Streamlining the Collaborative Chain of Models into A Single Forward Pass in Generation-Based Tasks

Anonymous ACL submission

Abstract

In Retrieval-Augmented Generation (RAG) and agent-based frameworks, the "Chain of Models" approach is widely used, where multiple specialized models work sequentially on distinct sub-tasks. This approach is effective but increases resource demands as each model must be deployed separately. Recent advancements attempt to address this by applying prompt tuning, which allows a shared base model to adapt to multiple tasks with minimal parameter changes. However, a key challenge remains: intermediate outputs, passed between models as plain text, require recomputation of hidden states (i.e., Key and Value (KV) states in Transformers) during inference. In this paper, we introduce FTHSS, a novel prompt-tuning method that enables models to share KV hidden states, eliminating redundant forward passes and reducing KV cache storage. By modifying input and attention masks during training, FTHSS allows models to effectively utilize KV hidden states from prior models in both single- and multi-round scenarios. Empirical results on four tasks show that FTHSS matches the performance of traditional model chains while improving inference efficiency. [1]

1 Introduction

In many Retrieval-Augmented Generation (RAG) and agent-based frameworks (Lewis et al., 2020), multiple Large Language Models (LLMs) often collaborate sequentially. Each model focuses on a specific sub-task and passes its output as input to the next model until the task is completed(Zhang et al., 2024b). For instance, some RAG post-retrieval optimization methods (Xu et al., 2023; Kim et al., 2024) involve summarizing retrieved documents with a summarization model, and then generating answers with a question-answering model. These stepwise approaches leverage the strengths of individual models and have proven effective in many scenarios. As a result, the "Chain of Models" approach has gained popularity (Zhang et al., 2024b).

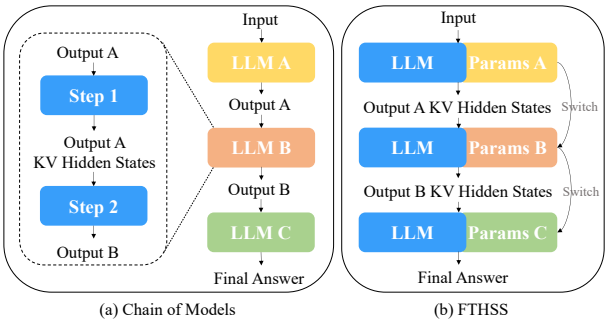


Figure 1: Comparison of "Chain of Models" (a) and FTHSS (b): In (a), models sequentially pass outputs as plain text, requiring KV recomputation. In (b), FTHSS shares KV hidden states, reducing redundant forward passes. PEFT methods allow the deployment of multiple models on a single device, with parameters changing, so there is no communication overhead for hidden states.

Deploying every specialized LLM in such chains significantly increases the resources needed. To address this, researchers have explored parameter-efficient fine-tuning (PEFT) methods, such as prompt tuning (Liu et al., 2021) and LoRA (Hu et al., 2021). These techniques allow fine-tuning with a fraction of the parameters when training. During inference, a shared base model is deployed on a single device and handles multiple tasks with distinct parameter configurations. This approach merges "Chain of Models" workflows into a single architecture, adapting to various sub-tasks through selective parameter usage. However, a critical bottleneck remains: in the chain, the intermediate key-value(KV) hidden states from one model cannot be directly reused by the next model due to parameter differences. As a result, communication between models in the chain relies on passing plain text, forcing the downstream model to recompute hidden states. This practice not only adds computational overhead, but also raises KV cache storage

[1] Code: https://anonymous.4open.science/r/FTHSS-8367/.

requirements for each model in the chain, further hampering efficiency.

In this paper, we argue that such recomputation is unnecessary. Even with parameter differences, the KV hidden states produced by one model should only differ marginally from those recalculated by the next. Particularly in prompt-tuning methods, the KV hidden states produced by the previous model are essentially conditioned on a few noisy tokens. With appropriate fine-tuning, the subsequent model can effectively interpret and utilize the KV hidden states of the previous model despite these noises, as Figure 1 shows.

To realize this vision, we propose FTHSS(Fine-Tuning for Hidden State Sharing), a prompt-tuning-based method that enables models in a chain to share KV hidden states. Specifically, when fine-tuning the model in single-round scenarios, where each model is invoked only once, we use KV hidden states from the prior models as input rather than plain text. This training approach requires extensive storage and access to KV hidden states, which may potentially increase training time and storage demands. To mitigate this, we introduce an online optimization strategy. By modifying the input and attention mask for each layer, we recompute the prior model's KV hidden states in memory during training, thus avoiding the overhead of storage and access. In multi-round scenarios, where models in the chain are invoked repeatedly, each model must adapt to the KV hidden states of others, so all models in the chain are trained synchronously to ensure mutual adaptation. After fine-tuning, models can dynamically switch learnable prompt tokens during inference, adapting based on task requirements, while leveraging precomputed KV cache for direct generation. And since prompt-tuning-based methods enable the deployment of multiple models on a single device, communication overhead for hidden states is effectively eliminated.

Empirical results on four tasks, including single-round and multi-round, demonstrate that FTHSS leads to a comparable performance to the chain of models, while enhancing inference efficiency. Technical contributions of this paper can be summarized as follows:

- To the best of our knowledge, we are the first to streamline the chain of models by sharing KV hidden states, thereby reducing the need for recomputing intermediate results.
- We introduce a prompt-tuning-based training strategy, FTHSS, that supports KV hidden state sharing across models in both single-round and multi-round scenarios.
- Experimental results show that FTHSS maintains comparable performance while significantly reducing inference latency and eliminating redundant KV cache storage.

2 Related Work

2.1 Chain of Models

The Chain of Models approach sequentially links specialized models, using the output of one as the input for the next (Zhang et al., 2024b). This method allows for incremental processing of subtasks, and has been widely adopted across various domains. For example, Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) improves the performance of question-answering (QA) tasks by combining retrieval and generation models. Additionally, the Chain of Models framework has proven highly effective for mathematical reasoning(Sun et al., 2023; Dong et al., 2024; Lei et al., 2024) and long-text generation (Xi et al., 2025; Wang et al., 2024).

While leveraging specialized models improves performance, it also increases deployment costs. One optimization strategy is to consolidate multiple models into a single, unified model through distillation. For instance, GritLM (Muennighoff et al., 2024) enables task-switching through instruction modifications, combining retrieval and generation. OneGen (Zhang et al., 2024a) introduces retrieval tokens, allowing LLMs to handle both tasks in a single forward pass. RankRAG (Yu et al., 2024) integrates ranking and generation into a single retrained model. However, these methods require the distilled model to perform well in multiple tasks, which remains a significant challenge. The FTHSS method proposed in this paper diverges from the distillation paradigm, and it still leverages the strengths of multiple models while reducing the demand for computing resources.

2.2 KV Cache Compression and Sharing

Large Language Models (LLMs) face significant bottlenecks due to high memory and computational demands, with the key-value (KV) cache being a major contributor. The KV cache stores the keys and values for each Transformer layer during generation to avoid redundant computations. During

deployment, the KV cache can occupy over 30% of GPU memory (Kwon et al., 2023).

Some straightforward approaches address this issue by compressing context length (Ge et al., 2023; Jiang et al., 2023a; Li et al., 2023) or employing sparse attention matrices (Xiao et al., 2023; Han et al., 2023). More recently, methods focusing on KV cache reuse have been proposed. YOCO (Sun et al., 2024) utilizes a cross-decoder mechanism with cross-attention to reuse cached values, allowing the model to store KV pairs only once while maintaining global attention capabilities. LCKV (Wu and Tu, 2024) and KVSharer (Yang et al., 2024) enable KV cache sharing across layers within the same model. While these methods effectively enhance model efficiency by reusing and sharing KV caches at different layers of a single model, FTHSS extends this concept to multiple models.

3 Methodology

In this section, we begin by highlighting a key challenge: model chains rely on text-based communication, which prevents the direct transfer of KV hidden states between models. We then explore the feasibility of fine-tuning the downstream model to process KV hidden states from the upstream model, although these hidden states often include noise tokens irrelevant to the downstream task. Lastly, we propose training strategies, FTHSS, to achieve KV hidden state sharing.

3.1 Preliminary

Multiple models $M_1, M_2, \ldots, M_n$ often collaborate sequentially in RAG and agent-based tasks, with each model M_i handling a specific task component. Specifically, model M_i processes the output T_{i-1} from the previous model, along with its unique input x_i, to produce output T_i for the next model. This process is expressed as: $T_i = M_i(T_{i-1}, x_i)$, forming a chain of models.

Given the high cost of deploying all models in such a chain, we can adopt a prompt-tuning approach. A shared base model M_θ is fine-tuned to perform different tasks, with each model M_i distinguished solely by its fine-tuned prompt tokens P_i. This approach allows us to deploy only M_θ, dynamically adjusting prompt tokens to replicate the behavior of multiple models:

$$T_i = M_\theta(T_{i-1}, x_i, P_i). \quad (1)$$

While this approach simplifies the model chain, communication between models still occurs via text. Upon receiving the output T_{i-1} from the previous model, each model M_i recalculates the hidden state of T_{i-1} based on its prefix P_i(a process known as "prefilling"), and then generates the output T_i and the corresponding hidden states O_{T_i} autoregressively(a process known as "decoding"):

$$H_{T_{i-1}}, H_{x_i}, H_{P_i} = \text{Prefilling}(T_{i-1}, x_i, P_i), \quad (2)$$

$$T_i, O_{T_i} = \text{Decoding}(H_{T_{i-1}}, H_{x_i}, H_{P_i}), \quad (3)$$

where T_i is the output text and O_{T_i} is the output hidden states of T_i.

In this paper, we argue that recalculating the KV hidden state $H_{T_{i-1}}$ is unnecessary. Instead, model M_i can directly use KV hidden states $O_{T_{i-1}}$ output by the previous model M_{i-1} as inputs. Besides, since prompt tuning allows the deployment of multiple models on a single device, there is no communication overhead of hidden states.

3.2 Fine-Tuning for Hidden State Sharing

Based on the above analysis, we aim to ensure that the KV hidden states computed by the previous model can be directly interpreted by the next. This is feasible due to the minimal differences between H_{T_i} and O_{T_i}. Since models fine-tuned with prompt tuning on the same base model share identical structures and parameters, they differ only in the fine-tuned prompt tokens and input data.

Specifically, the output KV hidden state of M_i during generation of the $j+1$-th token:

$$_, O_{T_{i,j}} = \text{Decoding}(T_{i,1:j}, H_{T_{i-1}}, H_{P_i}), \quad (4)$$

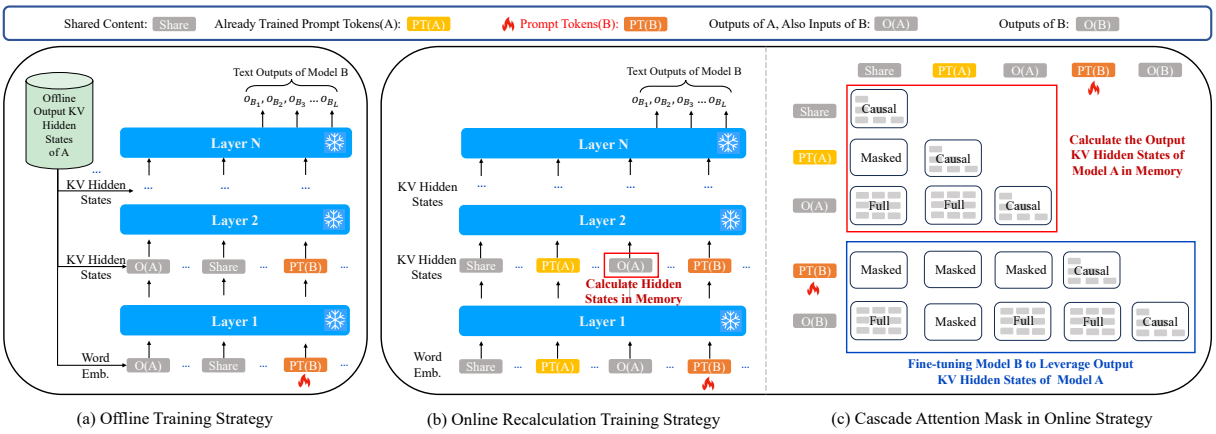
where $O_{T_{i,j}}$ is the hidden state of token $T_{i,j}$ output by M_i. We ignore the unique input for simplicity.

When $T_{i,j}$ serves as the input of M_{i+1} rather than the output of M_i, the KV hidden state must be recomputed:

$$H_{T_{i,j}} = \text{Prefilling}(T_{i,1:j}), \quad (5)$$

where $H_{T_{i,j}}$ is the KV hidden state of token $T_{i,j}$ calculated by M_{i+1}. We omit the prefix P_{i+1} as it can be appended after $T_{i,1:j-1}$.

Since the attention calculation method is the same in both prefilling and decoding stages, the difference between equations (4) and (5) is minimal, with only the prefixes and inputs differing. This suggests that the output hidden state of M_i introduces minimal noise for M_{i+1}, and fine-tuning may be a practical solution.



(a) Offline Training Strategy (b) Online Recalculation Training Strategy (c) Cascade Attention Mask in Online Strategy

Figure 2: An example of fine-tuning model B in the model chain A → B. For simplicity, the unique inputs of model A and model B are omitted. **Left:** Offline fine-tuning, where the output KV hidden states of fully trained model A are stored and used as input for model B. **Middle:** Online, where the output KV hidden states of model A are recalculated in memory. **Right:** We calculate the output KV hidden states of model A in memory and fine-tune model B by adjusting the attention mask for each layer. We use the online training strategy in practical applications.

We propose FTHSS (Fine-Tuning for Hidden State Sharing), a fine-tuning method to minimize these differences. By fine-tuning model M_i with noisy KV hidden states from model M_{i-1} as input, rather than the original ones, performance can be maintained despite the noise. We are currently exploring the implementation of this process.

3.2.1 Fine-Tuning Strategies of Single-Round

In practical applications, model chains are deployed in two configurations: single-round, where each model is called once, and multi-round, where models may be invoked multiple times. These configurations require distinct fine-tuning strategies.

Consider a model chain consisting of A and B in a single-round scenario, where model A precedes model B, and its output serves as B's input. The training data and processes are organized as:

Model Input Since model A is the first in the chain, it does not require adjustment to any preceding model's input. Thus, the fine-tuning data for model A follows standard prompt tuning. However, we refine this process by reordering the input:

- Model A input order: shared content tokens, learnable prompt tokens (A), unique input content tokens for A.

We place the shared content before the learnable prompt tokens. Since the shared content is used across all models in the chain, this arrangement ensures that the KV hidden states of the shared content remain unaffected by the learnable tokens, thereby preventing the introduction of noise.

Since the output of model A serves as the input for model B, A must be fully fine-tuned before fine-tuning B. Besides, the input to model B should consist of the output KV hidden states from A, rather than the tokens generated by A.

- Model B input order: shared content tokens, output KV hidden states of fine-tuned model A, learnable prompt tokens (B), and unique content tokens for B.

Fine-Tuning Process As mentioned earlier, model B must be trained after model A, using the output KV hidden states from A. The fine-tuning process for the model chain proceeds as follows:

- Fine-tune A to generate output A.
- Store the output KV hidden states from the fully fine-tuned model A.
- Offline load the hidden states and fine-tune B to leverage them in generating output B.

When fine-tuning model B, the position ID should not start at 0. Since model A's hidden states already contain position information, the position IDs for model B should begin at $l + 1$, where l is the last position ID in model A. As the LLM in this paper employs relative position encoding (e.g., RoPE (Su et al., 2024)), the absolute position is not critical. Therefore, the position ID ranges $[0, 1, \ldots, l]$ and $[l + 1, l + 2, \ldots, 2l + 1]$ are equivalent for attention computation. The proof is provided in Appendix D.

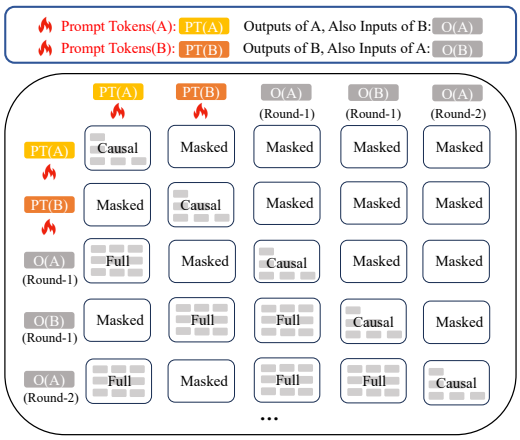


Figure 3: Cascade attention mask for every layer in the multi-round scenario.

Fine-Tuning Tricks to Save Storage Given that most existing LLMs are based on the Transformer architecture, they typically include numerous layers and attention heads. As Figure 2(a) shows, the approach described above requires storing and accessing a large number of KV hidden states, which can be impractical. To address this, we propose recomputing the output KV hidden states of model A in memory, rather than storing them offline, as illustrated in Figure 2(b).

Specifically, during the training of model B, we modify the input to B as follows:

- Model B input order: shared content tokens, fine-tuned prompt tokens (A), unique input content tokens for A, output tokens of A, learnable prompt tokens (B), and unique content tokens for B.

Notably, We incorporate the fine-tuned prompt tokens (A), along with both the input and output tokens of model A, as part of model B's input. By adjusting the attention mask, we calculate model A's output KV hidden states in memory (red box in Figure 2(c)). Simultaneously, the learnable prompt tokens (B) are fine-tuned to generate model B's output, using the recalculated KV hidden states from model A (blue box in Figure 2(c)).

The above algorithm outlines the fine-tuning process for a simplified model chain A→B. In practical applications, when more than two models are involved in a model chain, each model can be trained sequentially, following the order of the chain. During this process, each model's input and attention mask should be adjusted accordingly.

3.2.2 Fine-Tuning Strategies of Multi-Round

In a multi-round scenario, models may be invoked sequentially multiple times, allowing for more complex chains, such as $A \rightarrow B \rightarrow A \rightarrow B$. In this context, model B must adapt to the output of model A, while model A must also adapt to the output of model B. This differs from a single-round scenario, since models must be fine-tuned simultaneously.

To address this challenge, we modify the inputs and attention masks for both models, as illustrated in Figure 3. Specifically, the prompt tokens for both models are positioned at the beginning of the input. When computing the loss on the output of model A, attention scores are computed while masking the prompt tokens of model B. Conversely, When computing the loss on the output of model B, the prompt tokens of model A are masked. This ensures that model A's tasks are guided solely by its own prompt tokens, while model B's tasks are directed by its respective prompt tokens.

4 Experiments

4.1 Setup

We conduct experiments on both single-round and multi-round tasks. These experiments aim to evaluate whether the FTHSS approach can retain the functionality of model chains while improving inference efficiency in various scenarios.

4.1.1 Single-Round Evaluation

Tasks Many RAG frameworks involve chains of models due to their modular nature, making them suitable for our evaluation. Common RAG optimization methods include pre-retrieval and post-retrieval optimization. We select two tasks from each of them as benchmarks:

- Context Compression & Question Answering
- Query Rewriting & Question Answering

The Context Compression & QA task involves compressing retrieved content into a noise-free context for the final response. The Query Rewriting & QA task rewrites the query to retrieve more relevant information, and then generates the final response.

For the training data of Context Compression & QA task, we follow the data specified in RecComp(Xu et al., 2023), while for the training data of Query Rewriting & QA task, we adhered to the data outlined by Ma et al. (2023).

Task (→)	Context Compression & QA					
Dataset (→)	HQA		TQA		NQ	
Metric (→)	EM	F1	EM	F1	EM	F1
Single Model						
Native	14.4	22.8	40.1	53.7	14.5	26.4
Standard RAG	24.0	36.2	47.0	58.3	28.5	44.8
Prompt Tuning	26.0	36.2	26.4	44.2	32.7	45.1
Chain of Models						
Compress&QA	**30.4**	**43.8**	**59.7**	**68.3**	**35.0**	**48.3**
Streamlining						
Distill	28.3	42.1	54.3	63.9	21.4	33.1
FTHSS(Our)	**29.0**	**42.2**	**59.3**	**67.5**	**35.8**	**45.6**

Table 1: Performance on the single-round task: Compression&QA for FTHSS and other methods. **Bold numbers** indicate the best performance, except for the original chain of models (denoted in gray). Same below.

Task (→)	Query Rewriting & QA			
Dataset (→)	HQABM25		2WikiBM25	
Metric (→)	EM	F1	EM	F1
Single Model				
Native	13.4	19.5	13.8	21.4
Standard RAG	19.0	31.1	14.4	21.6
Prompt Tuning	18.2	29.8	20.6	27.4
Chain of Models				
Rewrite&QA	**27.0**	**37.2**	**24.4**	**30.2**
Streamlining				
Distill	20.8	30.4	18.0	23.9
FTHSS(Our)	**27.4**	**36.6**	**24.0**	**29.9**

Table 2: Performance on the single-round task: Query Rewrite&QA for FTHSS and other methods.

Baselines In the experiment, we compare three types of methods: (1) direct answer from a single model (Native, Standard RAG, Prompt Tuning), (2) using a model chain to generate intermediate results, which are then used to provide the final answer (Compress&QA, Rewrite&QA), and (3) simplifying the model chain to perform similarly to a single model (Distill, FTHSS). Distill refers to fine-tuning one model to generate all intermediate steps, effectively distilling the capabilities of multiple models into a single model. We use Llama-3-8B (Dubey et al., 2024) as the base model for all models in the chain. To ensure a fair comparison, all fine-tuning techniques discussed in this paper employ prompt tuning (Liu et al., 2021).

Datasets We use the following widely adopted datasets to validate our approach: Natural Questions (NQ)(Kwiatkowski et al., 2019), TriviaQA (TQA)(Joshi et al., 2017), 2WikiMultiHopQA(2Wiki) (Ho et al., 2020) and HotpotQA (HQA)(Yang et al., 2018).

4.1.2 Multi-Round Evaluation

Tasks In multi-round scenarios, models in a chain are invoked repeatedly. We selected "Reasoning & Memory" as a validation task (Jin et al., 2024), which decomposes the inference process into two iterative steps: (1) memory recall, retrieving relevant knowledge from the model's memory, and (2) reasoning, applying logical operations to the recalled knowledge. Additionally, we evaluate our methods on an active retrieval augmented generation task (Jiang et al., 2023b). The Active RAG task involves multiple rounds of retrieval, which actively decides what to retrieve across the course of the generation.

For the training data of Memory&Reasoning task, we use the data from Jin et al. (2024), while for the training data of Active RAG task, we follow the data proposed by Lyu et al. (2024).

Baselines The multi-round baselines are essentially identical to the single-round approach. They are categorized into three types: (1) direct answering (Single Model), (2) using a model chain to generate intermediate results (Memory&Reason, Plan&Generation), and (3) simplifying the model chain (Distill, FTHSS).

Datasets We take the following widely adopted datasets for evaluation: StrategyQA (Geva et al., 2021), TruthfulQA(TruthQA) (Lin et al., 2021), CommonsenseQA(ComQA) (Talmor et al., 2018), PubHealth (Zhang et al., 2023), 2WikiMultiHopQA(2Wiki) (Ho et al., 2020) and HotpotQA (HQA)(Yang et al., 2018).

For more details, we explain each task and other hyper-parameters in the Appendix A and B.

4.2 Main Results

FTHSS leads to a comparable performance with the chain of models in both single-round and multi-round scenarios. We benchmark FTHSS with other models in Table 1 and 2 in single-round settings, and find that FTHSS outperforms all single models while achieving comparable performance to the chain of models. This demonstrates that our method avoids repeated computation of intermediate KV hidden states, improving efficiency

Task (→)	Memory & Reasoning		
Dataset (→)	StrategyQA	ComQA	TruthQA
Metric (→)	Acc	Acc	Acc
Single Model			
Zero-shot	63.0	57.9	39.0
CoT	63.0	66.1	47.6
Prompt Tuning	63.6	66.7	65.2
Chain of Models			
Memory&Reason	70.1	71.3	69.2
Streamlining			
Distill	65.1	62.3	65.2
FTHSS(Our)	**69.2**	**70.3**	**68.9**

Table 3: Performance on the multi-round task: Memory & Reasoning for FTHSS and other methods. We evaluate the performance on multiple-choice questions using accuracy as the metric.

Task (→)	Active RAG	
Dataset (→)	Pubhealth	2WikiBM25
Metric (→)	Acc	F1
Single Model		
Native	69.5	21.4
Standard RAG	56.1	21.6
Prompt Tuning	69.1	27.4
Chain of Models		
Plan&Generation	73.4	33.6
Streamlining		
Distill	70.1	23.1
FTHSS(Our)	**72.0**	**31.9**

Table 4: Performance on the multi-round task: Plan & Retrieval for FTHSS and other methods.

without sacrificing performance.

For instance, in the Context Compression&QA task on the TQA dataset, FTHSS achieves an EM score just 0.4 points lower than the approach using separate models, demonstrating nearly identical performance. Importantly, the compressed context no longer requires a forward pass through the QA model. Instead, it directly leverages the KV hidden states output by the compression model, reducing redundant computations and inference time.

Table 3 and 4 present the results of multi-round experiments, which align closely with the findings from single-round experiments. This consistency highlights that, in addition to eliminating redundant intermediate computations, our method also removes the necessity of storing KV caches for individual models within the chain.

The chain of models outperforms single models. As shown in Tables 1, 2, 3, and 4, methods like Compress&QA and Query Rewrite&QA, which generate intermediate results, outperform single-model approaches. This highlights the potential of chain-of-model collaboration. Our FTHSS method further optimizes this by reducing redundant computations, yielding significant efficiency gains.

FTHSS outperforms Distill in both single-round and multi-round scenarios. While Distill attempts to fine-tune a single model to handle all intermediate steps, distilling multiple models' capabilities into one, this approach presents notable challenges. It requires the model to excel across all intermediate tasks; otherwise, the final result may be compromised. As shown in Table 1, 2, 3, and 4, experimental results reveal that distilling the capabilities of multiple models into a single model leads to varying degrees of performance degradation in both single-round and multi-round tasks. This underscores the superiority of FTHSS, where each model is allowed to specialize in its strengths, resulting in improved overall performance.

4.3 Inference Efficiency Improvements

To demonstrate the efficiency of our method, we present latency speed-ups achieved by eliminating redundant forward passes over intermediate results. We compare the inference latency of model B in FTHSS with that of the original model chain (where model A's output serves as input to model B), evaluating various intermediate result lengths. Results are averaged over 10 runs, performed on an Nvidia L20 GPU with the Llama-3-8B architecture.

Table 5 shows that for input sequences of 3,000 tokens, FTHSS reduces inference latency to less than one-third of the original model's. This improvement demonstrates that FTHSS maintains accuracy while significantly reducing latency. For sequences of 250 tokens, however, the speed-up is minimal due to GPUs' efficient parallel processing, limiting acceleration for smaller token counts.

In multi-round tasks, where each model in the chain may be repeatedly invoked, multiple copies of KV Caches are typically stored. FTHSS addresses this by enabling shared KV hidden states across models, reducing KV Cache storage to a single instance, regardless of chain length. As shown in Table 5, FTHSS significantly reduces GPU memory usage compared to a standard model

Inference latency(s)(single-round task)		
tokens	Chain of models	FTHSS
250	0.45	0.41
500	0.52	0.42
1000	0.66	0.43
3000	1.44	0.46
KV cache size(MB)(multi-round task)		
Models	Chain of models	FTHSS
1	137.5	137.5
2	$137.5 * 2$	137.5
3	$137.5 * 3$	137.5

Table 5: **Top:** Inference latency of model B in the chain A $\rightarrow$ B, with varying intermediate result lengths (in tokens), while output length is fixed at 16. **Bottom:** GPU memory occupancy for KV cache under varying model counts in multi-round tasks, with total length fixed at 1000. Latencies are measured on an NVIDIA L20, with KV states stored in bfloat16.

chain. For the Llama-3-8B architecture, the KV Cache size for an input sequence of 1000 tokens is 137.5 MB. When multiple models are used, FTHSS saves $(n-1) \times 137.5$ MB of GPU memory. Thus, in multi-round tasks, FTHSS not only eliminates redundant computations, reducing latency, but also removes the need for multiple KV Cache copies, resulting in substantial memory savings.

4.4 Further Analysis

In practice, specialized models are already trained using methods like Prompt Tuning. To apply our approach and simplify the model chain, these models may require re-fine-tuning, which can be computationally expensive. This raises the question: can these trained models—trained on plain text instead of the KV hidden states of previous models—be used with minimal or no fine-tuning?

Figure 4 compares three approaches: (1) the standard prompt-tuned model without additional re-fine-tuning (Standard), which attempts to interpret the noisy KV hidden states of the previous model directly; (2) continuing fine-tuning the standard prompt-tuned model on 5,000 examples using FTHSS (5000 samples); and (3) fully fine-tuning the base model on the entire dataset with FTHSS (Fully FTHSS). Experiments on the Context Compression&QA task show that even the standard fine-tuned model generates mostly correct answers. On the TQA dataset, the F1 score of the standard model

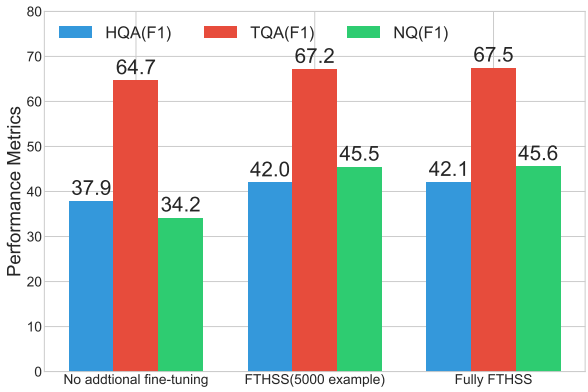


Figure 4: Performance comparison of three fine-tuning strategies on Context Compression & QA task: (1) No additional fine-tuning, using noisy KV hidden states directly; (2) FTHSS (5000 samples), where the standard-prompt-tuning model is fine-tuned on 5,000 examples; and (3) Fully FTHSS, where the base model undergoes full dataset fine-tuning.

is close to that of the fully fine-tuned model using FTHSS. However, performance drops on more complex datasets like HotpotQA and NQ due to noise in the KV hidden states. Additionally, fine-tuning models on a small dataset significantly improves performance. This suggests that fine-tuning standard prompt-tuned models on a small dataset using FTHSS is sufficient to mitigate noise, making full-dataset re-fine-tuning unnecessary.

As discussed in Section 3.2, passing KV hidden states between models mainly introduces unnecessary attention to noisy tokens. Previous work, such as attention sink (Xiao et al., 2023), has shown that attention exhibits sparsity properties, meaning that a few noisy tokens do not significantly impact the final output. Consequently, using a standard prompt-tuned model without further fine-tuning can still yield strong performance on simpler tasks.

5 Conclusion

In this paper, we introduced FTHSS, a method that enables models in a chain to directly share KV hidden states, eliminating redundant forward passes over intermediate results and reducing KV cache storage. By reordering the input and attention masks at each layer, FTHSS allows downstream models to leverage KV hidden states from upstream models. Our experiments demonstrate that FTHSS matches the performance of traditional model chains while significantly improving the inference efficiency in both single-round and multi-round scenarios.

6 Limitations

While our method is effective for open-source models, it cannot be directly applied to closed-source models that only provide API access, limiting its applicability in such settings. Additionally, because the method involves fine-tuning, experiments were not conducted on large models, such as those with 70B parameters, due to computational resource constraints. Future work should explore the generalizability of hidden state sharing methods in larger models and across diverse, high-quality, and challenging datasets.

References

Zixuan Dong, Baoyun Peng, Yufei Wang, Jia Fu, Xiaodong Wang, Yongxue Shan, and Xin Zhou. 2024. Effiqa: Efficient question-answering with strategic multi-model collaboration on knowledge graphs. *arXiv preprint arXiv:2406.01238*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Tao Ge, Jing Hu, Xun Wang, Si-Qing Chen, and Furu Wei. 2023. In-context autoencoder for context compression in a large language model. *arXiv preprint arXiv:2307.06945*.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. 2022. Accelerate: Training and inference at scale made simple, efficient and adaptable. https://github.com/huggingface/accelerate.

Chi Han, Qifan Wang, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2023. Lm-infinite: Simple on-the-fly length generalization for large language models. *arXiv preprint arXiv:2308.16137*.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023a. Llmlingua: Compressing prompts for accelerated inference of large language models. *arXiv preprint arXiv:2310.05736*.

Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023b. Active retrieval augmented generation. *arXiv preprint arXiv:2305.06983*.

Mingyu Jin, Weidi Luo, Sitao Cheng, Xinyi Wang, Wenyue Hua, Ruixiang Tang, William Yang Wang, and Yongfeng Zhang. 2024. Disentangling memory and reasoning ability in large language models. *arXiv preprint arXiv:2411.13504*.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*.

Jaehyung Kim, Jaehyun Nam, Sangwoo Mo, Jongjin Park, Sang-Woo Lee, Minjoon Seo, Jung-Woo Ha, and Jinwoo Shin. 2024. Sure: Summarizing retrievals using answer candidates for open-domain qa of llms. *arXiv preprint arXiv:2404.13081*.

Diederik P Kingma. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.

Bin Lei, Yi Zhang, Shan Zuo, Ali Payani, and Caiwen Ding. 2024. Macm: Utilizing a multi-agent system for condition mining in solving complex mathematical problems. *arXiv preprint arXiv:2404.04735*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Yucheng Li, Bo Dong, Chenghua Lin, and Frank Guerin. 2023. Compressing context to enhance inference efficiency of large language models. *arXiv preprint arXiv:2310.06201*.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2021. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958*.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2021. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*.

Songshuo Lu, Hua Wang, Yutian Rong, Zhi Chen, and Yaohua Tang. 2024. Turborag: Accelerating retrieval-augmented generation with precomputed kv caches for chunked text. *arXiv preprint arXiv:2410.07590*.

Yuanjie Lyu, Zihan Niu, Zheyong Xie, Chao Zhang, Tong Xu, Yang Wang, and Enhong Chen. 2024. Retrieve-plan-generation: an iterative planning and answering framework for knowledge-intensive llm generation. *arXiv preprint arXiv:2406.14979*.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query rewriting for retrieval-augmented large language models. *arXiv preprint arXiv:2305.14283*.

Niklas Muennighoff, Hongjin Su, Liang Wang, Nan Yang, Furu Wei, Tao Yu, Amanpreet Singh, and Douwe Kiela. 2024. Generative representational instruction tuning. *arXiv preprint arXiv:2402.09906*.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Qiushi Sun, Zhangyue Yin, Xiang Li, Zhiyong Wu, Xipeng Qiu, and Lingpeng Kong. 2023. Corex: Pushing the boundaries of complex reasoning through multi-model collaboration. *arXiv preprint arXiv:2310.00280*.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. 2024. You only cache once: Decoder-decoder architectures for language models. *arXiv preprint arXiv:2405.05254*.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2018. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*.

Qiyao Wang, Shiwen Ni, Huaren Liu, Shule Lu, Guhong Chen, Xi Feng, Chi Wei, Qiang Qu, Hamid Alinejad-Rokny, Yuan Lin, et al. 2024. Autopatent: A multi-agent framework for automatic patent generation. *arXiv preprint arXiv:2412.09796*.

Haoyi Wu and Kewei Tu. 2024. Layer-condensed kv cache for efficient inference of large language models. *arXiv preprint arXiv:2405.10637*.

Zekun Xi, Wenbiao Yin, Jizhan Fang, Jialong Wu, Runnan Fang, Ningyu Zhang, Jiang Yong, Pengjun Xie, Fei Huang, and Huajun Chen. 2025. Omnithink: Expanding knowledge boundaries in machine writing through thinking. *arXiv preprint arXiv:2501.09751*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2023. Recomp: Improving retrieval-augmented lms with compression and selective augmentation. *arXiv preprint arXiv:2310.04408*.

Yifei Yang, Zouying Cao, Qiguang Chen, Libo Qin, Dongjie Yang, Hai Zhao, and Zhi Chen. 2024. Kvsharer: Efficient inference via layer-wise dissimilar kv cache sharing. *arXiv preprint arXiv:2410.18517*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*.

Yue Yu, Wei Ping, Zihan Liu, Boxin Wang, Jiaxuan You, Chao Zhang, Mohammad Shoeybi, and Bryan Catanzaro. 2024. Rankrag: Unifying context ranking with retrieval-augmented generation in llms. *arXiv preprint arXiv:2407.02485*.

Jintian Zhang, Cheng Peng, Mengshu Sun, Xiang Chen, Lei Liang, Zhiqiang Zhang, Jun Zhou, Huajun Chen, and Ningyu Zhang. 2024a. Onegen: Efficient one-pass unified generation and retrieval for llms. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 4088–4119.

Tianhua Zhang, Hongyin Luo, Yung-Sung Chuang, Wei Fang, Luc Gaitskell, Thomas Hartvigsen, Xixin Wu, Danny Fox, Helen Meng, and James Glass. 2023. Interpretable unified language checking. *arXiv preprint arXiv:2304.03728*.

Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Ö Arik. 2024b. Chain of agents: Large language models collaborating on long-context tasks. *arXiv preprint arXiv:2406.02818*.

Dataset name	Train/Test
Context Compression&QA task	
Natural Questions(NQ)	39,466/3,610
TriviaQA(TQA)	47,531/11,313
HotpotQA(HQA)	26,556/500
Query Rewrite&QA task	
Training Data (Ma et al., 2023)	37,520/-
2WikiMultiHop(2Wiki)	-/500
HotpotQA(HQA)	-/500
Memory&Reasoning task	
Training Data (Jin et al., 2024)	10,925/-
StrategyQA	-/687
TruthfulQA(TruthQA)	-/164
CommonsenseQA(ComQA)	-/1,221
Active RAG task	
Training Data (Lyu et al., 2024)	47,689/-
2WikiMultiHop(2Wiki)	-/500
Pubhealth	-/987

Table 6: Dataset statistics.

A Hyperparameters and Datasets

Hyperparameters. We fine-tune all parameters of our models for up to 3 epochs on 4 Nvidia A6000 GPUs. Our learning rate is 2e-4, and the gradient accumulation step is set to 8. We use 3% of steps for linear warm-up of the learning rate and decay it linearly to 0 over training. To save memory, we use DeepSpeed ZeRo-2 (Rajbhandari et al., 2020; Rasley et al., 2020) optimization, gradient checkpointing, and BF16 mixed precision training. During training, we use a maximum sequence length of 1224 for every sample, 100 learnable prompt tokens, and finetune using the Adam optimizer (Kingma, 2014) with no weight decay. Our training script is based on HuggingFace accelerate (Gugger et al., 2022) libraries.

All base models in this paper are Llama-3-8B-Base unless otherwise specified. All PEFT fine-tuning methods are based on Prompt tuning, with the number of learnable prompt tokens set to 100. For methods that do not involve model training (e.g., Native, Standard RAG, and CoT), we utilize Llama-3-8B-Instruct, as its instruction-following capability is essential for these approaches.

Datasets. The statistical details of the training and test datasets used in the experiments are provided in Table 6. In the Context Compression&QA task, the training phase utilizes the augmented NQ, TQA, and HQA datasets from recomp (Xu et al., 2023). These datasets were created by using ChatGPT to semantically compress retrieved documents into concise summaries, generating synthetic training data. For model evaluation, we use the full test sets of NQ and TQA, along with a subset of the HQA development set, as validation benchmarks to ensure a comprehensive and reliable assessment of model performance. In the Query Rewrite&QA task, we use the dataset from Ma et al. (2023) for training and evaluate the model on the multi-hop question datasets HQA and 2Wiki. In the Activate RAG task, we use the dataset from Lyu et al. (2024) for training and evaluate the model on the short-form QA dataset PubHealth and the multi-hop QA dataset 2Wiki.

As for the retrieved documents, by default, we use the top one document ranked by Contriever-MS MARCO (Izacard et al., 2021) on Wikipedia corpus from Dec. 20, 2018, which is done to ensure a fair comparison among all baseline models. In the Query Rewrite&QA and Active RAG tasks, we use the top one document ranked by the BM25 retrieval algorithm. Improving the retriever is not the primary focus of this work; therefore, the retriever selection criterion is to maintain consistency with the papers that proposed these tasks.

B Task Explanation

In this section, we provide detailed examples to demonstrate why the evaluation tasks used in this paper involve multiple models. Table 7 illustrates the Compression&QA task. The documents retrieved by RAG are often excessively long and contain a significant amount of noise, which can mislead the question-answering model if input directly. By first using a model to compress the documents and then providing its output as input to the question-answering model, the accuracy of the responses can be significantly improved. The model chain in the Compression&QA task is designed based on this approach, consisting of a summarization model whose output serves as the input to the question-answering model.

Table 8 presents the Query Rewriting&QA task. For complex problems such as multi-hop QA, directly using the question as a query often fails to retrieve the appropriate context. To address this, we utilize another model to rewrite the query, which is then used to retrieve more accurate contextual

Input: How many episodes are there in dragon ball z?(*NQ*)
Input of the compression model: <retrieved>April 5, 2009, the series premiered in Japan airing in Fuji TV. D̈ragon Ball Z Kaiẗreduced the episode count to 159 episodes (167 episodes internationally), from the original footage of 291 episodes. Damaged frames were removed, resulting in some minor shots being remade from scratch in order to fix cropping, and others to address continuity issues. The majority of the international versions, including Funimation Entertainment's English dub, are titled D̈ragon Ball Z Kai.̈ premiered on Fuji TV on February 7, 1996 and ran until November 19, 1997 for 64 episodes. Unlike the first two anime series, it is not. Edition,ẅhich collects three individual volumes into a single large volume. However, in 2013 Viz began publishing new 3-in-1 volumes collecting the entire manga series, including what they previously released as D̈ragon Ball Z,̈ under the D̈ragon Ballñame. The D̈ragon Ball Z̈films comprise a total of 15 entries as of 2015...</retrieved>
Output of the compression model: <compressed>PD̈ragon Ball Z Kaiẗhas a total of 159 episodes (167 internationally), reducing the original count of 291 episodes. The last aired episode of D̈ragon Ball Kaiïn Japan was the 97th episode, with the 98th episode being released direct-to-video on August 2, 2011.</compressed>
Output of the question answering model: <final_answer>291 episodes</final_answer>

Table 7: An Example for NQ datasets in Compression&QA task. Model chain: compression model→question answering model.

Input: Which one was established first, Grouplogic or Inbios?(*2Wiki*)
Output of the query rewriting model: <query_rewrite>Grouplogic establishment year; Inbios establishment year</query_rewrite>
Input of the question answering model: <retrieved>GroupLogic, Inc., founded in 1988 and headquartered in Arlington, Virginia, USA, is an enterprise software company that develops, sells and supports software for moving and storing data including activEcho, mobilEcho, ArchiveConnect, MassTransit and ExtremeZ- IP... InBios International, Inc. is a medical diagnostic company based in Seattle that specializes in the detection of infectious diseases. The company was founded in 1996, and since its inception has developed several technologies useful in designing rapid and ELISA based immunodiagnostic assays. In April 2011, InBios International announced the clearance of its IgM test for dengue fever, DENV Detect IgM Capture ELISA...</retrieved>
Output of the question answering model: <Final answer>291 episodes</Final answer>

Table 8: An Example for 2Wiki datasets in Query Rewriting&QA task. Model chain: query rewriting model→question answering model.

information, followed by inputting this refined context into the question-answering model. The Query Rewriting&QA task also involves a model chain.

Tables 9 and 10 demonstrate the model chains in multi-round scenarios, where the models are not invoked only once but are iteratively called. In the Memory&Reasoning task, the model first recalls the knowledge required to answer the question, then uses this recalled knowledge to reason and generate the answer. Since these two sub-tasks differ significantly, different models must be deployed to handle them separately. Furthermore, a single round is insufficient to ensure that all required knowledge is retrieved, so these two sub-tasks need to be executed alternately and repeatedly. Additionally, the Active RAG task involves multiple rounds of retrieval, where the model dynamically decides what to retrieve during the generation process (the planning phase), followed by generating the response based on the retrieved information (the answering phase). The planning and answering sub-tasks are iteratively performed, requiring two distinct models to be deployed.

C Inference Details

Algorithm 1 illustrates the process of inference in a single-round task, where multiple prompt-tuning-based models share KV hidden states. The performance of a shared base model across different tasks depends on the learnable, task-specific prompt tokens. During inference, these prompt tokens are dynamically switched, as demonstrated in line 12 of the algorithm. Furthermore, sharing hidden states implies that the KV cache from the previous model can be reused directly, without the need to recompute the intermediate KV hidden states of Y_i. The red-striped portion in the algorithm shows the computational savings of our approach compared to previous prompt-tuning methods.

Assuming the total length of intermediate results

Input: Question: Sammy wanted to go to where the people were. Where might he go? Options: A.race track B.populated areas C.the desert D.apartment E.roadblock (*ComQA*)
Output: <reason_start>Let's reason with knowledge: The objective is to determine a location where there are many people, as per Sammy's desire to be where the people are. Each option needs to be evaluated based on the likelihood of it being a place with many people.<reason_end><memory_start>Let's recall the knowledge: A race track can be very crowded during events. However, it may not always be populated, so it lacks consistency in being populated. Populated areas are locations where there are a high number of people living or congregating, such as cities and towns. This option aligns closely with the goal of being where people are consistently present. Deserts are typically sparsely populated because of harsh living conditions, making them unlikely places for finding many people. While an apartment building may contain a number of residents, it is not as widely or consistently populated as broader populated areas like cities. A roadblock is usually not a gathering place for people. It might temporarily have a group or line of vehicles, but it is not known for large groups of people congregating for extended periods.<memory_end><reason_start>Let's reason with knowledge: Based on the assessments, option B (populated areas) is the most appropriate choice as it consistently fulfills the objective of being a place where there is a high number of people present. [The answer is]: B <reason_end>

Table 9: An Example for CommonsenceQA datasets in Memory&Reasoning task. The reasoning and memory models are iteratively called multiple times during inference.

Input: Which movie came out first Escape to Witch Mountain or Pete's Dragon?(*HotpotQA*)
Output: <plan_start>Escape to Witch Mountain release<plan_end><retrieved>Escape to Witch Mountain is a 1975 American fantasy-children's film, adapted from the 1968 science fiction novel of the same name written by Alexander H. Key</retrieved><answer_start>Escape to Witch Mountain came out first,<answer_end><plan_start>Pete's Dragon release<plan_end><retrieved>Pete's Dragon is a 2016 American fantasy comedy-drama adventure film directed by David Lowery, written by Lowery and Toby Halbrooks, and produced by James Whitaker. </retrieved><answer_start>before Pete's Dragon. <answer_end>[Combine]<answer_start>Escape to Witch Mountain<answer_end>

Table 10: An Example for HotpotQA datasets in Active RAG task.

is n, the computational savings of this algorithm are $O(n^2)$, given the quadratic complexity of the transformer.

Notably, the computational savings occur during the prefilling phase, which runs in parallel. Therefore, when the length of the intermediate results is relatively short, the savings have a minimal impact on the inference latency.

D Position ID Rearrangement

If l is the last position ID of the preceding model, the position encoding of the current model should begin at $l+1$, ensuring that the accuracy of the attention computation during inference is unaffected. This is because, under Rotary Position Embedding (RoPE), the position ID ranges $[0, 1, \ldots, l]$ and $[l+1, l+2, \ldots, 2l+1]$ are equivalent in attention computation. The proof of this conclusion is presented below.

We prove that RoPE computes attention based solely on the relative position $m-n$, independent of the absolute positions m or n. Given a query vector $\boldsymbol{q}_m$ at position m and a key vector $\boldsymbol{k}_n$ at position n, RoPE applies rotations:

$$\boldsymbol{q}_m = R_m \boldsymbol{q}, \quad R_m = \begin{bmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{bmatrix},$$
$$\boldsymbol{k}_n = R_n \boldsymbol{k}, \quad R_n = \begin{bmatrix} \cos n\theta & -\sin n\theta \\ \sin n\theta & \cos n\theta \end{bmatrix}, \tag{6}$$

where θ is a frequency parameter. The attention score is:

$$\text{Score}(m, n) = \boldsymbol{q}_m^\top \boldsymbol{k}_n = (\boldsymbol{q}^\top R_m^\top)(R_n \boldsymbol{k}). \tag{7}$$

Since rotation matrices are orthogonal ($R^\top R = I$), and satisfy $R_m^\top R_n = R_{n-m}$, the score can simplify to:

$$\text{Score}(m, n) = \boldsymbol{q}^\top R_{n-m} \boldsymbol{k}, \tag{8}$$

which depends only on $(n-m)$. For high-dimensional vectors, RoPE divides the vector into $d/2$ subspaces, applying rotations independently in each subspace:

$$R_m^{(i)} = \begin{bmatrix} \cos m\theta_i & -\sin m\theta_i \\ \sin m\theta_i & \cos m\theta_i \end{bmatrix}, \tag{9}$$

Algorithm 1 The Inference Process of FTHSS in Single-Round Tasks(The red-striped portion represents operations that are necessary for the original model chain, but are optimized and removed in FTHSS).

Input: Input sequence $X = (x_1, x_2, \ldots, x_n)$
Output: Sub-task output sequences $Y_1 = (y_{11}, y_{12}, \ldots)$, $Y_2, \ldots, Y_t$
Initialize:
- Decoder-only transformer T with parameters θ
- Task-specific soft prompt tokens $\{P_1, P_2, \ldots, P_t\}$
- KV Cache: Cache $\leftarrow T(X)$ (Encode input sequence)
- Intermediate results: Y_0

for $i = 1$ **to** t **do**
Prefilling Phase:
~~Cache $\leftarrow \emptyset$~~
~~Cache $\leftarrow T(P_i, Y_{i-1}, \text{Cache})$~~
Cache $\leftarrow T(P_i, \text{Cache})$ (Compute and cache task-specific KV)
Initialize output sequence: $Y_i \leftarrow$ [<start>]
Decoding Phase (Autoregressive):
for $k = 1$ **to** max_length **do**
1. Current token: $y_{k-1} \leftarrow Y_i[-1]$ (Last generated token)
2. Compute embedding: $e_k \leftarrow E(y_{k-1})$
3. Update decoder layers with KV Cache:
$h_k, \text{Cache} \leftarrow T(e_k, \text{Cache})$ (Reuse cached KV)
4. Compute logits: $p(y_k) \leftarrow \text{softmax}(W_o h_k)$ (h_k from last layer)
5. Sample next token: $y_k \sim p(y_k)$
6. Append y_k to Y_i

end for
end for
Return: Output sequences $Y_1, Y_2, \ldots, Y_t$

yielding:

$$\text{Score}(m, n) = \sum_{i=1}^{d/2} \boldsymbol{q}_i^\top R_{n-m}^{(i)} \boldsymbol{k}_i. \qquad (10)$$

Thus, RoPE strictly encodes relative positions, eliminating absolute position dependence. This property has been utilized in some precomputed KV cache scenarios (Lu et al., 2024).