Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.

ICML
International Conference
On Machine Learning

SEOUL NATIONAL UNIVERSITY

LRAgent:
Efficient KV Cache Sharing for Multi-LoRA LLM Agents

Hyesung Jeon[1], Hyeongju Ha[1], Jae-Joon Kim [1†]

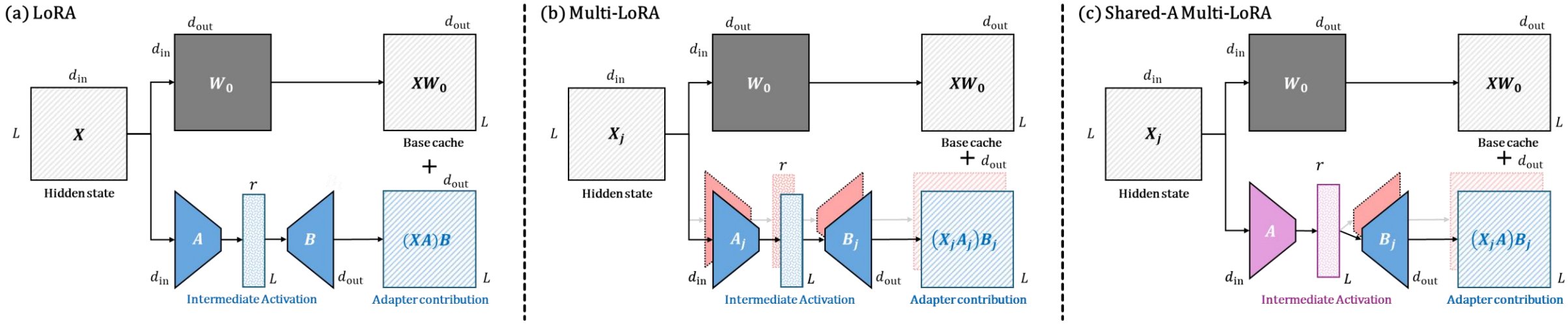
[1] Seoul National University, Dept. of ECE

Background

Multi-LoRA Agents and KV Cache Sharing

Although agent role specialization can be efficiently implemented on top of a shared backbone, the KV cache still grows proportionally with both the trajectory length and the number of agents.

Multi-LoRA Architecture



- Due to its training and inference efficiency,
  the multi-LoRA architecture is often utilized for multi-role LLM agents [AutoGen, Reflexion, AutoAct]
- Shared-A multi-LoRA enables stronger model accuracy [MTL-LoRA, Hydra-LoRA]
  - Enhanced generalizability through shared down-projection A
  - Task-specific differences effectively captured by up-projection B

Multi-LLM Agent Trajectory

- Multi-LLM agents see the same accumulative trajectory
  - $\times NL$ computations for previously processed contexts
  - $\times NL$ memory usage for KV cache

- Typical multi-LLM agent roles

  PLAN

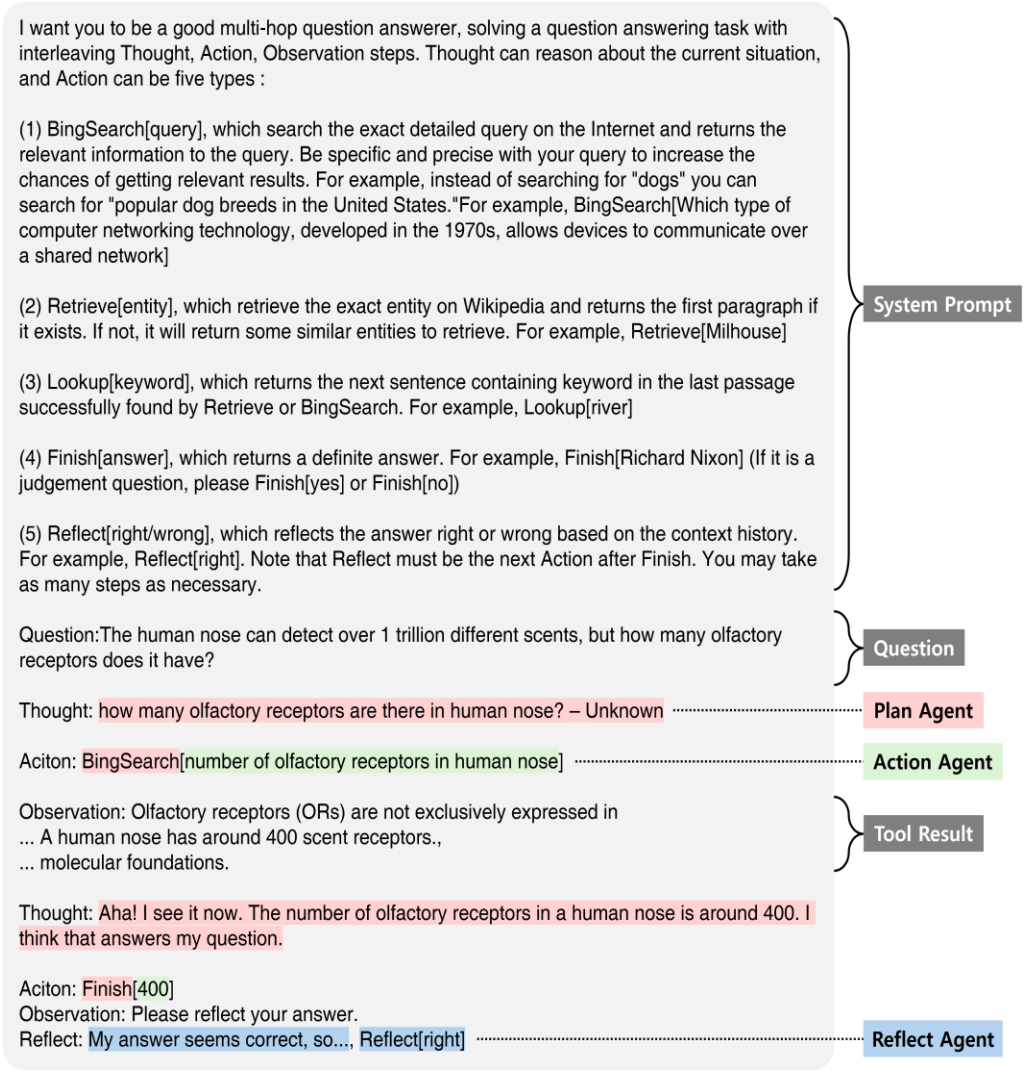
  Reasons and decide about what to do next.
  (e.g. web search, retrieve, look up a keyword, or finish)

  ACTION

  Returns a tool call parameter given the plan agent's tool call.
  (e.g. WebSearch target, Retrieve target, Finish answer)

  REFLECT

  Inspects the finished answer in context and decides
  whether to accept it or hand back to the plan agent.



Multi-LLM Agent KV Cache Sharing

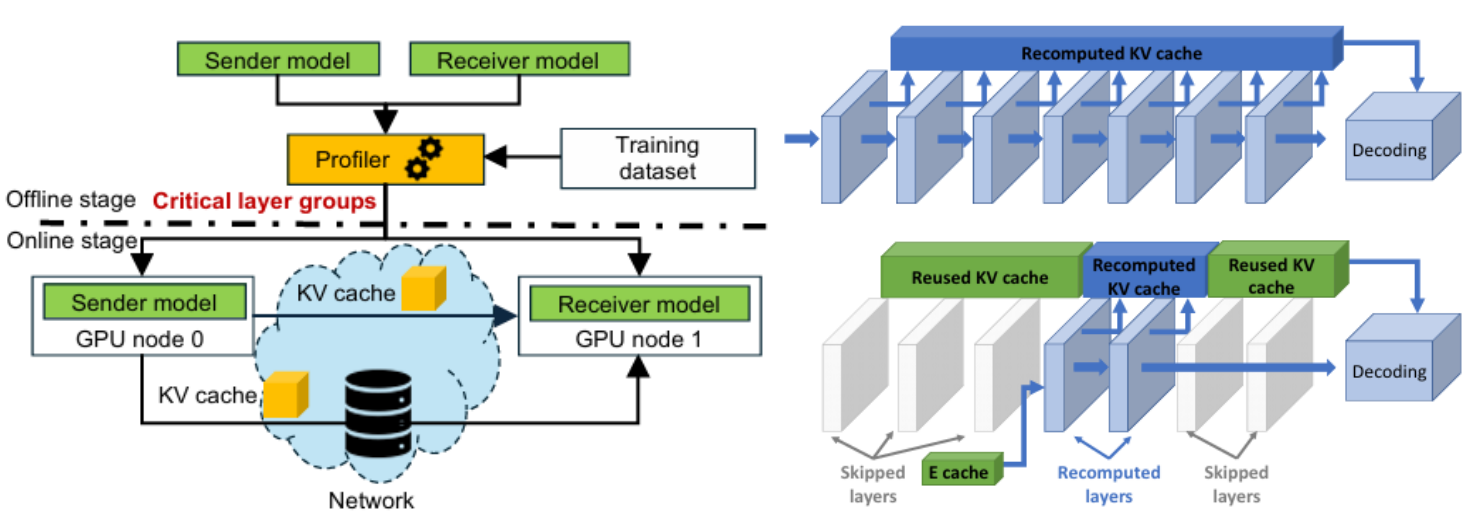


Diagram on KV cache transfer across the agents and recomputation in DroidSpeak [1]

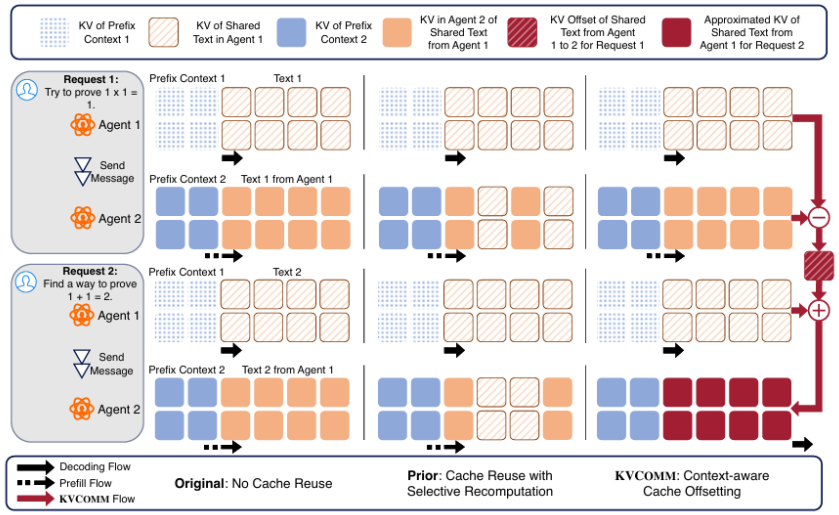


Diagram on KV cache offsetting in KVComm [2]

Family	Representative	Key Assumptions	Implications
Architectural Modification	Cache2Cache, ICaRus	Re-train the model so caches are fusible across agents.	Requires non-trivial re-training, not a drop-in.
Positional Re-Alignment	KVLink, KVFlow, KVComm	Different agents see different prefixes and need re-positioning.	Limited to positional recovery, which collapses to naïve fully shared schemes when prefix are shared.
Selective Recomputation	CacheBlend, DroidSpeak	A subset of tokens or layers needs to be recomputed for each agent.	Re-runs hidden states of previous contexts thereby has limited computational reduction.
Multi-LoRA-aware Design	LRAgent	The cross-agent variation is mainly due to the heterogeneous low-rank adapter output.	Reduction in both memory usage and computation without re-training.

[1] DroidSpeak: KV Cache Sharing for Cross-LLM Communication and Multi-LLM Serving, Yuhan Liu et al., NSDI 2024.
[2] KVComm: Enabling Efficient LLM Communication through Selective KV Sharing, Xiangyu Shi et al., ICLR 2026.

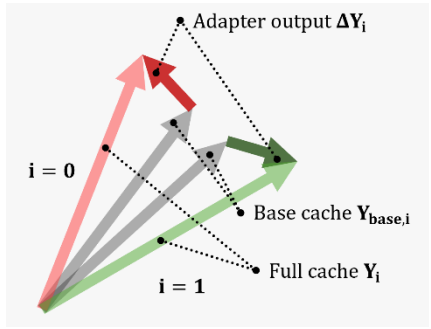
Motivation

Large and Similar Backbone Outputs vs. Small and Decorrelated Adapter Outputs:

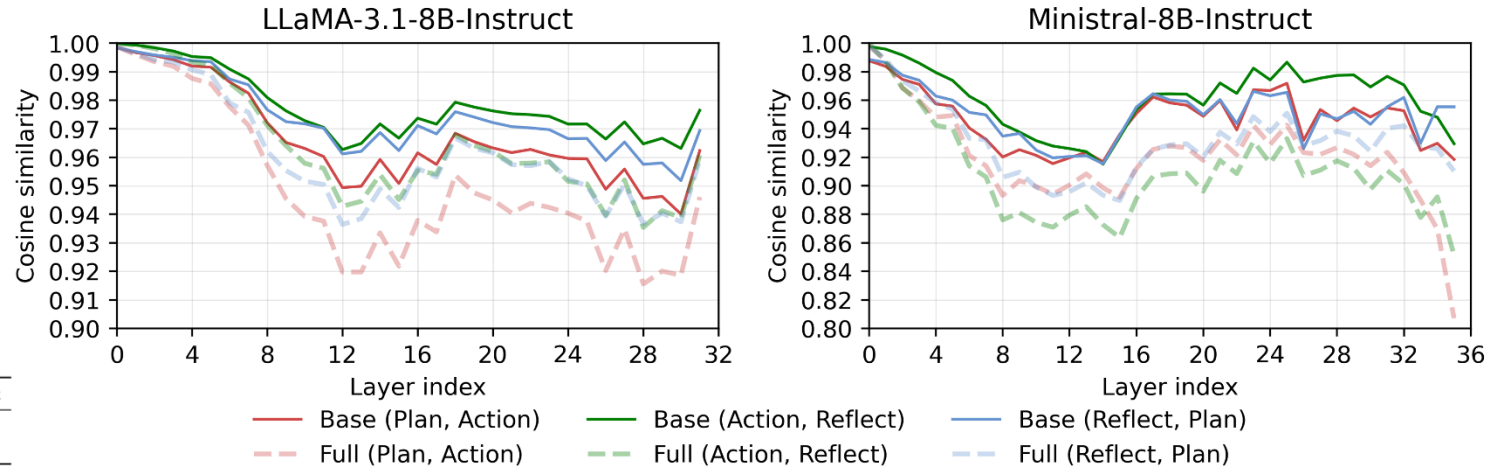
Observations on decoupled KV caches into base caches (XW_0) and the adapter outputs (XAB).

Base Cache and Adapter Output Similarity

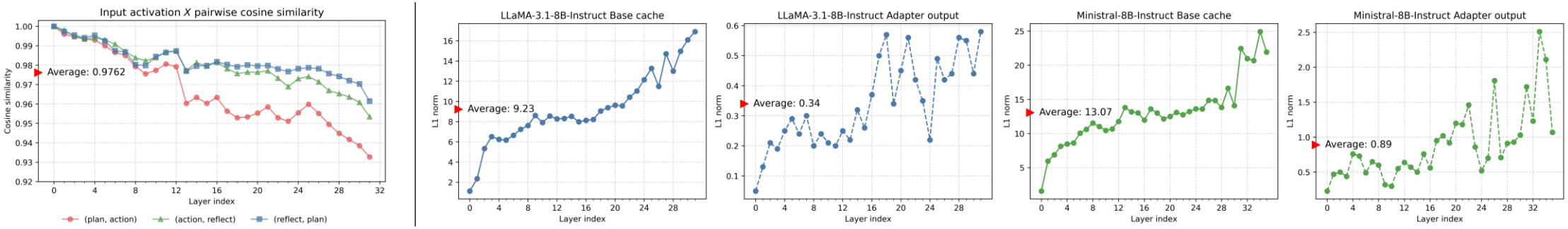
- Base cache $Y_{base,i} = X_i W_0$ exhibits high similarity across the agents despite the hidden state diff.
- Adapter output $\Delta Y_i = X_i \Delta W_i$ acts as a small, decorrelated perturbation.



Model	Full cache	Base cache	Adapter output
LLaMA-3.1-8B-Instruct	0.9576	0.9726	0.0538
Ministral-8B-Instruct	0.9200	0.9530	0.0225



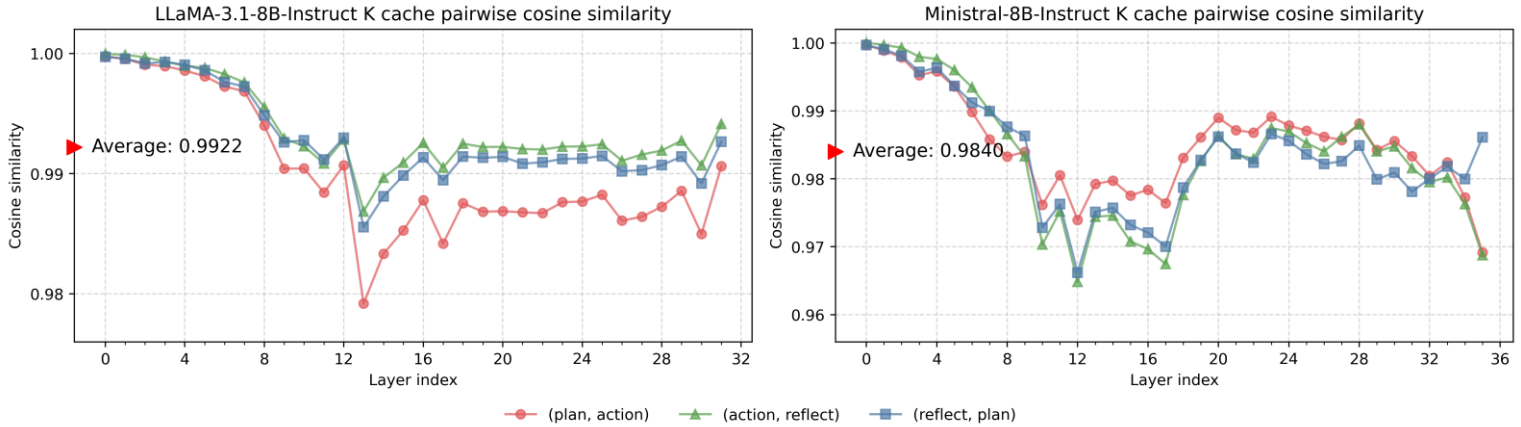
(Left) Diagram of full cache, base cache, and adapter output. (Right) Layer-wise pairwise cosine similarity of the base and full caches



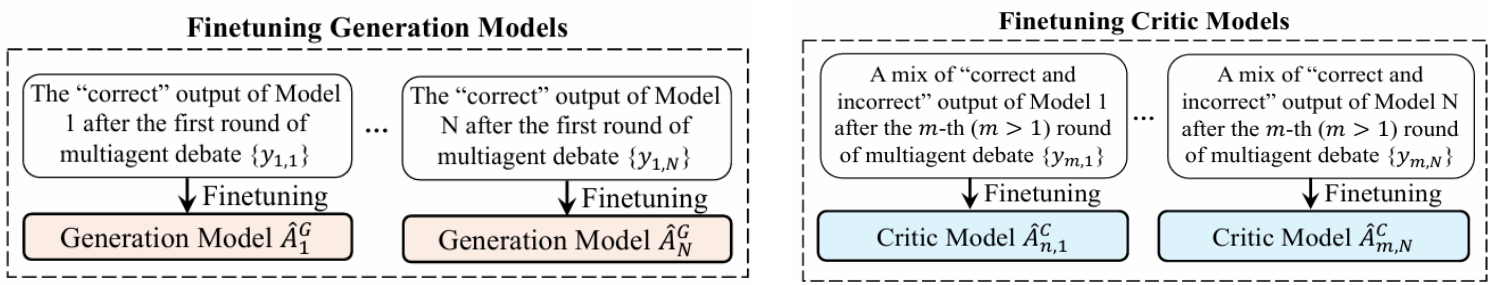
(Left) Layer-wise pairwise hidden state cosine similarity. (Right) Layer-wise base cache and adapter output L1 norm comparison

Further Observations on the Cache Similarity

- With typical LoRA application on Q,V (best accuracy), K cache exhibits sufficiently high similarity.
- Sim(base cache) > Sim(full cache) holds for various agent role scenarios.



Layer-wise pairwise cosine similarity of the K cache on LLaMA and Mistral models



Contribution	Full cache	Base cache	LR cache
Cosine Similarity	0.9553	0.9766	0.9620

Cache similarity in debate-style generation-critic agent framework [3]

[3] Multiagent Finetuning: Self Improvement with Diverse Reasoning Chains, Vighnesh Subramaniam et al., ICLR 2025.

Methodology

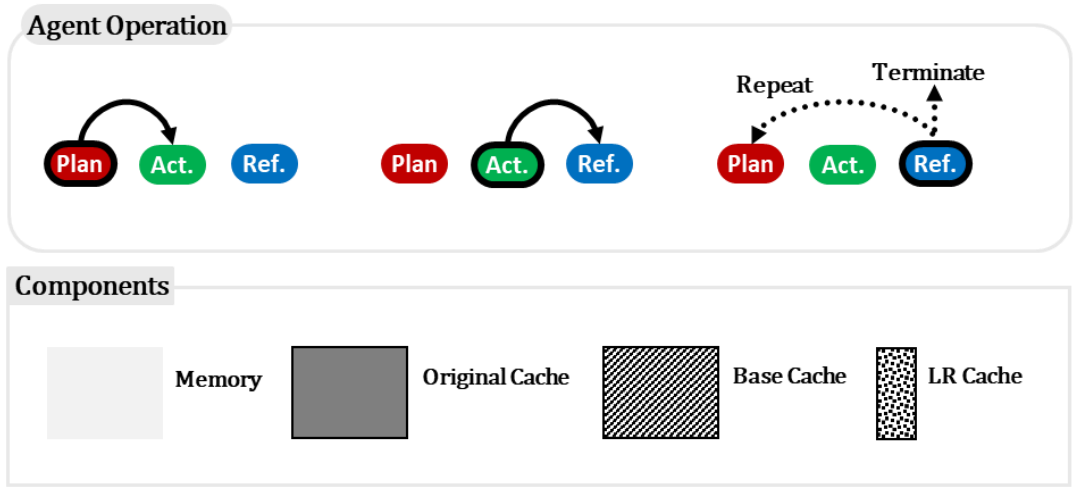
Share the Base, Track the Small Differences

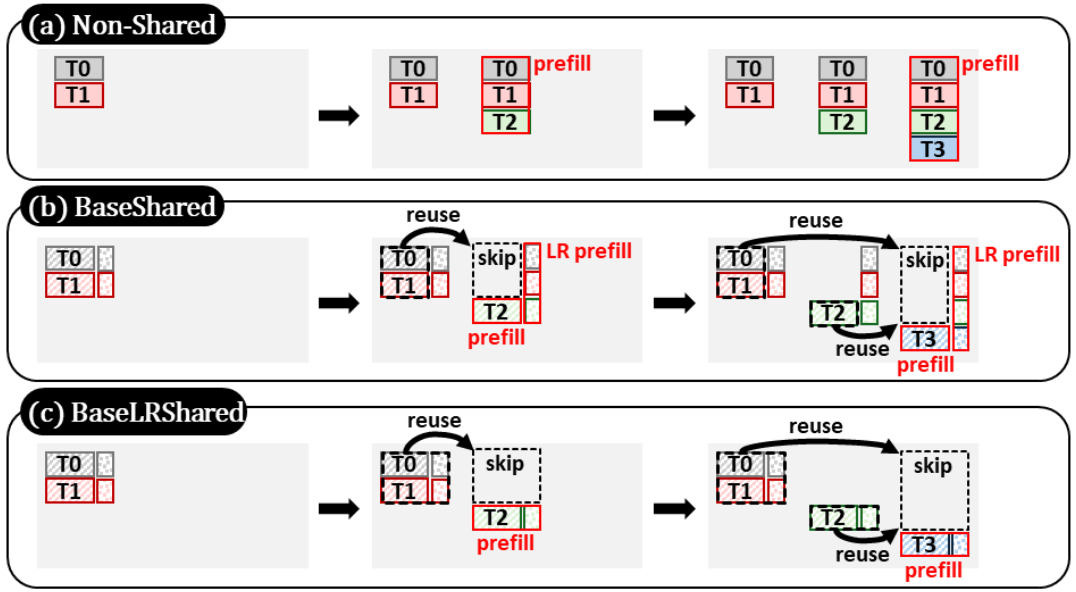
The main bottleneck arising when multiple LoRA adapters process the same trajectory can be resolved by sharing the single base cache and keep tracking of each adapter output which stored in its LR cache form.

LR Cache Management Strategies

- Base cache
  - Built once, reused by proceeding agents.
- LR cache
  - Kept in its native low-rank form.
  - Materialized to full dimension at runtime.

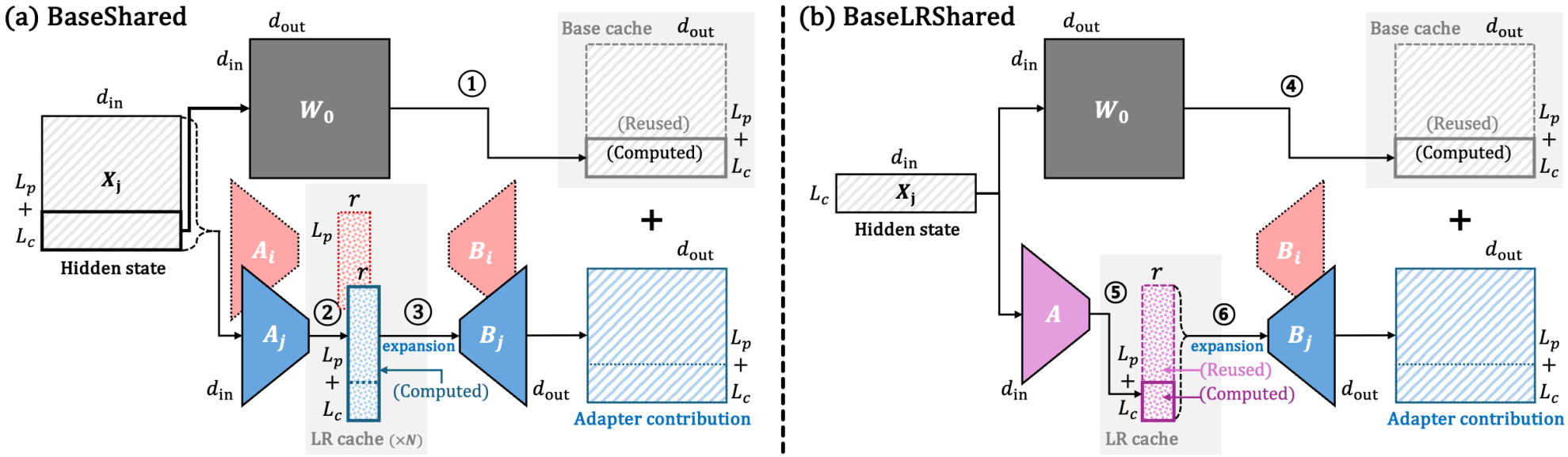
→ No extra compression / projection / parameters !





Agent execution and cache accumulation diagram on naïve fully-shared scheme (Non-Shared) and our three proposed methods.
Depending on the LR cache management strategy, we propose three KV cache sharing methods: BaseShared and BaseLRShared.

BaseShared and BaseLRShared



BaseShared

The base cache is shared across agents, but each agent still recomputes its own LR cache on its own hidden states for the new tokens.

Saves memory to $O(1)$, but still computation $O(NL^2)$

Drop-in into existing multi-LoRA architectures

Highly accurate; freshly generated hidden states

BaseLRShared

Under shared-A multi-LoRA, the LR cache is also shared across agents together with the base cache. Each agent's contribution is reconstructed by multiplying with its own B_i.

Saves memory to $O(1)$, computation $O(L^2)$

Leverages advanced shared-A multi-LoRA architecture

High computational efficiency

Flash-LoRA-Attention for **LR Cache Materialization**

- Efficient LR cache materialization via matrix multiplication associativity

Naïve - Materialize then Attend

$$O = P \cdot (V_{base} + V_{lr} \cdot B)$$

Build the full $L \times d_{out}$ adapter output, and then run Flash-Attention over it.

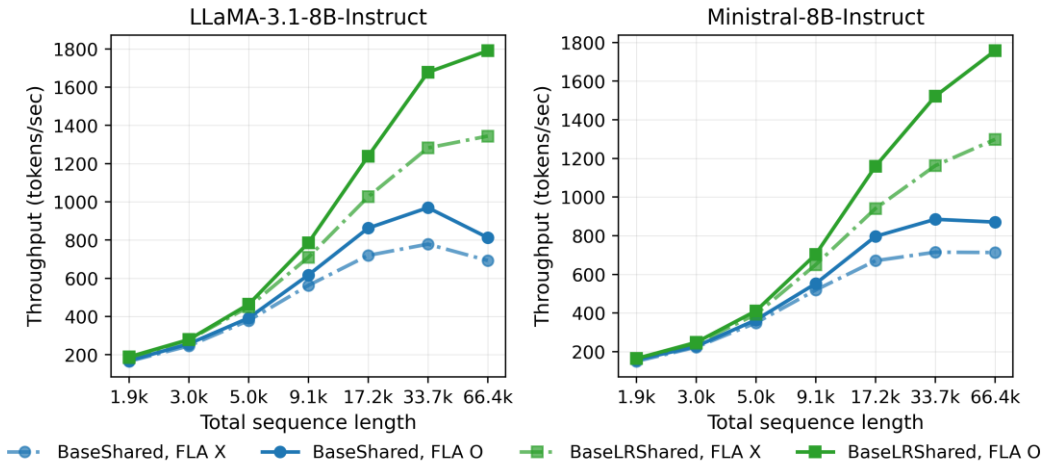
Requires $\Theta(Lrd)$ computation.

FLA - Attend then Expand

$$O = P \cdot V_{base} + (P \cdot V_{lr}) \cdot B$$

Attend in rank-r dimension first, multiply by B once at the end of each Q-block in Flash-Attention.

Reduced to $\Theta(Lr + rd)$ computation.
→ Up to 1.35x throughput gain !



End-to-end System throughput (tokens/sec) with and without Flash-LoRA-Attention (FLA)

Algorithm 1 Flash-LoRA-Attention Forward (head-wise)

Require: Accumulated context length $L = L_p + L_c$.
Require: Query $Q \in \mathbb{R}^{L_c \times d_{head}}$, key $K \in \mathbb{R}^{L \times d_{head}}$ in HBM.
Require: Base value cache $V_{base} \in \mathbb{R}^{L \times d_{head}}$ in HBM.
Require: LR value cache $V_{lr} \in \mathbb{R}^{L \times r}$, LoRA $B \in \mathbb{R}^{r \times d_{head}}$ in HBM, $r \ll d_{head}$.
Require: Block sizes B_r, B_c; $T_r = \lceil L_c / B_r \rceil$, $T_c = \lceil L / B_c \rceil$.
Ensure: Attention output $O \in \mathbb{R}^{L_c \times d_{head}}$ in HBM.

Split Q, O into T_r blocks $\{Q_i, O_i\}$ of size $B_r \times d_{head}$.
Split K, V_{base} into T_c blocks $\{K_j, V_{base,j}\}$ of size $B_c \times d_{head}$.
Split V_{lr} into T_c blocks $\{V_{lr,j}\}$ of size $B_c \times r$.
for $i = 0, 1, \ldots, T_r - 1$ **do**
  Load Q_i to SRAM.
  Init $O_i \leftarrow 0 \in \mathbb{R}^{B_r \times d_{head}}$, $O_{lr,i} \leftarrow 0 \in \mathbb{R}^{B_r \times r}$
  Init $m_i \leftarrow -\infty \in \mathbb{R}^{B_r}$, $\ell_i \leftarrow 0 \in \mathbb{R}^{B_r}$
  for $j = 0, 1, \ldots, T_c - 1$ **do**
    Load $K_j, V_{base,j}, V_{lr,j}$ to SRAM.
    $S_{ij} \leftarrow \mathrm{Mask}(Q_i K_j^\top / \sqrt{d_{head}})$
    $m_i^{new} \leftarrow \max(m_i, \mathrm{rowmax}(S_{ij}))$
    $\alpha \leftarrow \exp(m_i - m_i^{new})$
    $P_{ij} \leftarrow \exp(S_{ij} - m_i^{new})$
    $\ell_i \leftarrow \alpha \odot \ell_i + \mathrm{rowsum}(P_{ij})$
    $O_i \leftarrow \alpha \odot O_i + P_{ij} V_{base,j}$
    $O_{lr,i} \leftarrow \alpha \odot O_{lr,i} + P_{ij} V_{lr,j}$
    $m_i \leftarrow m_i^{new}$
  end for
  $O_i \leftarrow O_i + O_{lr,i} B$
  $O_i \leftarrow O_i / \ell_i$
  Write O_i to O in HBM.
end for

Experiments

Minimized Accuracy Degradation with Maximized Throughput

The three proposed methods achieves up to 2.46x throughput gain with ~1% of accuracy degradation, which dominates the non-LoRA-aware KV cache sharing schemes in both accuracy and throughput.

Agentic QA Benchmark Accuracy

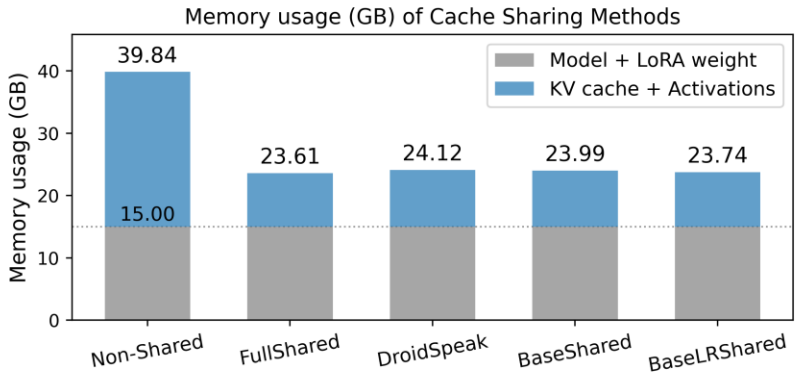
- BaseShared stays close to Non-Shared
  - Derives hidden states & LR cache only from the current agent.
- BaseLRShared exhibit slightly lower accuracy, but still maintain dominance over others.
  - Derives LR cache from hidden states of other agent.
- Hence, decoupling and sharing the base cache or LR cache is the key factor to maintain the agent accuracy.

		HotpotQA				ScienceQA			
Model	Method	Easy	Medium	Hard	Avg.	1-4	5-8	9-12	Avg.
LLaMA-3.1-8B-Instruct	Non-Shared	42.80	41.95	31.90	38.88 0.00	70.63	60.54	76.75	69.31 0.00
	FullShared	41.15	39.15	28.90	36.40 -2.48	68.00	55.67	72.00	65.22 -4.08
	DroidSpeak	40.60	39.55	30.15	36.77 -2.12	68.79	59.25	74.42	67.49 -1.82
	BaseShared	42.70	41.95	31.15	38.60 -0.28	70.38	60.75	76.58	69.24 -0.07
	BaseLRShared	42.40	40.80	30.55	37.92 -0.97	70.42	60.25	76.71	69.13 -0.18
Ministral-8B-Instruct	Non-Shared	41.30	37.75	28.75	35.93 0.00	71.50	63.83	70.92	68.75 0.00
	FullShared	39.60	33.95	24.80	32.78 -3.15	68.83	57.33	64.25	63.47 -5.28
	DroidSpeak	40.85	35.65	26.95	34.48 -1.45	68.88	59.54	69.96	66.13 -2.63
	BaseShared	40.95	37.60	29.00	35.85 -0.08	70.75	63.25	70.25	68.08 -0.67
	BaseLRShared	41.10	37.70	27.15	35.32 -0.62	69.71	62.08	70.17	67.32 -1.43

Benchmark accuracy (%) on default Non-Shared (upper bound), naïve FullShared, DroidSpeak, and our proposed methods.

Memory Usage, TTFT, and System Throughput

- We report efficiency measured on the controlled trace of sequence lengths.
  - Latency in agentic systems with tool calling are highly dynamic and depends on both the method's efficiency & accuracy.
  - Traces constructed by varying the amount of context retrieved from external tools from 1k to 64k.



Implications

· KV cache memory usage reduced by nearly 33% since LR cache is negligible in size.

· BaseShared & DroidSpeak has limited efficiency due to the hidden state computation.

· BaseLRShared reduces this hidden state related computational overhead, and further reduces the LR cache computation by sharing the LR cache.

Model	Method	1.9k	3.0k	5.0k	9.1k	17.3k	33.7k	66.4k
LLaMA-3.1-8B-Instruct	Non-Shared	1.94	2.55	3.72	6.79	16.38	38.87	OOM
	FullShared	1.13	1.28	1.63	2.40	4.19	9.13	23.28
	DroidSpeak	1.62	2.17	3.22	5.55	11.15	25.43	67.80
	BaseShared	1.62	2.12	3.06	5.26	10.51	23.91	67.80
	BaseLRShared	1.13	1.28	1.64	2.43	4.24	9.19	23.35
Ministral-8B-Instruct	Non-Shared	2.02	2.65	3.85	6.84	17.67	41.67	OOM
	FullShared	1.19	1.35	1.69	2.50	4.37	9.30	20.84
	DroidSpeak	1.65	2.22	3.28	5.69	11.53	26.71	OOM
	BaseShared	1.66	2.19	3.17	5.43	10.85	25.57	59.62
	BaseLRShared	1.20	1.35	1.71	2.53	4.42	9.38	20.98

TTFT (sec) under each emulation traces (denoted by their total sequence length).
→ Up to 1.63x (BaseShared) and 4.44x (BaseLRShared) reduction

Model	Method	1.9k	3.0k	5.0k	9.1k	17.3k	33.7k	66.4k
LLaMA-3.1-8B-Instruct	Non-Shared	176.2	262.4	401.3	592.1	669.2	683.2	OOM
	FullShared	193.7	293.4	468.5	808.1	1246.1	1697.6	1826.5
	DroidSpeak	182.4	263.5	412.3	633.5	844.2	931.0	813.1
	BaseShared	169.0	257.0	392.3	617.3	862.8	969.6	823.2
	BaseLRShared	188.7	279.4	463.8	785.7	1239.4	1678.1	1790.6
Ministral-8B-Instruct	Non-Shared	158.9	231.0	361.5	541.2	610.7	638.4	OOM
	FullShared	169.9	251.4	420.3	711.2	1163.9	1538.6	1768.3
	DroidSpeak	160.9	251.4	360.3	570.2	785.1	856.0	OOM
	BaseShared	157.0	227.4	364.0	552.1	796.5	885.0	870.5
	BaseLRShared	164.1	247.9	410.9	703.2	1159.2	1521.9	1757.0

End-to-end system throughput (tokens/sec) under each emulation traces.
→ Up to 1.42x (BaseShared) and 2.46x (BaseLRShared) gain

Ablative Studies

- Accuracy and efficiency when LoRA applied to Q,K,V,O projections.
  - Due to the RoPE, unable to extend FLA on K projection.
  - Still exhibits high accuracy and throughput as in our main experiments.
- Extended accuracy results on non-shared-A multi-LoRA.
  - Default Non-Shared achieves best result on shared-A, therefore used in our main experiments.
  - Still, BaseShared exhibits dominant accuracy to other KV cache sharing schemes.

Model	Architecture	Non-Shared	FullShared	BaseShared	BaseLRShared
LLaMA-3.1-8B-Instruct	Non-shared A	42.05	40.30	41.85	36.25
	Shared-A	42.80 +0.75	41.15 +0.85	42.70 +0.85	42.40 +6.15
Ministral-8B-Instruct	Non-shared A	41.10	37.40	40.80	36.95
	Shared-A	41.30 +0.20	39.60 +2.20	40.95 +0.15	41.10 +4.15

HotpotQA easy accuracy (%) in non-shared-A and shared-A multi-LoRA variants.

Method	Non-Shared	FullShared	DroidSpeak	BaseShared	BaseLRShared
Accuracy (%)	38.43	34.88	36.28	38.12	37.57

HotpotQA average accuracy (%) with LoRA applied to QKVO projections.
(r=4 is used to match the #parameters with LoRA applied to QV projections)

Method	1.9k	3.0k	5.0k	9.1k	17.3k	33.7k	66.4k
Non-Shared	4.79	5.00	4.94	10.52	20.73	50.04	OOM
FullShared	1.23	1.40	1.81	2.70	4.82	9.57	24.05
DroidSpeak	1.77	2.29	3.37	5.84	11.54	25.43	67.80
BaseShared	1.78	2.34	3.29	5.59	10.99	26.20	70.57
BaseLRShared	1.27	1.46	1.92	2.89	4.93	10.30	25.30

TTFT (sec) under each traces with LoRA applied to QKVO projections.

Method	1.9k	3.0k	5.0k	9.1k	17.3k	33.7k	66.4k
Non-Shared	123.1	184.4	297.1	467.1	525.0	556.0	OOM
FullShared	155.8	215.1	357.2	626.3	1080.6	1544.9	1699.4
DroidSpeak	145.9	211.8	326.2	519.6	739.1	887.2	792.9
BaseShared	127.2	195.6	325.7	484.9	679.3	755.6	673.2
BaseLRShared	150.3	219.7	361.3	560.3	806.7	998.8	1051.9

System throughput (tokens/sec) with LoRA applied to QKVO projections.

Conclusion

Summary and Future Works

Summary and Future Works

- Summary
  - We introduce LRAgent, a KV cache sharing framework for multi-LoRA agent systems.
  - Decoupling the value cache into base and low-rank caches is crucial for accuracy preservation.
  - BaseShared improves memory efficiency, and BaseLRShared further improves computational efficiency through Flash-LoRA-Attention compared to state-of-the-art agent KV cache sharing methods.
- Future Works
  - Reducing hidden-state misalignment across agents
  - Extended methods to further improve either accuracy or efficiency under non-shared-A
  - Applications to the more various prompting-based agentic systems.

Thank you for listening.

Any questions or discussions are appreciated!