ICaRus: Identical Cache Reuse for Efficient Multi Model Inference

Sunghyeon Woo*, **Jaeeun Kil***, **Hoseung Kim, Minsub Kim, Joonghoon Kim, Ahreum Seo, Sungjae Lee, Minjung Jo, Jiwon Ryu, Baeseong Park, Se Jung Kwon, Dongsoo Lee**

NAVER Cloud

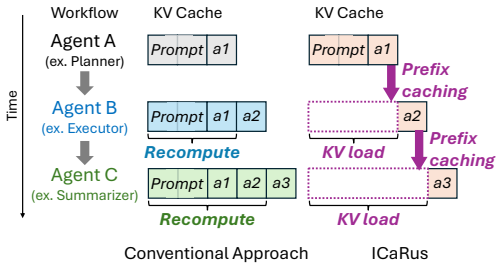
*Equal contribution

Abstract

Multi model inference, where multiple task-specialized models collaborate to solve complex real-world problems, has recently emerged as a prominent paradigm, particularly in the development of agentic AI systems. However, in such scenarios, each model must maintain its own Key-Value (KV) cache for the identical prompt, leading to substantial memory consumption. This explosive growth of KV caches forces LLM serving systems to evict previously stored caches, which in turn introduces significant recomputation overhead whenever the evicted caches are required again. Moreover, prefix caching is inherently infeasible across different models, forcing each model to recompute KV cache for the identical prompt, which leads to significant overhead. To alleviate these issues, we propose **I**dentical **Ca**che **Reus**e (**ICaRus**), a novel architecture that allows multiple models to share identical KV caches across all layers. ICaRus is based on the key observation that a decoder-only Transformer can be conceptually decomposed into a logical encoder, which generates KV caches, and a logical decoder, which predicts output tokens from the KV caches. ICaRus fine-tunes only the logical decoder while freezing the logical encoder, enabling multiple models to share an identical KV cache. This eliminates cache memory explosion and unexpected evictions while also allowing cross-model reuse of KV caches for new input tokens, thereby removing redundant recomputation in multi model inference achieving both efficiency and scalability. Moreover, by incorporating lightweight adapters such as LoRA, ICaRus parallelizes KV cache generation and next-token prediction during decoding. ICaRus achieves comparable accuracy to task-specific fine-tuned model across a diverse set of tasks, while allowing multiple specialized models to fully share KV caches. ICaRus achieves up to $11.1\times$ lower P95 latency and $3.8\times$ higher throughput in agentic workflow with 8 different models, compared to conventional multi model system.

1 Introduction

Large Language Models (LLMs) have shown strong performance across domains (Zhao et al., 2024; Dubey et al., 2024; Comanici et al., 2025; Yang et al., 2025); however, a single model struggles with complex tasks that demand multi step reasoning and domain-specific expertise (Tang et al., 2020; Yao et al., 2023; Sun et al., 2024). Recently, the emerging paradigm of multi model inference addresses this limitation by orchestrating task-specialized models, achieving higher accuracy and problem-solving ability than a general-purpose model (Fu et al., 2023; Du et al., 2024; Shen et al., 2024; Subramaniam et al., 2025). However, this paradigm introduces severe challenges in managing the Key-Value (KV) cache: each model maintains its own cache even for identical prefixes, causing memory consumption to grow rapidly with the number of models. Once GPU memory is saturated by KV cache, serving systems (Kwon et al., 2023; Zheng et al., 2024) must evict caches, which triggers redundant recomputation and significantly degrades throughput. Furthermore, because KV caches are model-specific, prefix caching (Kwon et al., 2023; Zheng et al., 2024) cannot be applied across different models, which forces identical prompts to rebuild KV caches independently and thereby increases latency.



(a) KV Cache management strategies in agent workflow using multi model

	Single Model	Multi Model	
		Conventional	ICaRus (Ours)
Training Method	Prompting	Fine-tuning	**Fine-tuning** (only logical Decoder)
Task Performance	Weak	Strong	**Strong**
KV Sharing	Inherent	Unsupported	**Supported**
KV Memory Usage	Low	High	**Low**
# Prefill Recomputation	Low	High	**Low**

(b) Comparison of ICaRus and conventional approaches

Figure 1: Comparison of KV cache management strategies and effectiveness in multi model scenarios between conventional approaches and ICaRus.

Previous KV cache optimization techniques, such as pruning (Zhang et al., 2023), quantization (Hooper et al., 2024; Yang et al., 2024), and inter-layer sharing (Qiao et al., 2024), reduce cache size while minimizing accuracy degradation. Unlike traditional LRU-based prefix caching, KVFlow (Pan et al., 2025) schedules KV cache eviction and prefetching based on anticipated agent workflow, reducing recomputation overhead. However, these methods focus only on single model cache management, leaving unresolved the challenges of cache explosion and the lack of KV cache sharing of prefix in multi model settings. DroidSpeak (Liu et al., 2024b) addressed multi model KV cache management by sharing non-sensitive layer caches between a base model and its fine-tuned variants, thereby reducing recomputation cost. However, this approach has inherent limitations, as caches from sensitive layers remain unshared and must still be recomputed.

To address these issues, we propose **I**dentical **Ca**che **Reus**e (**ICaRus**), a novel architecture that enables multiple models to share and reuse the same KV cache across all layers. The core idea of ICaRus originates from conceptually decomposing a decoder-only Transformer into two parts: a logical encoder, which is responsible for generating KV cache, and a logical decoder, which predicts the next token from the cache. We freeze the logical encoder of pretrained LLM (i.e. base model) and fine-tune only the logical decoder. Since all specialized models share the identical logical encoder, the KV cache generated for an identical prompt is likewise identical, enabling direct sharing without redundant memory usage as shown in Fig. 1(a). This prevents GPU memory from rapidly saturating due to KV cache growth, avoiding costly recomputation caused by cache eviction. Moreover, shared KV caches enable prefix caching across models, eliminating redundant computation for identical prompts and further improving efficiency as depicted in Fig. 1(b). In addition, ICaRus leverages the adapter architecture to generate the KV cache for the next step in parallel with the next-token computation during the decode phase. We evaluate ICaRus across diverse tasks including mathematics, coding, and knowledge understanding on a wide range of model families and scales (LLaMA-3.1-8B, Qwen3-1.7B/8B/14B). The results demonstrate that ICaRus achieves accuracy comparable to task-specific fine-tuned models, even though ICaRus-tuned models are able to share KV caches across tasks. Furthermore, when integrated into the vLLM serving system and evaluated in various multi agent scenarios including ReAct (Yao et al., 2023) and Reflexion (Shinn et al., 2023), ICaRus delivers as much as a $11.1\times$ reduction in 95th-percentile (P95) latency and a $3.8\times$ throughput gain compared to conventional multi model system.

In summary, the main contributions of this work are as follows:

- We propose ICaRus, the first architecture that enables multiple decoder-only Transformers to fully share KV caches, guaranteeing high generation quality in real serving scenarios by explicitly modeling the fully shared-KV setting already at training time.
- We demonstrate that ICaRus achieves accuracy comparable to task-specific fine-tuning across diverse tasks (mathematics, coding, and knowledge understanding) and model architectures (LLaMA-3.1-8B, Qwen3-1.7B/8B/14B).

- We confirm that ICaRus significantly improves efficiency in multi agent workflows, achieving up to $11.1\times$ reduction in P95 latency and $3.8\times$ improvement in throughput compared to conventional multi model system.

2 BACKGROUND & MOTIVATION

Key-Value Cache in LLM Serving Systems. During autoregressive inference, decoder-only Transformers generate tokens sequentially, where each new token depends on all previously generated tokens. Computing self-attention naïvely for every step requires recomputation over the entire sequence, incurring a per-token complexity of $\mathcal{O}(n^2)$ where n is the sequence length. To avoid this quadratic overhead, modern LLM serving systems cache the key and value representations of previously processed tokens (Vaswani et al., 2017). By reusing these cached states, each new decoding step only attends to the most recent token, reducing the per-token attention complexity to $\mathcal{O}(n)$ and thereby significantly lowering computational cost. However, the size of KV caches grows linearly with both sequence length and model depth, imposing substantial memory pressure on GPU-based serving systems (Kwon et al., 2023; Zheng et al., 2024). Consequently, memory-efficient cache management has emerged as a critical challenge for scalable LLM deployment.

Prefix Caching in LLM Serving Systems. Prefix caching is a widely adopted optimization that reuses the KV cache corresponding to a fixed prefix across multiple queries sharing the same initial context (Kwon et al., 2023; Zheng et al., 2024). This technique is particularly effective in scenarios such as retrieval-augmented generation (RAG) (Lewis et al., 2020) and instruction-tuned applications (Chung et al., 2024; Ouyang et al., 2022), where prompts often contain long but invariant components like system prompts, task-specific templates, or retrieved documents. By reusing the cached key-value states of these repeated prefixes, serving systems can avoid redundant computation during the prefill phase, effectively reducing the computational complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(mn)$, where n denotes the sequence length and m denotes the variable suffix length with $m \ll n$, thereby improving both throughput and latency. Moreover, prefix caching is highly beneficial in multi-turn conversational settings, where a large dialogue history is preserved across turns and only the most recent user utterance changes; by caching the KV states of the shared history and computing attention only for newly appended tokens, serving systems can efficiently support interactive dialogues without recomputing the entire context at every turn (Kim et al., 2025).

Agentic AI Workflow and Multi Model Inference. Agentic AI and workflow-based reasoning have given rise to complex pipelines in which models are orchestrated to perform specialized roles. For instance, ReAct (Yao et al., 2023) alternates between *Thought* $\rightarrow$ *Act* $\rightarrow$ *Observation*, Reflexion (Shinn et al., 2023) incorporates self-evaluation loops, LATS (Zhou et al., 2024) explores reasoning through parallel branch expansion, and LLMCompiler (Kim et al., 2024) constructs a DAG to schedule overlapping tool and model calls. When executed within a single model, such workflows can leverage prefix caching to avoid redundant computation, thereby reducing effective memory usage, lowering P95 latency, and improving throughput (Kim et al., 2025). However, in multi model settings where task-specialized models collaborate within a single pipeline, each model must maintain its own KV cache even for identical prefixes. Such KV cache duplication leads to memory usage that grows linearly with the number of active models; once GPU capacity is saturated, this growth inevitably triggers cache eviction, which in turn forces recomputation of evicted prefixes. Moreover, since prefix caching typically operates only within individual models, identical prefixes must be recomputed separately across models, leading to redundant prefill computation that inflates both latency and energy consumption. These limitations underscore the need for new architectures that support cross-model KV sharing and prefill de-duplication in multi model inference.

3 DESIGN OF ICARUS

3.1 DECODER-ONLY TRANSFORMER AS LOGICAL ENCODER AND DECODER

We first present a mathematical formulation of the decoder-only Transformer, which predicts the next token conditioned on the current token context. Specifically, we abstract x_i, k_i, and v_i as the i-th token, its key representation, and its value representation, respectively, and denote the decoder-

only Transformer by F. In this case, the next-token generation from the current token context in a decoder-only Transformer can be expressed as $x_{i+1} = F(x_1, x_2, \ldots, x_i)$. To generate the next token x_{i+1}, the model requires two types of information: the current token x_i and the accumulated key–value pairs. We denote the key set and value set up to step i as $K_{1:i} = \{k_1, k_2, \ldots, k_i\}, V_{1:i} = \{v_1, v_2, \ldots, v_i\}$. More concretely, in the attention operation, the query derived from x_i is generated anew at each step, whereas the keys and values are continuously appended to the cache and reused across subsequent decoding steps. In other words, the query does not persist beyond its step, but the KV pairs accumulate and form the long-term memory. This dependency can be expressed as

$$x_{i+1} = F(x_1, x_2, \ldots, x_i) = F\big(x_i,\, K_{1:i}, V_{1:i}\big). \tag{1}$$

Eq.1 indicates that a decoder-only Transformer predicts the next token conditioned on the current token x_i and the KV cache constructed up to this point. More generally, the generation process can be decomposed into two conceptual stages: (1) constructing the key set K_i and the value set V_i from the input sequence $x_{1:i} = \{x_1, x_2 \ldots, x_i\}$, and (2) decoding the next token x_{i+1} based on the current token x_i together with the accumulated sets (K_i, V_i). Formally, this can be expressed as

$$K_{1:i}, V_{1:i} = E(x_{1:i}), \tag{2}$$

$$x_{i+1} = D\big(x_i,\, K_{1:i}, V_{1:i}\big), \tag{3}$$

where E denotes the logical encoder that transforms the input sequence into its key and value representations, thereby constructing the KV cache, and D denotes the logical decoder that consumes the current token and the KV set to generate the next token. Importantly, a decoder-only Transformer can be interpreted as the special case where the parameters of the logical encoder and logical decoder are identical. More detailed concept of logical encoder-decoder architecture is depicted in Appendix C.

3.2 ICaRus: Identical Cache Reuse across LLMs

As described in Section 3.1, a decoder-only model can be decomposed into a logical encoder, which generates key–value pairs from a given token, and a logical decoder, which predicts the next token using the current token and the accumulated KV cache, as shown in Eqs. 2–3. From this perspective, task-specific fine-tuning can be viewed as jointly training both the logical encoder and the logical decoder to specialize in a given task. While such task-tuned models achieve strong task-specific capabilities, each maintains its own logical encoder thereby preventing KV cache sharing even when prompts are identical across models.

Building on this insight, we propose the ICaRus architecture which fine-tunes only the logical decoder of a decoder-only Transformer as below.

$$K_{1:i}, V_{1:i} = E_t(x_{1:i}) = E(x_{1:i}), \tag{4}$$

$$x^t_{i+1} = D_t\big(x_i,\, K_{1:i}, V_{1:i}\big), \tag{5}$$

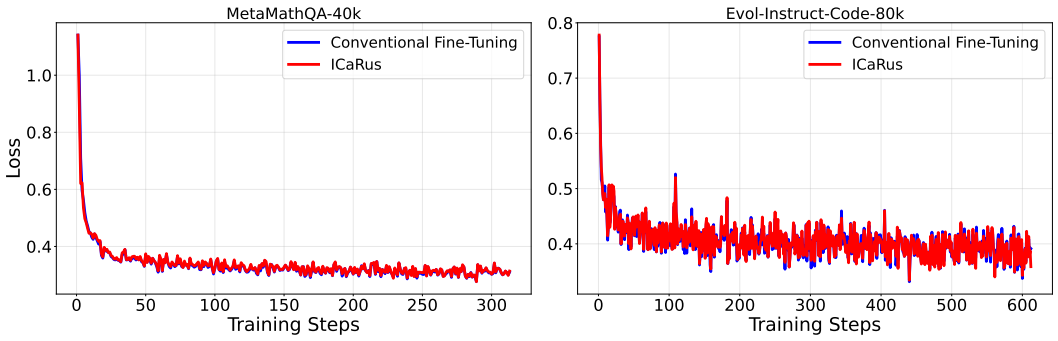


Figure 2: Training loss curves of conventional fine-tuning and ICaRus, both applied with LoRA on LLaMA-3.1-8B, trained on the MetaMathQA-40k and Evol-Instruct-Code-80k dataset.

Here, t and D_t denote a specific task and the logical decoder fine-tuned for that task, respectively. Specifically, the logical encoder (E) and the logical decoder (D) are initialized with the parameters of the base model, a pretrained decoder-only Transformer. The task-specific logical decoder D_t in Eq. 5 is then trained, starting from the base decoder D, to predict the next token x_{i+1} under two objectives: (1) specializing in the target task, and (2) leveraging the KV cache generated by the frozen logical encoder in Eq. 4. As a result, multiple task-specific logical decoders (e.g., D_{math}, D_{coding}, $D_{\text{reasoning}}$) can share a single logical encoder (i.e., $E_{\text{math}} \equiv E_{\text{coding}} \equiv E_{\text{reasoning}} \equiv E$), which is identical to the base model, thereby enabling all models to reuse the identical KV cache generated by the shared encoder, as illustrated in Fig. 1.

During training, the input data are duplicated and provided to both the logical encoder and the logical decoder. The logical encoder generates the corresponding key–value representations, while the logical decoder computes attention over these representations with its final output used to compute the training loss for gradient updates. The logical encoder is kept frozen during training to ensure cache sharing across tasks. This training procedure, which explicitly accounts for KV cache sharing, helps ensure robustness when KV caches are shared at inference time in real serving scenarios, especially compared with approaches that attempt to share KV caches across models trained independently without considering KV cache sharing.

Figure 2 shows the training loss of LLaMA-3.1-8B on MetaMathQA-40k (Yu et al., 2023) and Evol-Instruct-80k (Roshdieh, 2023). The ICaRus curves almost perfectly overlap with those of conventional task-specific fine-tuning, indicating that restricting learning to the logical decoder does not hinder optimization and is sufficient for task-specific adaptation even when the logical encoder is shared across models. In other words, freezing the logical encoder forces all task-specialized models to reuse a common sequence representation and express their differences only through the decoder, which can be interpreted as a form of implicit regularization.

The core idea of ICaRus is to factorize a decoder-only Transformer into a logical encoder and a logical decoder, and to train only the logical decoder so that KV caches can be shared across different models. Consequently, the specific adaptation method used to train the logical decoder is not essential to ICaRus itself: in principle, it could be trained via full-parameter fine-tuning, prompt tuning (Lester et al., 2021), LoRA (Hu et al., 2022) or variants (Liu et al., 2024a; Jiang et al., 2024; Woo et al., 2025) thereof. We adopt LoRA to train the logical decoder because LoRA offers high training efficiency, which enables rapid deployment of new agents in multi-agent systems, while achieving performance comparable to full-parameter fine-tuning (Schulman & Lab, 2025) and making it straightforward to optimize the decoding phase in ICaRus for inference efficiency. In the following section, we describe how we integrate LoRA into ICaRus and how this design further optimizes the overall inference cost.

3.3 Optimizing ICaRus for Multi Model Inference

In Section 3.2, we introduced the concept and training methodology of ICaRus. In this section, we explain how ICaRus operates in multi model inference scenarios and discuss its key optimization strategies. During the prefill phase, ICaRus uses only the logical encoder, which encodes the input prompt into a KV cache and produces the next token. In the subsequent decode phase, ICaRus duplicates the current token (x_i) and performs two operations: (1) encoding x_i into a key–value pair (k_i, v_i) through the logical encoder, and (2) predicting the task-specific output token (x_{i+1}) through the logical decoder by using the duplicated x_i together with the accumulated KV cache ($\{k_1, \ldots, k_i\}, \{v_1, \ldots, v_i\}$), as in Eq. 5. Consequently, regardless of which model performs decoding, the KV cache is always generated by the logical encoder, and other role-specific decoders can directly reuse this shared KV cache without any need to recompute or further update it. The details can be found in Appendix C

Sequential execution of the logical encoder and decoder may incur up to 2× latency overhead compared to a single model execution, since both weights and KV caches are accessed twice. To mitigate the problem, we insert and fine-tune only lightweight adapters within the logical decoder instead of fully fine-tuning the decoder. Consequently, the logical encoder and logical decoder share most parameters except for the adapters, enabling the shared parameters to be loaded only once and allowing the computations of the two modules to be executed in parallel as depicted in Fig. 3.

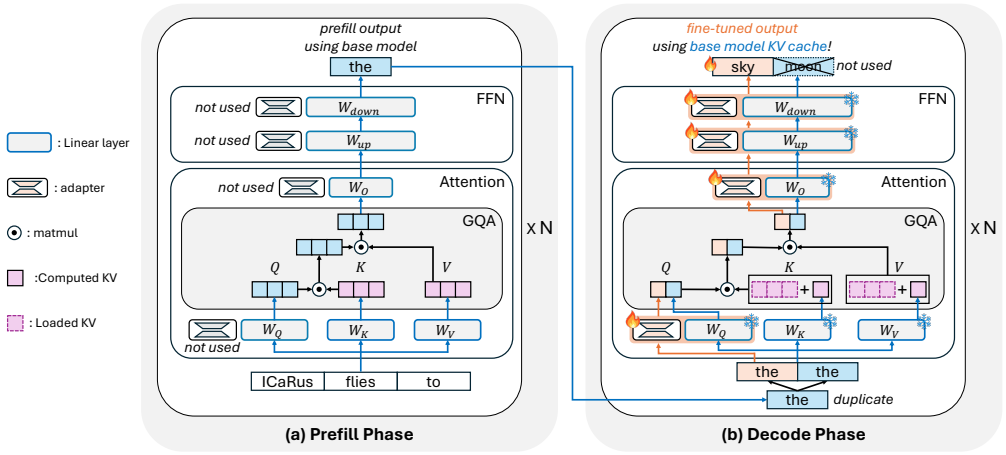


Figure 3: Overview of the ICaRus architecture. The base model, a pretrained decoder-only Transformer, serves as the logical encoder, while the adapter-tuned model (consisting of the base model and a tunable adapter) serves as the logical decoder. The blue and orange lines indicate computations performed by the base model and the adapter-tuned model, respectively. The purple square denotes that the same base model generates the KV cache during both the prefill and decoding phases. Consequently, different ICaRus models that share this base model as their logical encoder can reuse KV caches, even when the KV caches were created during the decoding phase by a different ICaRus model.

Table 1: Space and time complexity comparisons between single model and multi model scenarios.

Scenario	Method	Space Complexity	Time Complexity		
		Total	Prefill	Decode (per token)	
				Memory Access	Compute
Single Model	—	$\mathcal{O}(M + L_t)$	$\mathcal{O}(ML_t + L_t^2)$	$\mathcal{O}(M + L_t)$	$\mathcal{O}(M + L_t)$
Multi Model	BaseLine	$\mathcal{O}(M + NL_t)$	$\mathcal{O}(N(ML_t + L_t^2))$	$\boldsymbol{\mathcal{O}(M + L_t)}$	$\boldsymbol{\mathcal{O}(M + L_t)}$
	ICaRus	$\boldsymbol{\mathcal{O}(M + L_t)}$	$\boldsymbol{\mathcal{O}(ML_t + L_t^2)}$	$\boldsymbol{\mathcal{O}(M + L_t)}$	$\mathcal{O}(2M + 2L_t)$

In addition, because both models attend to the identical KV cache generated by the base model, we optimize attention computation by concatenating the query representations of the logical encoder and decoder along the head dimension (Fig. 3). This enables parallel attention computation without redundant KV cache reads. Consequently, although the decoding phase of ICaRus appears to double the computational workload by running both the logical encoder and decoder, the system adds only negligible latency overhead. This is because parallel execution generates memory traffic (base parameters, KV caches, and lightweight adapter weights) that is almost the same as that of a single model. The detailed algorithm can be found in Appendix B,

To validate the effectiveness of ICaRus, we further analyze the time and space complexity of multi model system built with the conventional approach (baseline) and with ICaRus, using N adapters in multi agent scenarios. Table 1 summarizes the results. We denote the input prompt length as L_i, the number of interaction turns per adapter as t, and the number of output tokens per turn as L_o, with the total sequence length $L_t = L_i + tL_o$. The base model size is represented by M. In the baseline, each model independently allocates KV memory and recomputes prefill for the same prompt, yielding space complexity $\mathcal{O}(M + NL_t)$ and prefill complexity $\mathcal{O}(N(ML_t + L_t^2))$. In contrast, ICaRus shares a single KV cache across models, reducing both to single model order, with space $\mathcal{O}(M + L_t)$ and prefill $\mathcal{O}(ML_t + L_t^2)$. The advantage grows with longer sequences from inter-model communication and with larger agent counts N.

During decoding, the baseline requires $\mathcal{O}(M + L_t)$ memory access and computation per token because each adapter-tuned model reads the model weights and its own KV cache. ICaRus computes

Table 2: Comparison of conventional methods and ICaRus on diverse datasets. Single Model denotes the base model without fine tuning. Multi Model consists of three independently fine tuned models: one on MetaMathQA-40K, one on Evol-Instruct-Code, and one on Oasst1. ICaRus uses the same three specializations, but trains only task-specific logical decoders on a shared logical encoder, enabling KV cache sharing across models.

Model	Method	KV Sharing	Math		Coding		Knowledge
			GSM8K	GSM+	HEval	HEval+	GPQA
LLaMA3.1-8B	Single Model	.	25.9	18.0	36.6	29.9	16.7
	Multi Model	X	**69.7**	**48.5**	48.2	41.5	27.3
	ICaRus (Ours)	O	67.9	45.8	**48.2**	**43.9**	**28.8**
Qwen3-8B-Base	Single Model	.	11.8	12.5	68.3	61.6	24.2
	Multi Model	X	85.4	66.1	81.7	75.6	**34.3**
	ICaRus (Ours)	O	**87.3**	**67.5**	**86.6**	**79.9**	33.8

both the logical encoder and decoder ($\mathcal{O}(2M + 2L_t)$) but parallelizes most of the computation so that the model and KV cache are read only once, restoring $\mathcal{O}(M + L_t)$. In multi-model, long-context, many-turn settings where decoding is memory-bound, memory access dominates; accordingly, ICaRus achieves decoding latency comparable to the baseline.

4 Evaluation

4.1 Experimental Setup

We evaluate ICaRus from two perspectives: (1) accuracy and (2) performance in multi model inference. In section 4.2, we construct multi model systems as follows. Starting from LLaMA-3.1-8B (Dubey et al., 2024) and Qwen3-1.7B/8B/14B-Base (Yang et al., 2025), we build three task-specific models per base model by fine-tuning on MetaMathQA-40k for mathematics (Yu et al., 2023), Evol-Instruct-Code-80k for coding (Roshdieh, 2023), and OASST1 for instruction tuning (Köpf et al., 2023) using either conventional fine-tuning or ICaRus. These systems are then evaluated on benchmarks aligned with each task: GSM8K (Cobbe et al., 2021) and GSM-Plus (Li et al., 2024) for mathematics, HumanEval (Chen et al., 2021) and HumanEval+ (Liu et al., 2023) for coding, and GPQA-Diamond (Rein et al., 2024) for knowledge understanding, using lm-eval-harness (Biderman et al., 2024) and EvalPlus (Liu et al., 2023) to measure zero-shot accuracy. For comparison, both the conventional fine-tuning and ICaRus use LoRA (Hu et al., 2022) as the adaptation method.

For multi model inference (Section 4.3), we measure latency and throughput under representative agent workflows such as ReAct (Yao et al., 2023) and Reflexion (Shinn et al., 2023) on the HotPotQA dataset (Yang et al., 2018), and for each workflow we evaluate configurations with 2, 4, and 8 agents. We adapt these workflows to a multi model, multi-turn request-routing setup: within a single workflow, successive requests from a multi-turn interaction are routed in a round-robin manner to different models. In this setting, the baseline is a conventional multi-LoRA system, whereas ICaRus replaces it with a cache-sharing multi agent system. To ensure a fair comparison, we integrate both systems into the vLLM serving framework and evaluate them under identical settings. More details can be found in the Appendix A.

4.2 Accuracy Evaluation

Accuracy on diverse task. We first train and evaluate ICaRus alongside **conventional** fine-tuning across mathematics, coding, and instruction-tuning tasks using LLaMA-3.1-8B and Qwen3-8B, as reported in Table 2. The results show that ICaRus achieves accuracy comparable to, or even surpassing, task-specific fine-tuning across all tasks. In particular, for the Qwen3-8B-Base model, ICaRus outperforms prior task-tuned models by at least 1.4% on benchmark evaluations for both mathematics and coding tasks. We expect that the superior accuracy of ICaRus stems from a generalization effect: by fine-tuning only the logical decoder while keeping the logical encoder frozen, ICaRus reduces the risk of overfitting compared to full task-specific fine-tuning.

Table 3: Comparison of conventional fine-tuning and ICaRus across different model sizes (Qwen3-1.7B/8B/14B-Base) trained on the MetaMathQA-40K dataset.

Model	**Qwen3-1.7B-Base**		**Qwen3-8B-Base**		**Qwen3-14B-Base**	
Method	Baseline	ICaRus	Baseline	ICaRus	Baseline	ICaRus
GSM8K	73.2	**74.0**	85.4	**87.3**	85.6	**88.8**
GSM+	53.7	**54.1**	66.1	**67.5**	66.7	**68.8**

Scaling with model size. We also examine the scalability of ICaRus with respect to model size by conducting experiments on Qwen3-1.7B/8B/14B-Base in Table 3. The results show that ICaRus consistently achieves higher accuracy compared to prior **conventionally fine-tuned baseline**, with improvements exceeding 2% on Qwen3-14B-Base, demonstrating that our method remains competitive as model capacity increases. Additionally, we verify the robustness of ICaRus across tasks and its scalability to larger model sizes by evaluating Qwen3-32B on tool-calling tasks, as described in Appendix D.

Table 4: Comparison of conventional methods and ICaRus in multi-model inference scenarios. Base Model denotes the LLaMA-3.1-8B-Base model without fine-tuning, while Math, Coding, and IF denote models fine-tuned on MetaMathQA-40K, Evol-Instruct-Code, and OASST1, respectively. Multi Model and ICaRus both consist of these three task-specific models; in ICaRus, however, only the logical decoders are fine-tuned while the logical encoder is shared across models.

# Model	**Method**	**KV Sharing**	**Math**		**Coding**		**Knowledge**	**Avg.**
			GSM8K	**GSM-Plus**	**HEval**	**HEval+**	**GPQA**	
1	Base Model	.	25.9	18.0	36.6	29.9	16.7	25.4
	Math Model	.	**69.7**	**48.5**	42.7	36.6	20.7	43.6
	Coding Model	.	22.8	17.5	**48.2**	41.5	21.7	30.3
	IF Model	.	24.5	16.5	44.5	39.0	27.2	30.3
3	Multi Model	X	**69.7**	**48.5**	**48.2**	41.5	27.2	**47.0**
	ICaRus (Ours)	O	67.9	45.8	**48.2**	**43.9**	**28.8**	46.9

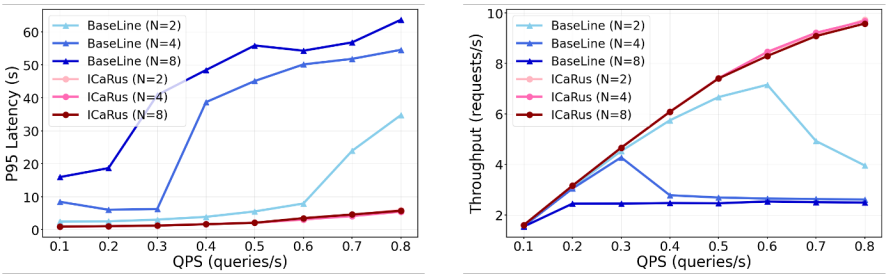
Multi domain orchestration results. Table 4 compares ICaRus orchestration with diverse single and multi model configurations using LLaMA-3.1-8B. Each task-tuned model is fine-tuned on a single domain-specific dataset (MetaMathQA for mathematics, Evol-Instruct-Code-80K for coding, and OASST1 for instruction-tuning). The results show that while a single task-specific fine-tuned model achieves high accuracy on its target task, the model suffers from significant performance degradation on other tasks. In contrast, a multi model system composed of multiple task-specific fine-tuned models achieves consistently high accuracy across all tasks. Our ICaRus also attains accuracy comparable to such multi model system, while additionally benefiting from KV cache sharing across agents, which enables orchestration at substantially lower computational cost.

4.3 Performance in Multi Model Inference

P95 latency and throughput across QPS. ICaRus consistently outperforms a baseline multi model system across all load levels in both latency and throughput, as evaluated on LLaMA-3.1-8B under the ReAct workflow (Fig. 4). We measure performance as the number of queries per second (QPS) increases; latency is reported at the 95th percentile (P95).

A key advantage of ICaRus is its ability to reuse identical prefix caches across models, avoiding the redundant recomputation required in baseline system where each model reconstructs its own cache. For example, at QPS 0.3 with 4 models, ICaRus reduces P95 latency by $5.1\times$ compared to the baseline, and this benefit becomes more pronounced as the number of models increases.

As the QPS increases, the cumulative KV cache size of baseline system soon exceeds GPU memory capacity, triggering eviction of previously stored KV caches and their subsequent recomputation. Consequently, throughput first plateaus and then declines, with the degradation occurring earlier as



(a) P95 latency across QPS

(b) Throughput across QPS

Figure 4: P95 latency and throughput of ICaRus compared with multiple task-specific agents fine-tuned from the LLaMA-3.1-8B base model under the ReAct workflow. Here, N denotes the number of LoRA modules, which are integrated into multi model system built using either the conventional approach or ICaRus.

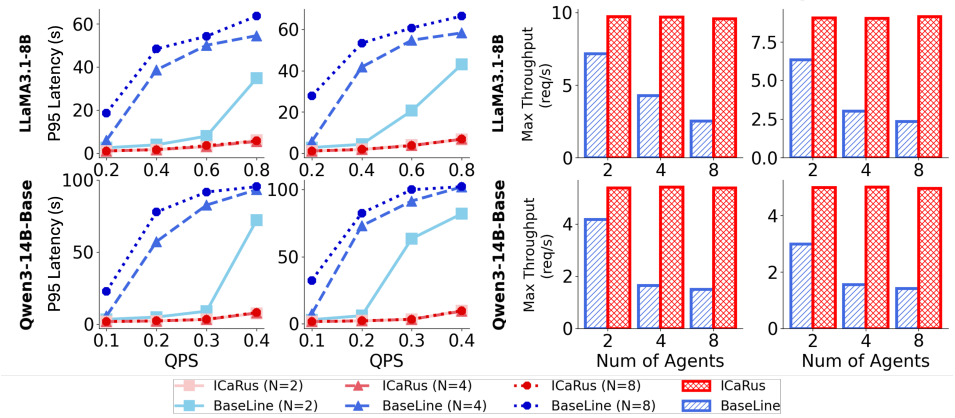


Figure 5: Comparison of P95 latency and maximum throughput across QPS for LLaMA3.1-8B and Qwen-3-14B Base under ReAct and Reflexion workflows.

the number of models increases (e.g., at 0.6 QPS for two models and 0.3 QPS for four models; Fig. 4(b). In contrast, ICaRus avoids redundant cache growth through cross-model KV sharing, allowing throughput to continue increasing even as baseline system plateau and decline.

Consequently, when comparing maximum achievable throughput, ICaRus outperforms the baseline by $1.4\times$, $2.3\times$, and $3.8\times$ with 2, 4, and 8 models, respectively. At the QPS where baseline system reach their peak throughput, ICaRus also achieves substantially lower P95 latency-$3.8\times$, $5.1\times$, and $11.1\times$ for 2, 4, and 8 models, respectively. Furthermore, we confirm that ICaRus continues to achieve lower latency and higher throughput than the baseline even in scenarios where evicted KV cache entries are managed by swapping rather than recomputation, as detailed in Appendix E.

Performance under diverse workflows or models. We further evaluate baseline system and ICaRus system across different models (LLaMA-3.1-8B and Qwen3-14B-Base) and multi agent workflows (ReAct and Reflexion). Specifically, we measure P95 latency over varying QPS and the maximum throughput achieved at the optimal QPS setting, as summarized in Fig. 5.

ICaRus prevents KV cache explosion and enables cross-model prefix caching, thereby achieving lower P95 latency and higher throughput in multi agent workflows. These gains persist even for larger models like Qwen3-14B, where ICaRus achieves up to $7.4\times$ lower latency and $3.6\times$ higher throughput compared to the baseline. Additionally, we verify that the advantages of ICaRus are preserved even under more realistic agentic patterns, where agents are invoked in a random order and the workload is skewed across agents, as demonstrated in Appendix F.

5 Related Work

Multi model Inference Leveraging multiple models has been widely explored as a way to improve performance over a single model. Routing methods either select the most appropriate model or use multiple models in a cascade (Chen et al., 2024; Shnitzer et al., 2024), while ensemble approaches combine the outputs of multiple models, either at the token level (Yu et al., 2024; Huang et al., 2024) or at the reasoning step level (Park et al., 2025). Multi model approaches have also been applied in multi agent systems, where interactions among agents have been shown to enhance performance across diverse tasks (Fu et al., 2023; Sun et al., 2024; Du et al., 2024). In these systems, each agent used either a base model or fine-tuned variants obtained with methods such as LoRA or instruction tuning (Mineiro, 2024; Liu et al., 2025b).

KV Cache Optimization KV cache stores the keys and values of previous tokens to avoid redundant recomputation during autoregressive generation and is traditionally used on a per-request basis (Vaswani et al., 2017). Prefix caching techniques extend the lifetime of the KV cache beyond a single request, enabling multiple turns or related requests to share the same cache (Gao et al., 2024; Gim et al., 2024). However, prefix caching alone cannot address the challenge of deploying multiple models, as KV caches cannot be shared across different models even for identical prompts, and each model generates a distinct KV cache. DroidSpeak (Liu et al., 2025b) addresses this issue by reusing the KV cache of a shared foundational model for non-sensitive layers, while selectively recomputing only the sensitive layers in each agent model. This approach requires identifying sensitive layers that must be recomputed by the agent model, thereby affecting subsequent layers. On a different axis, KVFlow (Pan et al., 2025) manages KV caches by evicting and prefetching based on predetermined agentic workflows instead of an LRU policy, but it remains a single model approach with agents defined by prompts.

6 Conclusion

In this work, we presented ICaRus, a KV cache-sharing architecture for multi model inference. ICaRus addresses the memory inefficiency of conventional system by enabling cross-model KV cache reuse, while maintaining accuracy through fine-tuning. Experiments across mathematics, coding, and instruction-following tasks confirm that ICaRus delivers accuracy on par with task-specific fine-tuned models, yet achieves significantly lower latency and higher throughput in multi agent workflows. Taken together, these results establish ICaRus as a principled approach for scalable and efficient multi model inference. Looking ahead, we expect ICaRus to extend to large-scale models, heterogeneous agent systems, and real-world deployment scenarios where scalability and efficiency are increasingly critical.

Reproducibility Statement

We formulated the concept of the logical encoder and decoder in detail, which forms the foundation of the ICaRus algorithm, in Section 3.1. Furthermore, we provided a rigorous mathematical formulation of ICaRus, along with its training procedure and convergence of the loss curve, in Section 3.2. The inference process of ICaRus and the corresponding optimization strategies are described in Section 3.3, with pseudocode provided in Appendix B. Finally, the detailed experimental setup for both training and inference is presented in Section 4.1 and Appendix A.

References

Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sidney Black, Jordan Clive, Anthony DiPofi, Julen Etxaniz, Benjamin Fattori, Jessica Zosa Forde, Charles Foster, Jeffrey Hsu, Mimansa Jaiswal, Wilson Y. Lee, Haonan Li, Charles Lovering, Niklas Muennighoff, Ellie Pavlick, Jason Phang, Aviya Skowron, Samson Tan, Xiangru Tang, Kevin A. Wang, Genta Indra Winata, François Yvon, and Andy Zou. Lessons from the trenches on reproducible evaluation of language models. *arXiv preprint arXiv:2405.14782*, 2024. URL https://arxiv.org/abs/2405.14782.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=cSimKw5p6R.

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

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Y. Zhao, Yanping Huang, Andrew M. Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models. *J. Mach. Learn. Res.*, 25:70:1–70:53, 2024. URL https://jmlr.org/papers/v25/23-0870.html.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit S. Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, Luke Marris, Sam Petulla, Colin Gaffney, Asaf Aharoni, Nathan Lintz, Tiago Cardal Pais, Henrik Jacobsson, Idan Szpektor, Nan-Jiang Jiang, Krishna Haridasan, Ahmed Omran, Nikunj Saunshi, Dara Bahri, Gaurav Mishra, Eric Chu, Toby Boyd, Brad Hekman, Aaron Parisi, Chaoyi Zhang, Kornraphop Kawintiranon, Tania Bedrax-Weiss, Oliver Wang, Ya Xu, Ollie Purkiss, Uri Mendlovic, Ilaï Deutel, Nam Nguyen, Adam Langley, Flip Korn, Lucia Rossazza, Alexandre Ramé, Sagar Waghmare, Helen Miller, Nathan Byrd, Ashrith Sheshan, Raia Hadsell Sangnie Bhardwaj, Pawel Janus, Tero Rissa, Dan Horgan, Sharon Silver, Ayzaan Wahid, Sergey Brin, Yves Raimond, Klemen Kloboves, Cindy Wang, Nitesh Bharadwaj Gundavarapu, Ilia Shumailov, Bo Wang, Mantas Pajarskas, Joe Heyward, Martin Nikoltchev, Maciej Kula, Hao Zhou, Zachary Garrett, Sushant Kafle, Sercan Arik, Ankita Goel, Mingyao Yang, Jiho Park, Koji Kojima, Parsa Mahmoudieh, Koray Kavukcuoglu, Grace Chen, Doug Fritz, Anton Bulyenov, Sudeshna Roy, Dimitris Paparas, Hadar Shemtov, Bo-Juen Chen, Robin Strudel, David Reitter, Aurko Roy, Andrey Vlasov, Changwan Ryu, Chas Leichner, Haichuan Yang, Zelda Mariet, Denis Vnukov, Tim Sohn, Amy Stuart, Wei Liang, Minmin Chen, Praynaa Rawlani, Christy Koh, JD Co-Reyes, Guangda Lai, Praseem Banzal, Dimitrios Vytiniotis, Jieru Mei, and Mu Cai. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *CoRR*, abs/2507.06261, 2025. doi: 10.48550/ARXIV.2507.06261. URL https://doi.org/10.48550/arXiv.2507.06261.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=zj7YuTE4t8.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak,

Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024. doi: 10.48550/ARXIV.2407.21783. URL https://doi.org/10.48550/arXiv.2407.21783.

Yao Fu, Hao Peng, Tushar Khot, and Mirella Lapata. Improving language model negotiation with self-play and in-context learning from ai feedback, 2023. URL https://arxiv.org/abs/2305.10142.

Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. {Cost-Efficient} large language model serving for multi-turn conversations with {CachedAttention}. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pp. 111–126, 2024.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In P. Gibbons, G. Pekhimenko, and C. De Sa (eds.), *Proceedings of Machine Learning and Systems*, volume 6, pp. 325–338, 2024.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length LLM inference with KV cache quantization. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/028fcbcf85435d39a40c4d61b42c99a4-Abstract-Conference.html.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Yichong Huang, Xiaocheng Feng, Baohang Li, Yang Xiang, Hui Wang, Ting Liu, and Bing Qin. Ensemble learning for heterogeneous large language models with deep parallel collaboration. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=7arAADUK6D.

Ting Jiang, Shaohan Huang, Shengyue Luo, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, Qi Zhang, Deqing Wang, and Fuzhen Zhuang. Mora: High-rank updating for parameter-efficient fine-tuning. *CoRR*, abs/2405.12130, 2024. doi: 10.48550/ARXIV.2405.12130. URL https://doi.org/10.48550/arXiv.2405.12130.

Jiin Kim, Byeongjun Shin, Jinha Chung, and Minsoo Rhu. The cost of dynamic reasoning: Demystifying AI agents and test-time scaling from an AI infrastructure perspective. *CoRR*, abs/2506.04301, 2025. doi: 10.48550/ARXIV.2506.04301. URL https://doi.org/10.48550/arXiv.2506.04301.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=uQ2FUoFjnF.

Andreas Köpf, Yannic Kilcher, Dimitri Von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, et al. Openassistant conversations-democratizing large language model alignment. *Advances in neural information processing systems*, 36:47669–47681, 2023.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In Jason Flinn, Margo I. Seltzer, Peter Druschel, Antoine Kaufmann, and Jonathan Mace (eds.), *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pp. 611–626. ACM, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pp. 3045–3059. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.EMNLP-MAIN.243. URL https://doi.org/10.18653/v1/2021.emnlp-main.243.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

Qintong Li, Leyang Cui, Xueliang Zhao, Lingpeng Kong, and Wei Bi. Gsm-plus: A comprehensive benchmark for evaluating the robustness of llms as mathematical problem solvers. *arXiv preprint arXiv:2402.19255*, 2024.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36:21558–21572, 2023.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024a. URL https://openreview.net/forum?id=3d5CIRG1n2.

Weiwen Liu, Xu Huang, Xingshan Zeng, xinlong hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Wang Xinzhi, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. ToolACE: Winning the points of LLM function calling. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=8EB8k6DdCU.

Yuhan Liu, Esha Choukse, Shan Lu, Junchen Jiang, and Madan Musuvathi. Droidspeak: Enhancing cross-llm communication. *CoRR*, abs/2411.02820, 2024b. doi: 10.48550/ARXIV.2411.02820. URL https://doi.org/10.48550/arXiv.2411.02820.

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 for cross-llm communication and multi-llm serving, 2025b. URL https://arxiv.org/abs/2411.02820.

Paul Mineiro. Online joint fine-tuning of multi-agent flows, 2024. URL https://arxiv.org/abs/2406.04516.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html.

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. *CoRR*, abs/2507.07400, 2025. doi: 10.48550/ARXIV.2507.07400. URL https://doi.org/10.48550/arXiv.2507.07400.

Sungjin Park, Xiao Liu, Yeyun Gong, and Edward Choi. Ensembling large language models with process reward-guided tree search for better complex reasoning. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 10256–10277, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.515. URL https://aclanthology.org/2025.naacl-long.515/.

Aurick Qiao, Zhewei Yao, Samyam Rajbhandari, and Yuxiong He. Swiftkv: Fast prefill-optimized inference with knowledge-preserving model transformation. *CoRR*, abs/2410.03960, 2024. doi: 10.48550/ARXIV.2410.03960. URL https://doi.org/10.48550/arXiv.2410.03960.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

Nick Roshdieh. Evol-instruct-code-80k. https://huggingface.co/datasets/nickrosh/Evol-Instruct-Code-80k-v1, 2023. Hugging Face dataset.

John Schulman and Thinking Machines Lab. Lora without regret. https://thinkingmachines.ai/blog/lora/, 2025. Blog post.

Weizhou Shen, Chenliang Li, Hongzhan Chen, Ming Yan, Xiaojun Quan, Hehong Chen, Ji Zhang, and Fei Huang. Small llms are weak tool learners: A multi-llm agent. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pp. 16658–16680. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.EMNLP-MAIN.929. URL https://doi.org/10.18653/v1/2024.emnlp-main.929.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Tal Shnitzer, Anthony Ou, Mírian Silva, Kate Soule, Yuekai Sun, Justin Solomon, Neil Thompson, and Mikhail Yurochkin. Large language model routing with benchmark datasets. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Zb0ajZ7vAt.

Vighnesh Subramaniam, Yilun Du, Joshua B. Tenenbaum, Antonio Torralba, Shuang Li, and Igor Mordatch. Multiagent finetuning: Self improvement with diverse reasoning chains. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=JtGPIZpOrz.

Qiushi Sun, Zhangyue Yin, Xiang Li, Zhiyong Wu, Xipeng Qiu, and Lingpeng Kong. Corex: Pushing the boundaries of complex reasoning through multi-model collaboration. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=7BCmIWVT0V.

Hongyan Tang, Junning Liu, Ming Zhao, and Xudong Gong. Progressive layered extraction (PLE): A novel multi-task learning (MTL) model for personalized recommendations. In Rodrygo L. T. Santos, Leandro Balby Marinho, Elizabeth M. Daly, Li Chen, Kim Falk, Noam Koenigstein, and Edleno Silva de Moura (eds.), *RecSys 2020: Fourteenth ACM Conference on Recommender Systems, Virtual Event, Brazil, September 22-26, 2020*, pp. 269–278. ACM, 2020. doi: 10.1145/3383313.3412236. URL https://doi.org/10.1145/3383313.3412236.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA,*

USA, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Sunghyeon Woo, Sol Namkung, Sunwoo Lee, Inho Jeong, Beomseok Kim, and Dongsuk Jeon. Paca: Partial connection adaptation for efficient fine-tuning. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=iYkhxre0In.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

June Yong Yang, Byeongwook Kim, Jeongin Bae, Beomseok Kwon, Gunho Park, Eunho Yang, Se Jung Kwon, and Dongsoo Lee. No token left behind: Reliable KV cache compression via importance-aware mixed precision quantization. *CoRR*, abs/2402.18096, 2024. doi: 10.48550/ARXIV.2402.18096. URL https://doi.org/10.48550/arXiv.2402.18096.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

Yao-Ching Yu, Chun Chih Kuo, Ye Ziqi, Chang Yucheng, and Yueh-Se Li. Breaking the ceiling of the LLM community by treating token generation as a classification for ensembling. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 1826–1839, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.99. URL https://aclanthology.org/2024.findings-emnlp.99/.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. H2O: heavy-hitter oracle for efficient generative inference of large language models. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/6ceefa7b15572587b78ecfcebb2827f8-Abstract-Conference.html.

Justin Zhao, Timothy Wang, Wael Abid, Geoffrey Angus, Arnav Garg, Jeffery Kinnison, Alex Sherstinsky, Piero Molino, Travis Addair, and Devvret Rishi. Lora land: 310 fine-tuned llms that rival gpt-4, A technical report. *CoRR*, abs/2405.00732, 2024. doi: 10.48550/ARXIV.2405.00732. URL https://doi.org/10.48550/arXiv.2405.00732.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In Amir Globersons, Lester

Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=njwv9BsGHF.

APPENDICES

A EXPERIMENTAL SETUP

A.1 TRAINING SETUP

All experiments were conducted on a single node with 8xNVIDIA A100 GPUs (80GB each). Each GPU processed a micro-batch of size 1, and we applied gradient accumulation over 16 steps, resulting in an effective batch size of 128 examples across all devices. This corresponds to approximately 131k tokens per optimization step when the maximum sequence length was 1024, and 262k tokens when it was 2048.

We trained on three datasets: MetaMathQA (40k sampled examples), Evol-Instruct (80k full set), and OASST1 (10k sampled examples). The maximum sequence length was set to 2048 for Evol-Instruct and 1024 for the others. The number of training epochs was 1 for MetaMathQA and Evol-Instruct, and 3 for OASST1.

Optimization was performed using the AdamW optimizer with default hyperparameters (β_1=0.9, β_2=0.999) and a weight decay of 0.01. We used a cosine learning rate decay schedule with a warmup ratio of 0.03, and performed a grid search over learning rates $\{1 \times 10^{-4}, 2 \times 10^{-4}, 5 \times 10^{-4}\}$. No additional regularization techniques (e.g., dropout or gradient clipping) were applied.

For all experiments, we applied low-rank adaptation (LoRA) with a rank of 128 and an α of 256.

A.2 MULTI MODEL INFERENCE SETUP

A.2.1 AGENT WORKFLOW SELECTION AND DESIGN

We designed our experimental setup to evaluate the scalability and performance characteristics of multi model AI agent systems under realistic workload conditions. For this study, we selected two representative agent workflows that exemplify different reasoning patterns commonly deployed in production environments:

ReAct (Yao et al., 2023): This framework synergizes chain-of-thought reasoning with external tool use through an iterative process where agents generate reasoning traces and task-specific actions in an interleaved manner. In the ReAct paradigm, agents alternate between internal reasoning (thoughts) and external actions (tool calls), with each iteration consisting of a thought-action-observation cycle. This pattern is particularly effective for tasks requiring dynamic interaction with external knowledge bases and APIs.

Reflexion (Shinn et al., 2023): This framework reinforces language agents through linguistic feedback, maintaining reflective text in an episodic memory buffer to improve decision-making across multiple trials. Unlike ReAct, Reflexion adds self-evaluation capabilities where agents generate verbal reinforcement cues to assist in self-improvement, storing these experiences in long-term memory for rapid adaptation. This approach enables agents to learn from past mistakes without requiring model fine-tuning, achieving superior performance on complex reasoning tasks.

A.2.2 MULTI MODEL ARCHITECTURE WITH LORA ADAPTERS

To simulate realistic multi-tenant agent deployments, we implemented a multi model inference setup where each agent instance operates with its own Low-Rank Adaptation (LoRA) adapter. This configuration mirrors production scenarios where different agents may require specialized model behaviors or domain-specific fine-tuning. Specifically, we matched the number of concurrent agents to the number of LoRA adapters, ensuring that each agent maintains its own parameter space.

In evaluation, multiple task-specific LoRA adapters share the same base model on a single GPU. Under this setup, both the baseline multi-LoRA system and ICaRus already leverage the standard prefix/KV-aware mechanisms of the serving stack: requests routed to the same LoRA module reuse the existing KV cache for identical prefixes whenever possible, thereby sharing KV-cache memory and avoiding redundant prefill recomputation within each model.

This architectural choice has significant implications for system resources:

1. **Memory Overhead**: Each agent maintains its own KV cache throughout multi-turn interactions. With N concurrent agents, the memory requirement scales by a factor of N, as each agent's context must be preserved independently across conversation turns.
2. **Computational Load**: Multi-turn agent requests generate new computational burdens at each interaction step. As agents progress through reasoning chains (ReAct) or reflection cycles (Reflexion), each turn requires fresh attention computations over the accumulated context, leading to quadratic scaling in computational complexity.

A.2.3 Workload Characterization

For workload modeling, we used the HotPotQA dataset (Yang et al., 2018) as the underlying question-answering benchmark for both ReAct and Reflexion workflows, following the setup of Kim et al. (2025). Input/output distributions and tool-calling patterns were based on empirical measurements from Kim et al. (2025), which provides comprehensive statistics on real-world agent workflow characteristics. These patterns informed our synthetic workload generation, ensuring our experiments reflect actual deployment scenarios.

A.2.4 Experimental Parameters

We conducted systematic scaling experiments with the following configuration:

Agent Scaling: We evaluated system behavior with 2, 4, and 8 concurrent agents to understand how resource contention and memory pressure evolve with increasing agent density.

Request Rate (QPS):

- For Qwen2.5 14B: Tested at 0.1, 0.2, 0.3, and 0.4 QPS
- For Llama 3.1 8B: Tested at 0.2, 0.4, 0.6, and 0.8 QPS

The different QPS ranges reflect the computational differences between model sizes, with the smaller 8B model capable of sustaining higher request rates.

Throughput Measurement: We measured actual system throughput at the 0.8 QPS configuration to empirically determine system saturation points under peak load conditions.

Batch Size and Latency Dynamics: To understand latency behavior under constrained conditions, we fixed the total request count at 128 while varying QPS. This experimental design differs from unbounded request streams where continuously arriving requests would cause monotonically increasing batch sizes and consequently unbounded growth in 95th percentile latency. Under our fixed-request protocol, we observed that 95th percentile latency initially increases with QPS but eventually saturates at a plateau, indicating the system reaches a steady-state where all requests are being processed within the available compute budget.

This saturation behavior provides critical insights into:

- The maximum sustainable batch size for each agent configuration
- The point at which additional request rate increases no longer impact tail latency
- The effective capacity limits of multi agent systems under resource constraints

A.2.5 Rationale and Implications

Our experimental design captures several critical aspects of production multi agent systems:

1. **Resource Isolation**: By assigning separate LoRA adapters to each agent, we model scenarios where agents require distinct specializations (e.g., different domains, languages, or task-specific fine-tuning).
2. **Memory Pressure**: The multiplicative effect of agent count on KV cache requirements reflects real-world memory bottlenecks in multi-tenant deployments.
3. **Workflow Diversity**: The combination of ReAct's tool-calling patterns and Reflexion's self-improvement cycles represents a broad spectrum of agent behavioral patterns, from reactive tool use to iterative refinement.

4. **Scaling Characteristics**: Our range of agent counts (2–8) and QPS values provides insights into both vertical scaling (request rate) and horizontal scaling (agent parallelism) dimensions.

This setup enables us to quantify the trade-offs between agent autonomy, system throughput, and resource utilization in modern AI agent deployments, providing actionable insights for practitioners deploying multi agent systems at scale.

B Pseudo Algorithm

B.1 Prefill Phase in ICaRus

Algorithm 1: Prefill Phase (Standard Linear Only)

Input: Prompt tokens $P \in \mathcal{V}^N$
Output: First token $y_{\text{prefill}} \in \mathcal{V}$, $\texttt{KV\_CACHE}[1 \dots L]$

$X_1 \leftarrow \text{Embed}(P) \in \mathbb{R}^{N \times d}$
for $i = 1$ **to** L **do**
  $Q_i \leftarrow \text{Linear}(X_i; W_q^i), K_i \leftarrow \text{Linear}(X_i; W_k^i), V_i \leftarrow \text{Linear}(X_i; W_v^i)$
  $Q_i, K_i \in \mathbb{R}^{N \times d_k}, V_i \in \mathbb{R}^{N \times d_v}$
  `/* generate KV cache (w. the Logical Encoder) */`
  $\texttt{KV\_CACHE}[i] \leftarrow (\boldsymbol{K_i}, \boldsymbol{V_i})$
  $A_i \leftarrow \text{Attention}(Q_i, K_i, V_i) \in \mathbb{R}^{N \times d_v}$
  $X_{i+1} \leftarrow \text{FFN}(\text{AttentionOutput}(A_i)) \in \mathbb{R}^{N \times d}$
$y_{\text{prefill}} \leftarrow \text{Sample}(\text{LMHead}((X_{L+1}[N]))$ `// Prefill Result`

B.2 DECODE PHASE IN ICARUS

Algorithm 2: ICaRus Linear

Input: $X \in \mathbb{R}^{2\times T\times d}$ // batch=2, seqlen T, hidden size d

X[0]: Input for Logical Encoder (Base model)
X[1]: Input for Logical Decoder (Base model + Adaptive model)

Output: $Y \in \mathbb{R}^{2\times T\times d}$

/* Parallel execution for Base Model and Adaptive Model */
$\boldsymbol{X}_{\text{temp}} \leftarrow \mathbf{Linear}(\boldsymbol{X})$
$X_{\text{temp}}[1] \leftarrow X_{\text{temp}}[1] + \text{AdaptiveLinear}(X_{\text{temp}}[1])$
$Y \leftarrow X_{\text{temp}}$

Algorithm 3: Decode Phase (w. ICaRus Linear)

Input: $y_{\text{prefill}} \in \mathcal{V}$, **KV_CACHE**$[1 \dots L]$

KV_CACHE: Prompt KV cache from Logical Encoder (Base Model)

Output: Generated tokens $Y = (y_{N+1}, y_{N+2}, \dots, y_{N+T})$
(where N is the prompt length, T is the number of generated tokens)

$Input\_Token \leftarrow y_{\text{prefill}}$
for $t = 1 \dots T$ **do**
  $X_1 \leftarrow \text{Embed}(Input\_Token) \in \mathbb{R}^{N\times d}$
  /* Stack hidden states for ICaRus Execution */
  $\boldsymbol{X}_1^{\text{pair}} \leftarrow \mathbf{stack\_batch}(\boldsymbol{X_1}, \boldsymbol{X_1})$ // shape: [2,1,d]
  for $i = 1$ **to** L **do**
    /* KV cache from base model for sharing */
    $\boldsymbol{K}_i^{\text{step}} \leftarrow \mathbf{Linear}(\boldsymbol{X}_i; \boldsymbol{W}_k^i), \boldsymbol{V}_i^{\text{step}} \leftarrow \mathbf{Linear}(\boldsymbol{X}_i; \boldsymbol{W}_v^i)$
    $(K_i^{\text{cache}}, V_i^{\text{cache}}) \leftarrow$ KV_CACHE$[i]$
    $K_i \leftarrow \text{concat\_sequence}(K_i^{\text{cache}}, K_i^{\text{step}})$
    $V_i \leftarrow \text{concat\_sequence}(V_i^{\text{cache}}, V_i^{\text{step}})$
    KV_CACHE$[i] \leftarrow (K_i, V_i)$
    $Q_i^{\text{pair}} \leftarrow \text{ICaRusLinear}(X_i^{\text{pair}}; W_q^i, A_q^i)$ // shape: [2,1,H,d_k]
    /* Enable attention parallelism via GQA */
    $\boldsymbol{Q}_i \leftarrow \mathbf{concat\_numhead}(\boldsymbol{Q}_i^{\text{pair}}[\mathbf{0}], \boldsymbol{Q}_i^{\text{pair}}[\mathbf{1}])$ // shape: [1,2*H,d_k]
    $\boldsymbol{A}_i \leftarrow \mathbf{GQA}(\boldsymbol{Q}_i, \boldsymbol{k}_i, \boldsymbol{V}_i)$ // shape: [1,2*H,d_v]
    $\boldsymbol{A}_i^{\text{pair}} \leftarrow \mathbf{transpose\_and\_reshape}(\boldsymbol{A}_i)$ // shape: [2,1,H,d_v]
    $Z_i^{\text{pair}} \leftarrow \text{ICaRusLinear}(A_i^{\text{pair}}; W_o^i, A_o^i)$ // shape: [2,1,d]
    /* FFN: up → act → down (W.ICaRusLinear) */
    $F_i^{\text{pair}} \leftarrow \text{FFN}(Z_i^{\text{pair}})$ // shape: [2,1,d]
  /* use only Adaptive Result */
  $new\_token \leftarrow \text{Sample}(\text{LMHead}(F_{L+1}^{\text{pair}}[1]))$
  $Y \leftarrow \text{concat}(Y, new\_token)$
  $Input\_Token \leftarrow new\_token$

C Logical Encoder–Decoder: Concept and Inference Workflow

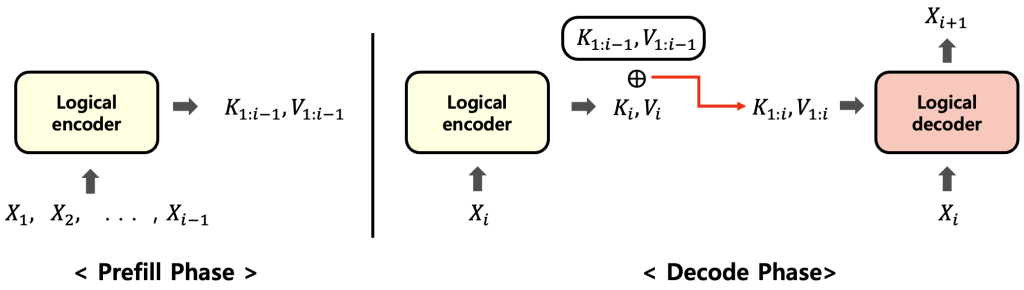


Figure 6: Inference workflow of the logical encoder-decoder.

In this section, we provide a more detailed explanation of the logical encoder–decoder concept. Inference in a decoder-only Transformer can be viewed as consisting of two phases: a *prefill* phase and a *decode* phase.

- **Prefill:** generate the KV cache for the input prompt.
- **Decode:** (1) generate the KV cache for the current token, and (2) predict the next token.

Motivated by this behavior, we conceptually decompose the model into a *logical encoder* and a *logical decoder*. The logical encoder denotes the part of the computation that is solely responsible for producing the KV cache, whereas the logical decoder denotes the part that predicts the next token during decoding and does not produce any new KV entries: it treats the KV cache as a pre-computed sequence representation and only issues queries against it to generate tokens. Under this decomposition, inference can be reinterpreted as follows:

- **Prefill:** the logical encoder generates the KV cache for the input prompt.
- **Decode:** (1) the logical encoder generates the KV cache for the current token, and (2) the logical decoder predicts the next token.

ICaRus fine-tunes only the logical decoder and freeze logical encoder. Specifically, the task-specialized decoders consume the shared KV cache from the common logical encoder for attention computation, as shown in Fig. 6, enabling heterogeneous, task-specialized decoders to operate on a single shared representation without any approximation or recomputation. In other words, ICaRus models can reuse KV cache entries produced not only in the prefill phase but also in the decode phase without any updates or reconstruction, because all KV entries are always generated by the same logical encoder.

D Robustness of ICaRus on Tool-Calling Tasks with Larger Models

To demonstrate the scalability and robustness of ICaRus, We conducted experiments with Qwen3-32B on the ToolAce dataset (Liu et al., 2025a) for tool calling related task, and evaluated the resulting models on the BFCL benchmark as shown below.

As shown in Fig. 2, the loss curve of ICaRus converges smoothly and is comparable to that of the baseline, which is consistent with the behavior observed in Figure 2 of the manuscript for math and coding tasks with 8B models. This indicates that our training procedure remains stable even when scaling to larger models and to a different task domain.

Moreover, as reported in Table 5, even with a larger 32B model and the tool calling task, ICaRus achieves comparable accuracy than a baseline that does not share the KV cache. This suggests that our method is not only trainable and stable, but also robust and effective, both in terms of model scale and task type.

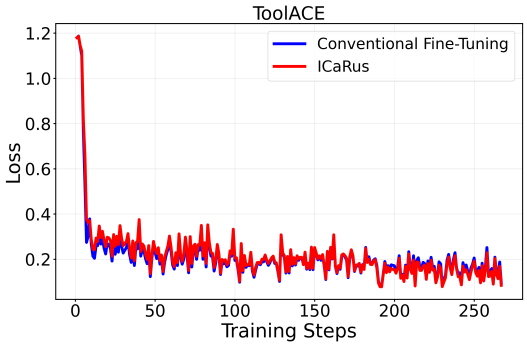


Figure 7: Training loss curves of conventional fine-tuning and ICaRus, both applied with LoRA on Qwen-3-32B, trained on the ToolAce dataset.

Table 5: Comparison of conventional fine-tuning and ICaRus when training Qwen3-32B on the ToolAce dataset.

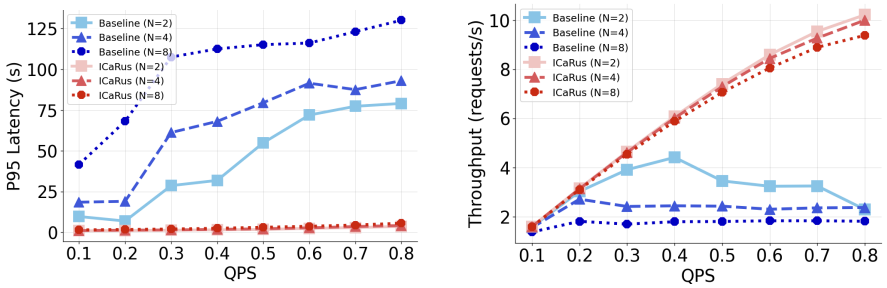
Model	Method	BFCL Non-live (AST)		
		Simple Python	Simple Java	Simple JavaScript
Qwen3-32B	Baseline	**96.5**	62.0	74.0
	ICaRus (Ours)	94.5	**63.0**	**76.0**

E ICaRus under Swap-based KV Cache Management

We conducted experiments with swap enabled (4GB swap space) using an earlier version of vLLM that supports this feature. The experimental results are reported below.

Figure 8b shows that ICaRus continues to provide lower P95 latency and higher throughput even when the multi-model system uses swap for KV cache management. In particular, with 8 LoRA modules, ICaRus achieves up to 12.1× lower P95 latency and 3.8× higher throughput than the baseline. This is because ICaRus reduces the KV cache footprint itself, so that even at higher QPS the GPU does not saturate and expensive swap operations are rarely triggered in the first place.

In summary, we emphasize that recompute/swap strategies and ICaRus address orthogonal aspects of the problem. Concretely, recompute or swap determine how to manage KV cache once GPU memory becomes full (e.g., whether to evict and reload from host storage or to recompute), whereas ICaRus fundamentally reduces KV pressure by enabling cross-model KV sharing across task-specialized models. By avoiding redundant KV construction across models, ICaRus effectively



(a) P95 latency across QPS

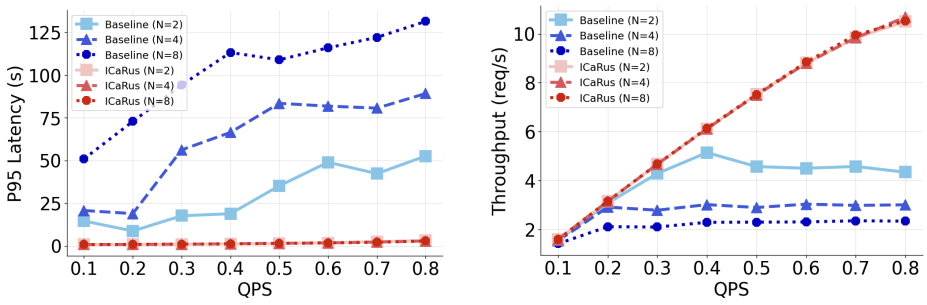
(b) Throughput across QPS

Figure 8: P95 latency and throughput of ICaRus compared with multiple task-specific agents fine-tuned from the LLaMA-3.1-8B base model under the ReAct workflow with swap-based KV cache management. Here, N denotes the number of LoRA modules, which are integrated into multi model system built using either the conventional approach or ICaRus.

delays or mitigates the point at which the KV cache saturates GPU memory, thereby improving performance regardless of whether the underlying system chooses recompute or swap as its eviction policy. In principle, ICaRus could also be combined with swap-based KV management.

F Performance under Random and Skewed Agentic Pattern in Real-World Scenarios

We evaluate the scenario in which the controller invokes agents at random with a skewed workload under ReAct workflow, so that on a typical turn only a subset of agents is active, better reflecting such real-world scenarios. Specifically, unlike the round-robin invocation pattern in Section 4.3, we construct a skewed workload in which one agent is invoked with probability 50% on each turn, while the remaining agents share the rest of the probability mass and are invoked in a random order rather than a fixed sequence. The experiments are conducted on the vLLM v0 architecture and the results are reported below.



(a) P95 latency across QPS

(b) Throughput across QPS

Figure 9: P95 latency and throughput of ICaRus compared with multiple task-specific agents fine-tuned from the LLaMA-3.1-8B base model under the ReAct workflow where the agent invocation pattern is random and skewed. Here, N denotes the number of LoRA modules, which are integrated into multi model system built using either the conventional approach or ICaRus.

Fig. 9 shows that ICaRus maintains low P95 latency and high throughput under dynamic and skewed agentic patterns. For example, with 2 models at 0.4 QPS, ICaRus achieves 15× lower P95 latency and 1.2× higher throughput than the baseline, demonstrating that the core advantage of ICaRus, enabling per-model prefix caching on top of cross-model KV sharing, is preserved even under skewed and random agent invocation patterns. Furthermore, in the baseline, throughput quickly saturates beyond a certain QPS because rapid growth of the KV cache triggers frequent evictions and recomputations. In contrast, ICaRus allows multiple models to share a single KV cache pool, keeping entries within the available GPU memory budget without eviction so that throughput continues to increase with QPS without saturation. As a result, in the 8-model setting, ICaRus achieves up to 3.5× higher throughput than the baseline under skewed and dynamic agent invocation patterns.

Table 6: Comparisons between DroidSpeak and ICaRus. α, N, and L represent recomputation ratio, the number of models, and total sequence length.

Aspect	DroidSpeak	ICaRus (Ours)
Training	Models are trained independently **without considering KV-cache sharing**.	Models are trained **with considering KV-cache sharing**.
KV-cache sharing	Reuses KV caches only in a **subset of "non-sensitive" layers**.	Reuses identical KV caches at **all layers**.
Accuracy	Accuracy degrades as more layers reuse shared KV caches.	Accuracy is preserved even when all layers share KV caches.
Robustness	Low robustness: **calibration-chosen "important" layers may not match those under real traffic**, leading to unpredictable quality drops in production.	High robustness: models are trained under full KV-cache sharing, so **KV caches can be fully shared at inference time just as during training**, without unexpected degradations in response quality.
KV memory	$\mathcal{O}((1+\alpha(N-1))L)$	$\mathcal{O}(L)$
# Prefill Compute	$\mathcal{O}((1+\alpha(N-1))L^2)$	$\mathcal{O}(L^2)$
Fine-tuning cost	o	o
Selective recomputation cost	o	x