**4,485**	**4,802**	29.33	29.49	29.77

KVCMAS achieves the lowest TTFT and highest throughput among the KV cache sharing works across all evaluated round counts. Additional rounds amortize the initial dense prefill across more reuse requests, reducing average TTFT and increasing throughput. Peak GPU memory increases slightly because the growing trajectory adds compact low-rank anchor states for newly accumulated agent outputs.