

# APE: Faster and Longer Context-Augmented Generation via Adaptive Parallel Encoding

Xinyu Yang<sup>†</sup>, Tianqi Chen<sup>†‡</sup>, Beidi Chen<sup>†</sup>

<sup>†</sup>Carnegie Mellon University, <sup>‡</sup>Nvidia

Context-augmented generation (CAG) techniques, including RAG and ICL, require the efficient combination of multiple contexts to generate responses to user queries. Directly inputting these contexts as a sequence introduces a considerable computational burden by re-encoding the combined selection of contexts for every request. To address this, we explore the promising potential of parallel encoding to independently pre-compute and cache each context’s KV states. This approach enables the direct loading of cached states during inference while accommodating more contexts through position reuse across contexts. However, due to misalignments in attention distribution, directly applying parallel encoding results in a significant performance drop. To enable effective and efficient CAG, we propose Adaptive Parallel Encoding (APE), which brings shared prefix, attention temperature, and scaling factor to align the distribution of parallel encoding with sequential encoding. Results on RAG and ICL tasks demonstrate that APE can preserve 98% and 93% sequential encoding performance using the same inputs while outperforming parallel encoding by 3.6% and 7.9%, respectively. It also scales to many-shot CAG, effectively encoding hundreds of contexts in parallel. Efficiency evaluation shows that APE can achieve an end-to-end 4.5× speedup by reducing 28× prefilling time for a 128K-length context.



Github: <https://github.com/Infini-AI-Lab/APE>

Website: <https://infini-ai-lab.github.io/APE-Page>

## 1 Introduction

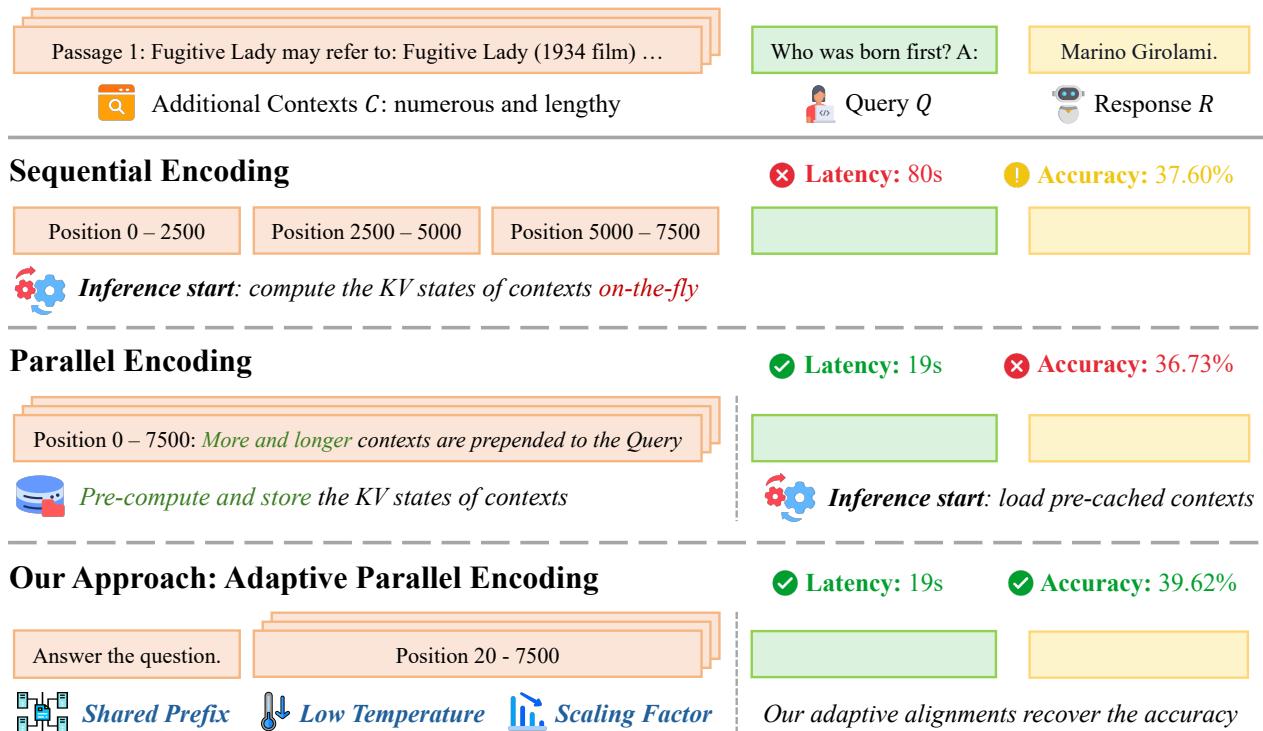
Recent advances in context-augmented generation (CAG) techniques, particularly retrieval-augmented generation (RAG) (Gupta et al., 2024; Gao et al., 2023) and in-context learning (ICL) (Dong et al., 2022; Wei et al., 2022), have been widely adopted in large language models (LLMs) (Dubey et al., 2024; Achiam et al., 2023), improving their ability to generalize to unseen tasks with contextual information, as demonstrated in Figure 1 (top). These techniques employ a *sequential encoding* process to ground LLM inputs with knowledge from external sources: concatenating the retrieved texts into one sequence, and encoding the sequence into key-value (KV) states as the context for subsequent queries. While this new, significantly longer input improves performance, the increased latency in context prefilling becomes a bottleneck in tasks that require long inputs but generate short outputs (Bai et al., 2023; Agarwal et al., 2024; Jiang et al., 2024b). For example, prefilling a 128K context takes 17 seconds, whereas generating 256 tokens requires only 6 seconds. This discrepancy leaves significant room to improve the practical efficiency of CAG systems in real-world deployments (Liu, 2022; Chase, 2022).

Since texts for CAG are typically stored independently in external databases (Zayarni et al., 2024; Douze et al., 2024), pre-caching all these texts for direct loading during inference offers a brute-force approach to accelerate CAG. However, for autoregressive LLMs, the KV states are inherently context-dependent. This dependency makes naive pre-caching impractical, as it would require caching all possible context permutations, leading to factorial growth in memory requirements as the database size increases. For instance, caching all permutations of just ten 256-token text chunks for the LLAMA-3-8B model would demand an impractical 22 PB of memory.

To address this issue, *parallel encoding* (Ratner et al., 2022; Yen et al., 2024; Li et al., 2024; Sun et al., 2024) is introduced to encode each context into KV states separately, ensuring that tokens from different contexts cannot attend to each other during encoding. Next, the on-the-fly generation starts by prefilling user queries, which can attend to the cached KV states from all contexts without re-encoding, offering two benefits:

**Pre-caching Contexts for Fast Inference:** Texts from external sources can be pre-computed and cached

## Problem Setup: Context-Augmented Generation



**Figure 1 Overview of Our Approach.** Context-augmented generation leverages additional contexts to improve LLM response quality to user queries. Sequential encoding prefills selected context chunks as a long sequence during inference, leading to high latency from on-the-fly re-encoding and low accuracy due to context window limitations. Parallel encoding offers an alternative method to pre-compute more and longer contexts within the same positional range but results in worse performance. To address these challenges, we propose Adaptive Parallel Encoding (APE) to re-align the attention weight distribution of parallel encoding with sequential encoding via three training-free steps: shared prefix, scaling factor, and adaptive temperature, leading to fast and accurate CAG systems in real-world applications.

into KV states, which serve as contexts for direct loading during inference. Additionally, this approach allows for cost-free manipulation of contexts, including operations like insertion, deletion, replacement, and swapping.

**Re-using Positions for Long Context:** Contexts can be inserted into the same range of positions in an LLM’s context window, allowing for more and longer context chunks. It also mitigates the problem of “lost in the middle” in context ordering (Liu et al., 2024a), as each context is equally “close” to the generated tokens.

Despite these advantages, parallel encoding leads to significant performance degradation across multiple RAG and ICL scenarios, as shown in Figure 2, with average declines of 4.9% (despite using 2-10 $\times$  more contexts) and 49.0%, respectively. While prior works (Sun et al., 2024; Yen et al., 2024) have attempted to correct this with fine-tuning, these methods continue to exhibit reduced accuracy in reasoning tasks (e.g., GSM8K). This decrease arises from the limited generalization capability of models fine-tuned on simple tasks to complex ones.

However, our results in Figure 2 also reveal that parallel encoding holds promise, as LLMs can still generate reasonable responses due to their inherent alignments with sequential encoding. Based on this observation, we aim to strengthen these alignments while addressing the remaining discrepancies to achieve more accurate parallel encoding. Our insight from Figure 3 and Figure 4 is that *KV states from independent contexts can be naturally merged into one sequence due to their similarity in direction and magnitude, attributed to the presence of an attention sink* (Xiao et al., 2023). This observation reduces our challenge to addressing residual misalignments, which manifest as anomalous distributions at the initial and recent positions within each context.

Motivated by this, we propose Adaptive Parallel Encoding (APE) to align the distribution between sequential and parallel encoding, which enables accurate and fast CAG (see Figure 1 (Bottom)). Our contributions involve:

- We systematically analyze the distribution properties of attention weights in parallel encoding, focusing on the magnitude and direction of KV states across various samples and positions. Our observations identify major alignments and minor misalignments between parallel and sequential encoding for further improvement.
- We propose APE to recover the accuracy of parallel encoding with three alignment steps: (i) Prepend a shared prefix to avoid the duplication of abnormal distribution of initial tokens. (ii) Adjust a lower attention temperature to sharpen the distribution, focusing on contextually important tokens. (iii) Apply a scaling factor to offset the increase in the magnitude of the LogSumExp value of attention scores from the context.
- We empirically show that (i) APE maintains 98% and 93% of the sequential encoding performance in RAG and ICL tasks, respectively. (ii) APE outperforms parallel encoding in RAG and ICL, yielding improvements of 3.6% and 7.9%, respectively. (iii) APE scales to handle hundreds of contexts in parallel, matching or exceeding sequential encoding in many-shot scenarios. (iv) APE accelerates long-context generation, achieving up to 4.5 $\times$  speedup through a 28 $\times$  reduction in prefilling time for a context including 128K tokens.

## 2 Background and Related Work

### 2.1 Context-Augmented Generation

This work explores CAG problems using LLMs, where user queries are enhanced with additional contexts from external databases. CAG typically involves two scenarios: RAG (Asai et al., 2024; Gupta et al., 2024; Gao et al., 2023), which focuses on directly retrieving relevant information, and ICL (Dong et al., 2022; Wei et al., 2022; Agarwal et al., 2024), which emphasizes further acquiring emergent capabilities from in-context examples.

### 2.2 Parallel Encoding

Next, we present the formulation of using parallel encoding in LLMs for CAG settings. Let  $\mathcal{S}$  represent the input sequence including  $N$  contexts  $C_1, \dots, C_N$  and one query  $Q$ . Formally, this can be denoted as:

$$\mathcal{S} = \underbrace{\{s_{C_1,1}, \dots, s_{C_1,l_1}\}}_{\text{Context 1}}, \underbrace{\{s_{C_2,1}, \dots, s_{C_2,l_2}\}}_{\text{Context 2}}, \dots, \underbrace{\{s_{C_N,1}, \dots, s_{C_N,l_N}\}}_{\text{Context N}}, \underbrace{\{s_{Q,1}, \dots, s_{Q,l}\}}_{\text{Query}}. \quad (1)$$

For simplicity, we can express this as:  $\mathcal{S} = \{S_{C_1}, S_{C_2}, \dots, S_{C_N}, S_Q\}$ . Given two models  $\Theta_{\text{Enc}}$  and  $\Theta_{\text{Dec}}$  (which may be the same model), a response  $\mathcal{R}$  is generated to the input  $\mathcal{S}$  using parallel encoding in two steps:

**Pre-caching Contexts.** The first step is to encode and cache the KV states for each context independently using  $\Theta_{\text{Enc}}$ . For a given context  $S_{C_i}$ , we compute its KV states offline as  $(K_{C_i}, V_{C_i}) = \Theta_{\text{Enc}}(S_{C_i})$  and store them for direct loading during inference. Specifically, we denote  $K_{C_i} = \{k_{C_i,1}, \dots, k_{C_i,l_i}\}$  and  $V_{C_i} = \{v_{C_i,1}, \dots, v_{C_i,l_i}\}$ .

**Generating Response.** Next, the user query is augmented by all relevant pre-cached KV states to generate the response:  $\mathcal{R} = \Theta_{\text{Dec}}(S_Q, K_C, V_C)$ , where  $K_C, V_C$  are subsets of  $\{K_{C_1}, \dots, K_{C_N}\}$  and  $\{V_{C_1}, \dots, V_{C_N}\}$ , respectively.

Parallel encoding significantly improves efficiency compared to sequential encoding by reducing the complexity of prefilling from  $O((l_1 + \dots + l_N + l_Q)^2)$  (i.e., quadratic) to linear concerning the total context length. With pre-caching, the cost becomes  $O((l_1 + \dots + l_N + l_Q) \cdot l_Q)$ . In the absence of pre-caching, the complexity is  $O(\max(l_1^2, \dots, l_N^2) + ((l_1 + \dots + l_N + l_Q) \cdot l_Q))$ , which remains efficient for multiple contexts of similar length.

Prior parallel encoding approaches vary in their design of  $\Theta_{\text{Enc}}$  and  $\Theta_{\text{Dec}}$ . Parallel Context Windows (PCW) (Ratner et al., 2022) directly employs pre-trained LLMs as both, resulting in significant performance drops. Block-Attention (Sun et al., 2024) further fine-tunes the model, successfully recovering performance in RAG tasks. Alternatively, CEPE (Yen et al., 2024) and FocusLLM (Li et al., 2024) train new Transformer-based encoders using encoder-only and decoder-only architectures, respectively. These methods also differ in  $\Theta_{\text{Dec}}$ : CEPE trains additional cross-attention layers for processing contexts, whereas other methods directly input the context into original self-attention layers. While these trainable methods show promising results in RAG tasks, challenges remain regarding their training overheads and generalization abilities to more complex ICL scenarios. Moreover, applying parallel encoding in CAG can be viewed as a kind of memory-augmented neural networks (Burtsev et al., 2020; De Jong et al., 2021; Févry et al., 2020), where external memory is directly stored into KV states.

## 2.3 Attention Mechanism

In a standard Softmax attention, we attend the query to all past KV states using the following formula:

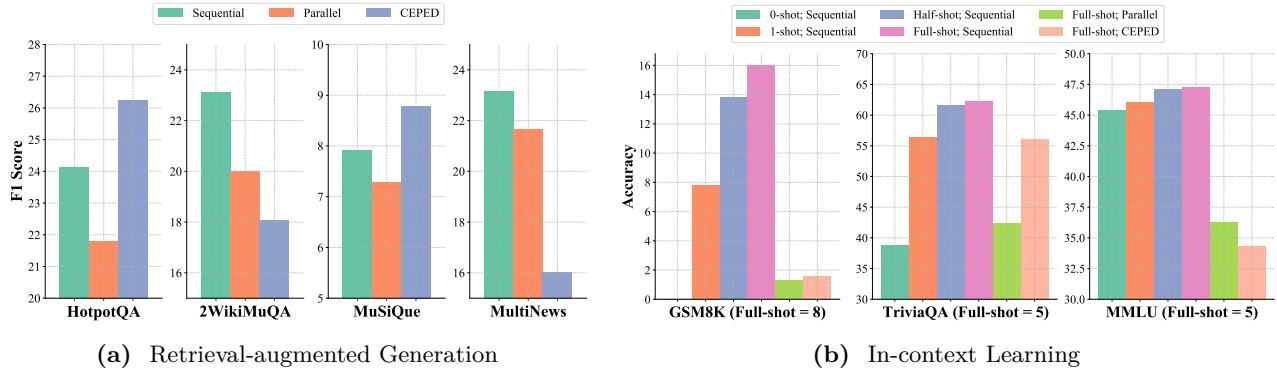
$$O = \text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \quad Q \in \mathbb{R}^{n \times d} \quad K, V \in \mathbb{R}^{m \times d}, \quad (2)$$

where  $Q$  is the query state, and  $K$  and  $V$  denote the key and value states, respectively. Previous research has revealed several significant insights into the distribution of attention weights (i.e.,  $\text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right)$ ).

**Attention Sink.** StreamingLLM (Xiao et al., 2023) identifies the presence of an “attention sink” in LLMs, a token that receives a significantly higher attention score than other tokens but provides limited semantic information. It observes that the attention sink exists in the initial token and influences the following tokens.

**Position Embedding.** To effectively process sequential input, LLMs require position embeddings, such as absolute position embeddings (Vaswani, 2017; Devlin, 2018) and relative position embeddings (Su et al., 2024; Press et al., 2021). However, the introduction of position embedding not only limits the context window to the training length (Chen et al., 2023) but also results in the “lost in the middle” (Liu et al., 2024a) issue, where LLMs struggle to produce correct answers when relevant information locates in the middle of the context.

## 3 Observations



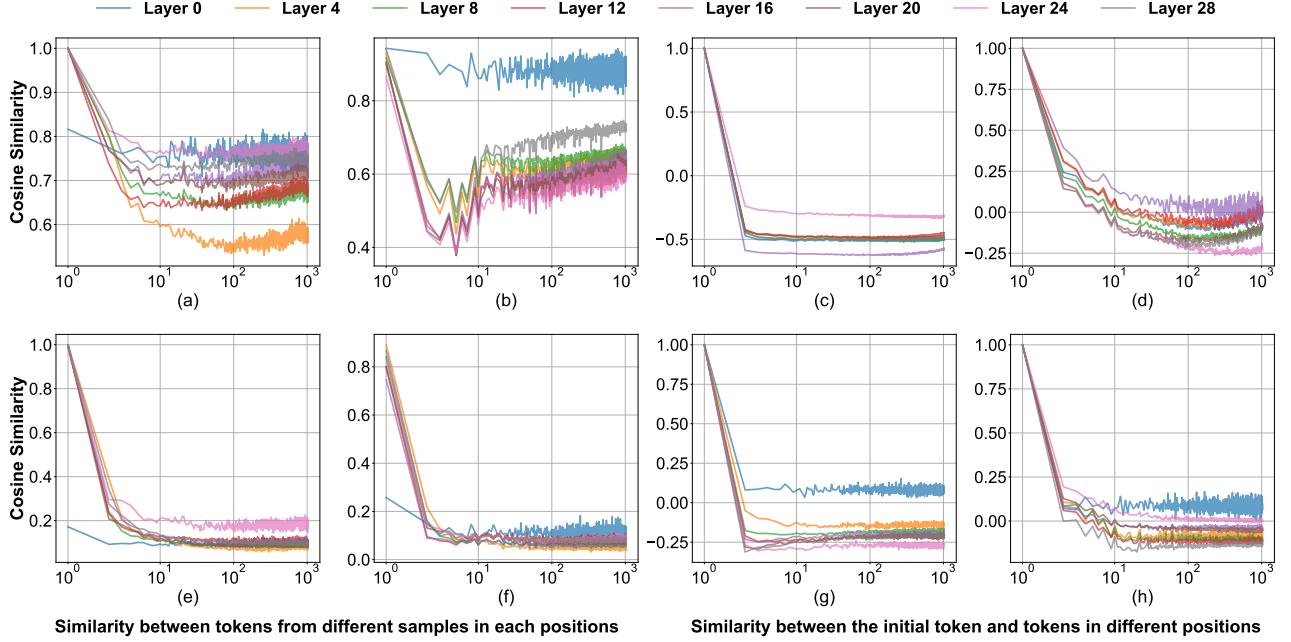
**Figure 2 Comparison of sequential encoding, parallel encoding, and CEPED in RAG and ICL scenarios.** Parallel encoding and CEPED degrades performance, especially on tasks such as GSM8K that requires reasoning ability.

In Section 3.1, we evaluate sequential encoding, parallel encoding, and CEPE-Distilled (CEPED) (Yen et al., 2024) using the LLAMA-2-7B-CHAT model<sup>1</sup>. Figure 2 presents our findings on various RAG and ICL tasks, highlighting the limitations of trainable approaches in generalizing to complex reasoning tasks. Next, we explore the alignments and misalignments between parallel encoding and sequential encoding in Section 3.2, providing insights into why parallel encoding remains effective and identifying opportunities for further improvement.

### 3.1 Trainable Approaches are only Effective for Easy Tasks.

In Figure 2, we compare the performance of different context encoding methods on RAG and ICL tasks, with detailed setups described in Appendix A. Our analysis of the long-context RAG capability on LongBench (Bai et al., 2023) is showcased in Figure 2a. Despite accessing more passages, CEPED only surpasses the sequential baseline in two of the three QA tasks, and it even notably underperforms parallel encoding in the summarization task (MultiNews), which requires synthesizing information from the entire context. We hypothesize that CEPED cannot process complex tasks since the encoder and decoder are only trained on the unlabeled pre-training corpus without instruction-tuning on high-quality QA samples. This conclusion is further supported by the results of ICL tasks (see Figure 2b), where CEPED performs on par with the 1-shot sequential encoding baseline

<sup>1</sup>We use the LLAMA-2 model for CEPED, as it is the only supported model. For other analyses, we employ LLAMA-3.



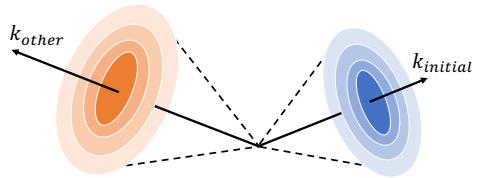
**Figure 3 Top Left:** Both LLAMA-3-8B-INSTRUCT (a) and MISTRAL-7B-INSTRUCT-v0.3 (b) exhibit a cosine similarity larger than 0.9 for the key states from distinct initial tokens. **Top Right:** Initial token’s key states show similar negative values to those from other positions for LLAMA-3-8B-INSTRUCT (c) and MISTRAL-7B-INSTRUCT-v0.3 (d) models. **Bottom:** Value states exhibit patterns similar to those observed in key states. The X-axis shows the positions of key and value states on a logarithmic scale. Visualizations and analyses for more base models are provided in Appendix B.

on TriviaQA but falls short of it on GSM8K and MMLU, despite using much more examples. The latter involves reasoning steps that are hard for the ill-trained model to understand. In conclusion, fine-tuning models to improve parallel encoding requires (i) more diverse and labeled data and (ii) resource-intensive instruction-tuning (e.g., SFT or RLHF ([Ouyang et al., 2022](#))). Given this unfavorable trade-off between training costs and model capabilities, we propose developing a training-free method to improve the performance of parallel encoding.

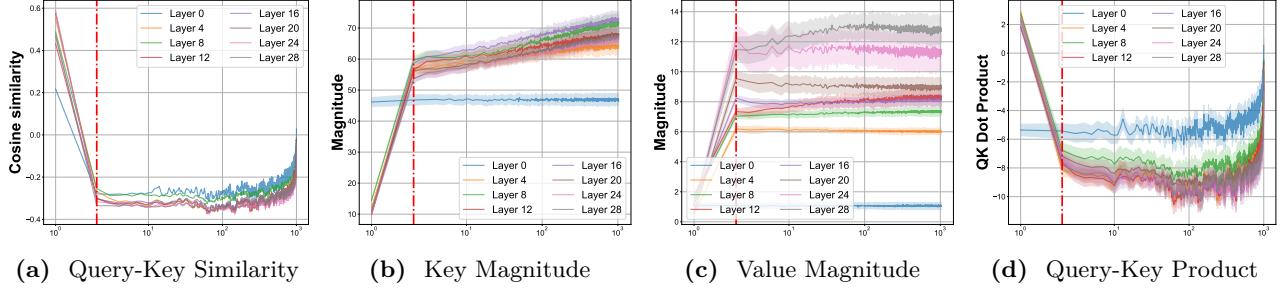
### 3.2 Comparing Parallel Encoding and Sequential Encoding.

In Figure 2, we observe that parallel encoding still holds promise, as it can generate reasonable responses without further modifications. This finding is non-trivial as contexts are encoded into KV states separately without guarantee that these states can be compared or combined. However, our analysis reveals that the attention mechanism naturally builds alignments between KV states from different positions in independent contexts similar to sequential encoding. To clarify this, Figure 3 focuses on the impact of the attention sink ([Xiao et al., 2023](#)), where we visualize the direction of KV states for different samples and positions. In Figure 4, we further visualize the distribution of various components in the Softmax attention, resulting in several findings.

**Key states from different contexts are similar.** In Figure 3a and 3b, we measure the cosine similarity between the key states of different initial tokens for the LLAMA-3-8B-INSTRUCT and MISTRAL-7B-INSTRUCT-v0.3 models, which consistently yields a value close to 1. This observation indicates that the direction of the initial key state remains invariant mainly across different inputs. Figure 3c and 3d further analyze the similarity between the initial key states and their subsequent states, where we observe comparable negative values from different positions. Therefore, the angles between the initial key states and their subsequent states are similar and significantly larger than the angles between different initial key states, as demonstrated in Figure 5. It suggests that the direction of key states remains relatively consistent across contexts, as they are primarily decided by the initial key states, which exhibit similar directions across examples. These findings, combined with the small variance in key state magnitudes across examples in Figure 4b, indicate



**Figure 5 Geometry of Key States.**



**Figure 4 Visualization of Different Components in Attention.** (a) The cosine similarity between query and key states increases as the distance between their positions decreases. (b) The magnitudes of key states show a slowly upward trend as position increases. (c) The magnitude of value states remain constant across positions. (d) Query-key dot products keep consistently low values except at initial and recent positions. A red dashed line marks the anomalous region for the first two tokens in all figures. The X-axis shows positions of KV states on a log scale. Results are measured with the LLAMA-3-8B-INSTRUCT model. Visualizations and analyses for more base models are provided in Appendix B.

that key states from different contexts share similar directions and magnitudes, making them comparable.

To further understand this, we experiment on HotPotQA using the LLAMA-3-8B-INSTRUCT model. Our analysis involves applying rotations of varying degrees around random axes to the initial key states. For parallel encoding, we explore two rotation modes: one using the same rotation axis for all contexts and another employing a random rotation axis for each context. Figure 6 reveals that sequential encoding keeps performance across various rotation degrees. In contrast, both modes in parallel encoding deteriorate when rotations exceed 150 degrees. This effect arises from the duplication of initial key states, intensifying our rotations' impact. Notably, using separate axes for each context leads to an earlier breakdown beginning at 90 degrees. This mode disrupts the directional similarity of key states with different initial tokens (i.e.,  $k_{\text{initial}}$ ) in Figure 5 and enlarges the angle between key states from different contexts.

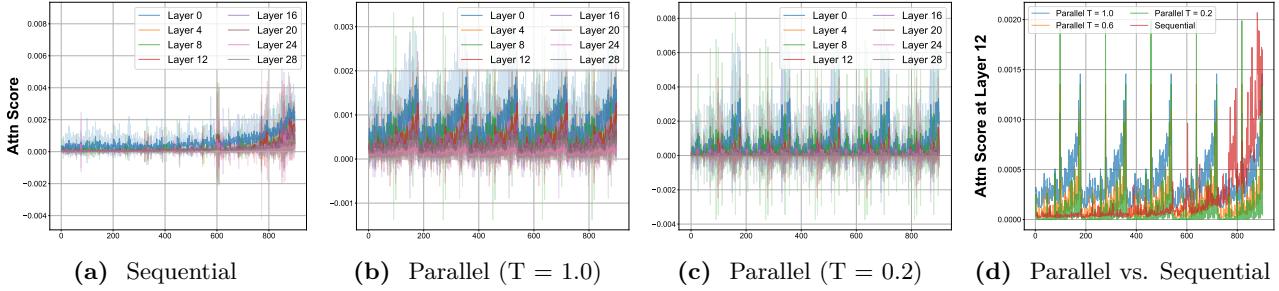
**Values states from different contexts can be combined.** In Equation (2), all value states are combined through a weighted summation, where the Softmax operator would normalize the weights of all value states to sum to 1. This normalization indicates that the magnitude of current value states is determined solely by those from previous positions, resulting in a similar  $L^2$  norm across positions, as shown in Figure 4c. Additionally, the small variance shows that the magnitudes are comparable among samples. This finding, coupled with a similar direction across samples and positions in Figure 3 (Bottom), indicates the possibility of combining value states.

**Opportunities for improvement.** Despite the KV states exhibiting similarity across contexts for most positions, the residual misalignments in Figure 4 still severely reduce accuracy. We summarize them as follows:

- In Figure 4, we observe a notable discrepancy in direction and magnitude for the initial positions, leading to large QK dot products at these positions in Figure 4d. They are identified as an anomaly in the context.
- Figure 4d shows the dot products between the query state and all past key states, revealing a notable increase when the states are positioned close to each other, as reflected in the larger similarity observed in Figure 4a.

## 4 Adaptive Parallel Encoding

With all the lessons learned in Section 3, we will design our **APE** to address the residual misalignments. APE enables a seamless shift to parallel encoding without requiring training while maintaining most of the model's capabilities. Our approach adaptively aligns the distribution of attention weights between sequential and parallel encoding via three steps as illustrated in Figure 1, thereby boosting efficiency and performance.



**Figure 7 Comparison of Attention Weight Distribution within Contexts.** (a) Sequential encoding allocates high attention scores to neighboring tokens. (b) Parallel encoding distributes attention scores more uniform across neighboring tokens from all contexts. (c) Adjusting the temperature  $T$  sparsifies the distribution. (d) After adjustment, the distribution in parallel encoding becomes similar to sequential encoding. The X-axis represents token positions.

#### 4.1 Prepending Shared Prefix.

Figure 4 shows that the distribution of the first few tokens differs significantly from that of subsequent tokens. This discrepancy poses a challenge when encoding contexts in parallel due to duplicating these abnormal KV states. To address this issue, we propose a simple yet effective solution: prepending a shared prefix to all contexts. This approach ensures that these KV states appear only once in each generation step. In practice, the choice of prefix varies with the model and task. We use existing system prompts and instructions as the shared prefix when available. Otherwise, we will insert a few newline characters (i.e., “\n”) before all contexts.

Although the later positions are not identified as abnormal, we still observe instability at the start of LLM inputs. To mitigate this issue, we may also consider extending the existing prefix with more newline characters.

#### 4.2 Adjusting Attention Temperature.

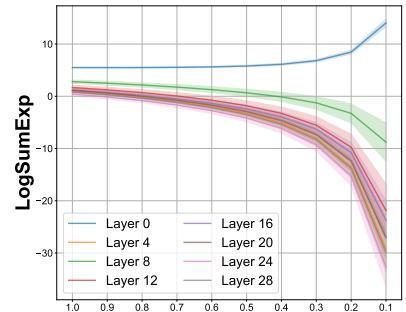
In Figure 4d, the value of QK dot products increases as the relative distance decreases, with a notably sharper rise when the distance approaches zero. To show its impact on parallel encoding, we set a 50-token prefix and query, encoding the remaining 900 tokens either sequentially or in five parallel chunks, with attention distributions shown in Figure 7. Comparing Figure 7b with 7a, duplicating neighboring KV states in parallel encoding will disperse the query’s attention to multiple contexts, resulting in a more uniform attention distribution. We adjust the attention temperature  $T$  to a value less than 1 to refocus on the most relevant tokens, sharpening the distribution after the Softmax operation. The comparison between different  $T$  is shown in Figure 7c and 7d.

#### 4.3 Adding Scaling Factor.

While adjusting the temperature sharpens the attention distribution among context tokens, it will also alter the overall attention allocated to the whole context, as indicated by the LogSumExp value in Figure 8. Specifically, when the sum of the original QK dot product values in a given layer is significantly greater than 0, reducing temperature amplifies these positive values, resulting in an increased, positive LogSumExp value. Conversely, when the sum is closer to 0, lowering temperature has a stronger effect on the negative QK dot products, leading to a decreased, negative LogSumExp value. These effects generally increase the absolute value of LogSumExp(QK). To compensate for these changes, we introduce a scaling factor  $S < 1$  to reduce this absolute value.

#### 4.4 Formulation.

Given these three steps, we can formulate the modified attention in APE. We begin with the standard Softmax attention, where  $Q$ ,  $K$ , and  $V$  are the query, key, and value states, respectively. We use the subscript  $C_i$  for elements from the context  $C_i$ , while those without a subscript correspond to user queries or generated texts.



**Figure 8 Parallel w/ Different  $T$ .**

$$O = \text{Softmax} \left( \frac{Q[K_{C_1}^\top, \dots, K_{C_N}^\top, K^\top]}{\sqrt{d}} \right) \times [V_{C_1}, \dots, V_{C_N}, V] \quad (3)$$

$$= \frac{[A_{C_1}, \dots, A_{C_N}, A]}{\sum_{i=1}^N \sum_{j=1}^{l_{C_i}} a_{C_i,j} + \sum_{j=1}^l a_j} \times [V_{C_1}, \dots, V_{C_N}, V], \quad (4)$$

where  $A_{C_i} = [\exp \frac{Qk_{C_i,1}^\top}{\sqrt{d}}, \dots, \exp \frac{Qk_{C_i,l_{C_i}}^\top}{\sqrt{d}}]$  and  $a_{C_i,j} = \exp \frac{Qk_{C_i,j}^\top}{\sqrt{d}}$ . Similar for  $A$  and  $a_j$ .

After incorporating the proposed changes, the formula for our refined attention calculation becomes:

$$O' = \frac{[A_P, A'_{C_1}, \dots, A'_{C_N}, A]}{\sum_{j=1}^{l_P} a_{P,j} + (\sum_{i=1}^N \sum_{j=1}^{l_{C_i}} a'_{C_i,j})^S + \sum_{j=1}^l a_j} \times [V_P, V_{C_1}, \dots, V_{C_N}, V], \quad (5)$$

where  $A'_{C_i} = [\exp \frac{Qk_{C_i,1}^\top}{T\sqrt{d}}, \dots, \exp \frac{Qk_{C_i,l_{C_i}}^\top}{T\sqrt{d}}] \cdot (\sum_{i=1}^N \sum_{j=1}^{l_{C_i}} a'_{C_i,j})^{S-1}$  and  $a'_{C_i,j} = \exp \frac{Qk_{C_i,j}^\top}{T\sqrt{d}}$ .

$A_P$  represents the attention weights for the shared prefix while  $A$  denotes that for query and generated tokens. The attention temperature  $T$  and the scaling factor  $S$  for the context are less than 1. Appendix C provides a detailed deduction of this formula for better understanding. All these modifications are compatible with fast attention implementations such as flash attention (Dao et al., 2022) by computing the context and non-context KV states separately and merging them into the attention output. This process only incur a negligible overhead.

For the choice of hyperparameters, we conduct a greedy search over a small validation set. If no prefix is provided, we begin by adding two “\n” and increase the prefix length by 10, 20, and 40.  $S$  and  $T$  are searched in the ranges [0.1, 1.0] using 0.1 step sizes. We use  $S \cdot T$  instead of  $S$  as the scaling factor to simplify our search.

## 5 Experiments

Empirically, we present the effectiveness and efficiency of APE in CAG scenarios such as RAG and ICL. Since we focus on context encoding problems, we do not include comparisons with long-context LLMs. Specifically,

- In Section 5.1, APE can maintain 98% of the accuracy on ChatRAG-Bench compared to sequential encoding. Furthermore, it improves 3.3% performance for RAG on LongBench by retrieving more and longer contexts.
- In Section 5.2, APE outperforms parallel encoding by 7.9% on average in three ICL tasks. Moreover, APE can maintain 93% of the accuracy achieved by sequential encoding when using the same number of examples.
- In Section 5.3, APE can scale to many-shot CAG tasks, effectively encoding hundreds of texts in parallel.
- In Section 5.4, APE achieves 4.5× faster inference for 128k context through 28× reduction in prefilling time.

### 5.1 Retrieval-Augmented Generation.

In the context of RAG tasks, we validate that APE retains most of the sequential encoding capability while accommodating more and longer contexts, mitigating retrieval errors, and outperforming encoding baselines.

#### 5.1.1 Retrieval for Multi-turn Question Answering.

**Setup.** APE is evaluated on five conversational QA tasks using ChatRAGBench (Liu et al., 2024b). For each query, we prepare about 100 text chunks. Three retrievers of varying quality are employed to retrieve up to the top-5 chunks for evaluation, including Contriever (Izacard et al., 2021), GTE-base Li et al. (2023), and Dragon-multiturn Liu et al. (2024b). We use LLAMA3-CHATQA-1.5-8B as the base model. To fairly measure performance drop after our modifications, the same retrieved texts are used for APE and sequential encoding.

**Table 1** Comparison between APE and sequential encoding using three retrievers on ChatRAG-Bench.

| Method                       | INSCIT       | Doc2Dial | TopicCQA | Qrecc        | QuAC         | Average      |
|------------------------------|--------------|----------|----------|--------------|--------------|--------------|
| Contriever, Sequential       | 19.97        | 23.85    | 30.49    | 46.75        | 26.57        | 29.53        |
| Contriever, APE              | 19.88        | 23.28    | 28.84    | 46.28        | 26.80        | 29.02        |
| $\Delta$                     | -0.09        | -0.57    | -1.65    | -0.47        | +0.23        | -0.51        |
| GTE-base, Sequential         | 21.58        | 32.35    | 33.41    | 46.54        | 30.69        | 32.91        |
| GTE-base, APE                | 20.85        | 30.99    | 31.92    | 45.83        | 30.35        | 31.99        |
| $\Delta$                     | -0.73        | -1.36    | -1.49    | -0.71        | -0.34        | -0.92        |
| Dragon-multiturn, Sequential | 25.42        | 36.27    | 36.10    | 49.01        | 35.12        | 36.38        |
| Dragon-multiturn, APE        | 23.84        | 34.93    | 33.80    | 48.70        | 34.92        | 35.24        |
| $\Delta$                     | -1.58        | -1.34    | -2.30    | -0.31        | -0.20        | -1.14        |
| All texts, APE               | <b>27.22</b> | 36.13    | 35.72    | <b>49.15</b> | <b>35.70</b> | <b>36.78</b> |

**Results.** Table 1 shows that switching from sequential encoding to APE results in performance drops of 0.51%, 0.92%, and 1.14% across different retrievers, respectively. While this drop increases with retriever quality, APE still keeps 97% of the sequential encoding performance for the best retriever. By increasing the text chunk length for 5 times, APE directly inputs all texts without any retrieval process, achieving superior performance.

### 5.1.2 Retrieval for Long-context Understanding.

**Setup.** Our evaluation involves eight tasks on LongBench (Bai et al., 2023). Given the long context, we split it into chunks with a size of  $M$  words, employ Contriever (Izacard et al., 2021) to compute the embeddings of all chunks and the query and retrieve the top- $N$  chunks according to the cosine similarity of their embeddings to the query embedding.  $M$  and  $N$  vary across different methods. We compare APE with sequential encoding with and without RAG, and PCW, using LLAMA-3-8B-INSTRUCT (Dubey et al., 2024), MISTRAL-7B-INSTRUCT-v0.3 (Jiang et al., 2023), GEMMA-2-9B-IT (Team et al., 2024), and LLAMA-3.1-8B-INSTRUCT as base models.

**Results.** In Table 2, APE consistently improves performance across all models, achieving a 5.6% average gain over sequential encoding without RAG. It also outperforms sequential RAG baselines by 3.3% by retrieving more and longer contexts. The superior performance over PCW further showcases the effectiveness of our modifications in APE. Notably, APE surpasses the 128K-context variant of the LLAMA-3.1-8B-INSTRUCT model by placing retrieved texts within the 8K context window, mitigating the “lost in the middle” phenomenon.

## 5.2 In-context Learning

**Setup.** We evaluate APE on three ICL tasks using the LM Evaluation Harness (Gao et al., 2024) codebase: GSM8K (8-shot) (Cobbe et al., 2021a), TriviaQA (5-shot) (Joshi et al., 2017), and MMLU (5-shot) (Hendrycks et al., 2020a). Experiments are conducted using the same base models as in our LongBench evaluations. We compare parallel encoding (PCW) to show the improvement of APE. Sequential encoding with varying numbers of shots (i.e., 1-shot, half-shots, and full-shots) is also employed to measure the gap from the ideal scenarios.

**Results.** In Figure 9, APE surpasses parallel encoding with average improvements of 15.4% on GSM8K, 4.7% on TriviaQA, and 3.5% on MMLU. When compared with the 1-shot sequential baseline with similar context length, our method consistently yields superior results. Moreover, APE performs better than half-shot sequential encoding in 8/12 settings and preserves 93% accuracy compared to full-shot sequential encoding. Additionally, the LLAMA family exhibits enhanced compatibility with parallel encoding, potentially due to the stronger directional alignment of initial tokens from different contexts (see Figure 3a). Across different tasks, the performance gap between APE and full-shot sequential encoding is the largest on GSM8K. This finding suggests that while APE keeps most capabilities, its effectiveness may decrease as task complexity increases.

**Table 2** Comparison between APE and baselines on LongBench across different models using RAG. C denotes Contriever, and  $M \times N$  indicates retrieval of the top- $N$  chunks, each containing  $M$  words.

| Model                          | MuSiQue      | Qasper       | 2WikiMQA     | DuRead       | HotpotQA     | NarratQA     | MFQA_zh      | MFQA_en      | Avg.         |
|--------------------------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| LLAMA-3-8B-INSTRUCT            | 20.70        | 41.05        | 30.02        | 9.55         | 45.90        | 20.98        | <b>58.54</b> | 45.04        | 33.97        |
| C $200 \times 20$ , Sequential | <b>27.93</b> | <b>42.71</b> | 38.35        | 12.65        | 49.60        | 22.78        | 57.82        | <b>48.94</b> | 37.60        |
| C $4000 \times 20$ , PCW       | 18.82        | 42.59        | 40.99        | 21.57        | 47.09        | 23.29        | 54.40        | 45.05        | 36.73        |
| C $4000 \times 20$ , APE       | 26.19        | 42.32        | <b>44.43</b> | <b>23.13</b> | <b>49.71</b> | <b>30.71</b> | 55.03        | 45.41        | <b>39.62</b> |
| MISTRAL-7B-INSTRUCT-v0.3       | 10.05        | 31.08        | 22.12        | 17.68        | 32.09        | 19.68        | 32.03        | 40.38        | 25.64        |
| C $200 \times 20$ , Sequential | 11.58        | 21.98        | 24.44        | 20.80        | 32.79        | 16.06        | 34.43        | 38.40        | 25.06        |
| C $4000 \times 20$ , PCW       | 17.58        | 35.57        | 32.97        | 18.70        | 37.05        | 14.10        | 34.69        | 40.14        | 28.85        |
| C $4000 \times 20$ , APE       | <b>20.30</b> | <b>36.81</b> | <b>34.37</b> | <b>21.89</b> | <b>42.33</b> | <b>20.49</b> | <b>40.20</b> | <b>44.03</b> | <b>32.55</b> |
| GEMMA-2-9B-IT                  | 22.57        | 39.99        | 48.06        | 27.40        | 47.49        | 23.11        | 50.81        | 45.35        | 38.10        |
| C $200 \times 10$ , Sequential | 30.69        | 42.86        | <b>53.55</b> | 28.04        | 52.05        | 24.45        | 50.25        | 48.34        | 41.28        |
| C $2000 \times 20$ , PCW       | 26.27        | 46.69        | 47.59        | 23.43        | 48.95        | 27.11        | <b>56.69</b> | 49.81        | 40.82        |
| C $2000 \times 20$ , APE       | <b>33.38</b> | <b>47.72</b> | 49.49        | <b>28.43</b> | <b>56.62</b> | <b>30.41</b> | 56.52        | <b>50.84</b> | <b>44.18</b> |
| LLAMA-3.1-8B-INSTRUCT          | 22.18        | <b>46.81</b> | 40.58        | <b>34.61</b> | 43.97        | 23.08        | 61.60        | 51.89        | 38.98        |
| 128K, Sequential               | 28.35        | 47.20        | 40.81        | 33.34        | 53.46        | 30.57        | 61.97        | 53.25        | 42.24        |
| C $200 \times 20$ , Sequential | <b>30.62</b> | 42.33        | 44.39        | 33.51        | 49.97        | 23.87        | 56.87        | <b>55.14</b> | 40.22        |
| C $4000 \times 20$ , PCW       | 21.23        | 41.52        | 44.87        | 31.11        | 49.47        | 19.98        | 60.90        | 51.19        | 38.44        |
| C $4000 \times 20$ , APE       | 26.88        | 43.03        | <b>50.11</b> | 32.10        | <b>55.41</b> | <b>30.50</b> | <b>62.02</b> | 52.51        | <b>42.86</b> |

**Table 3** Comparison between APE and sequential encoding in various many-shot RAG and ICL tasks.

| Method                | Retrieval-augmented Generation |              |              |              | In-context Learning |              |             |              |
|-----------------------|--------------------------------|--------------|--------------|--------------|---------------------|--------------|-------------|--------------|
|                       | ArguAna                        | FEVER        | NQ           | SciFact      | Date                | Salient      | Tracking7   | Web          |
| Sequential, Zero-shot | 11.15                          | 7.78         | 17.78        | 7.74         | 20.00               | 8.89         | 1.12        | 8.89         |
| Sequential, Few-shot  | 11.20                          | 9.78         | 17.81        | 9.49         | 36.64               | 38.89        | 6.67        | 38.89        |
| Sequential, Half-shot | 15.34                          | 13.12        | 19.64        | 16.12        | 45.55               | 42.22        | 8.89        | 55.56        |
| Sequential, Full-shot | 12.84                          | 14.19        | <b>24.54</b> | <b>16.88</b> | <b>46.67</b>        | <b>46.67</b> | <b>8.89</b> | <b>58.89</b> |
| APE, Full-shot        | <b>16.32</b>                   | <b>14.70</b> | 21.91        | 15.72        | 43.33               | 45.55        | <b>8.89</b> | <b>58.89</b> |

### 5.3 Many-shot Context-Augmented Generation

**Setup.** We evaluate the scalability of APE on four RAG and ICL tasks from the LOFT benchmark (Lee et al., 2024), each involving hundreds of additional texts. We employ LLAMA-3.1-8B-INSTRUCT as our base model to compare APE with sequential encoding, both applied to the same many-shot inputs. The total context lengths for the RAG and ICL tasks are 128K and 32K, respectively. We also include the zero-shot, few-shot ( $\leq 5$ ), and half-shot sequential encoding baselines. For metrics, F1 score and EM are used in RAG and ICL tasks.

**Results.** In Table 3, APE achieves performance comparable to sequential encoding when processing the same many-shot long-context inputs, showing its ability to encode hundreds of texts in parallel efficiently. Notably, it outperforms sequential encoding on ArguAna and FEVER for RAG tasks. While APE is expected to reduce performance, it recovers this drop by positioning all texts close to the query, mitigating the “lost in the middle” problem in long-context LLMs. For ICL tasks, APE can learn from examples as effective as sequential encoding.

### 5.4 Efficiency Evaluation

**Setup.** We measure the latency for sequential encoding, MInference (Jiang et al., 2024a), and APE using Llama-3.1-8B-Instruct (Dubey et al., 2024) on an H100 GPU with batch sizes of 1 and 4. The query and generation lengths are fixed at 256 tokens, while the context lengths range from 2K to 128K tokens. We employ VLLM (Kwon et al., 2023) as our inference engine and measure both prefilling time and total inference time.

**Results.** Comparing to sequential encoding and MInference, APE can accelerate inference up to  $4.5\times$  and

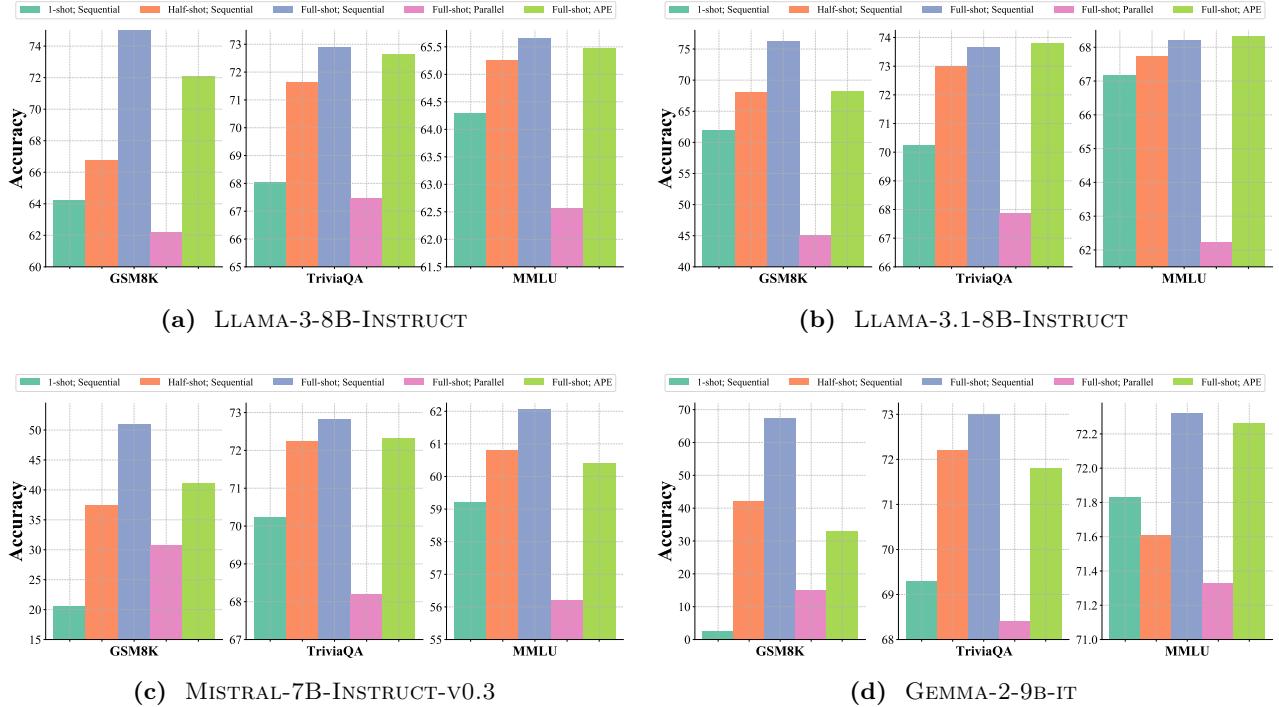


Figure 9 Performance comparison of APE, parallel encoding, and sequential encoding on ICL tasks.

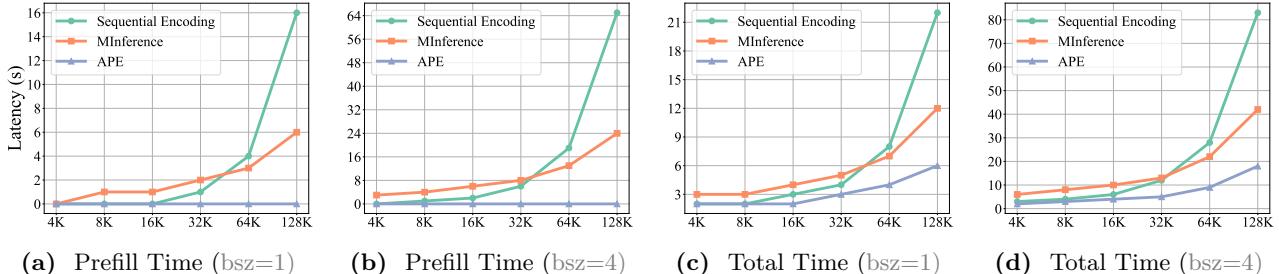


Figure 10 Latency on H100 GPU: prefill and total inference time (s). The gray text in brackets is batch size.

2.2 $\times$  respectively for long-context scenarios in Figure 10. For 128K-token contexts, APE reduces prefilling time by 28 $\times$  compared to MiInference. The prefilling cost of APE exhibits linear scaling and consumes less than 10% of inference time, whereas baselines require over 50% as context length increases. APE also shows superior versatility, while MiInference slows inference with additional overhead for short contexts and large batches.

## 6 Analysis

This section presents analyses to answer the following research questions: **RQ1**: Can APE improve performance for real-world RAG applications? **RQ2**: How does each component in APE contribute to the performance? **RQ3**: Can APE extend LLM context window size in long-context scenarios without RAG?

### 6.1 Can APE improve performance for real-world RAG applications?

In Table 4, we evaluate APE in real-world RAG scenarios using the CRAG benchmark (Yang et al., 2024). Task 1 augments the model with five webpages, while Task 2 provides an additional knowledge graph as another retrieval source. In our experiments, the sequential encoding baseline is limited to retrieving 4K tokens, whereas APE can process 20 parallel segments of 4K tokens each. By incorporating significantly more external texts,

APE consistently outperforms sequential encoding with limited context sizes while reducing latency. Moreover, the improvement in Task 2 shows the effectiveness of APE in merging text from multiple sources.

**Table 4 Performance and latency comparison using the Llama-3-8B-Instruct model on CRAG benchmark.**

| Task     | Model               | Latency (ms) | Accuracy (%) | Hallucination | Missing      | Score <sub>a</sub> |
|----------|---------------------|--------------|--------------|---------------|--------------|--------------------|
| LLM only | LLAMA-3-8B-INSTRUCT | 682          | 22.14        | 48.97         | 28.90        | -26.83             |
| Task 1   | LLAMA-3-8B-INSTRUCT | 1140         | 23.28        | 29.49         | 47.22        | -6.21              |
|          | +APE                | 1054         | <b>25.53</b> | <b>21.30</b>  | <b>37.93</b> | <b>-0.41</b>       |
| Task 2   | LLAMA-3-8B-INSTRUCT | 1830         | 24.46        | 28.38         | 47.15        | -3.92              |
|          | +APE                | 1672         | <b>27.04</b> | <b>18.74</b>  | <b>37.32</b> | <b>2.16</b>        |

## 6.2 How does each component in APE contribute to the performance?

In Table 5, we conduct an ablation study to examine each component in APE, including the shared prefix ( $P$ ), attention temperature ( $T$ ), and scaling factor ( $S$ ). We present results averaged across the four base models evaluated in Figure 9. Our findings indicate that incorporating each of these components can improve performance for all tasks, with average improvements of 5.19%, 0.59%, and 2.07%, respectively. Among them, adding a shared prefix leads to the largest improvement, while adjusting the attention temperature yields minimal accuracy gains without the complementary effect of the scaling factor.

**Table 5 Ablation study of APE components on ICL tasks.**  $P$ : shared prefix,  $T$ : attention temperature,  $S$ : scaling factor.

| $P$ | $T$ | $S$ | GSM8K         | TriviaQA      | MMLU          |
|-----|-----|-----|---------------|---------------|---------------|
|     |     |     | 38.25%        | 67.99%        | 63.09%        |
| ✓   |     |     | 50.42%        | 70.76%        | 63.70%        |
| ✓   | ✓   |     | 51.15%        | 71.03%        | 64.49%        |
| ✓   | ✓   | ✓   | <b>53.62%</b> | <b>72.64%</b> | <b>66.62%</b> |

## 6.3 Can APE extend context lengths in long-context scenarios without RAG?

Table 6 evaluates the effectiveness of APE in handling a single long-context input using the LLAMA-3-8B-INSTRUCT model on the LongBench dataset (Bai et al., 2023). To accommodate the long context within our APE, we split it into multiple segments of less than 7,500 tokens. Additionally, we append the last 500 tokens to the query for two code completion tasks. Our results indicate that APE enhances performance across 10/11 tasks, yielding an average improvement of 6.6% compared to the sequential encoding baseline with limited context window size. More baseline results of long-context LLM approaches are provided in Appendix D.

**Table 6 Performance comparison across different long-context tasks on LongBench (Bai et al., 2023).**

| Method              | NarratQA     | Qasper       | MultiFQA     | GovReport    | QMSum        | LCC          |
|---------------------|--------------|--------------|--------------|--------------|--------------|--------------|
| LLAMA-3-8B-INSTRUCT | 19.32        | 32.83        | 43.38        | 27.89        | 22.40        | 53.22        |
| +APE                | <b>26.87</b> | <b>39.14</b> | <b>59.12</b> | <b>29.10</b> | <b>23.08</b> | <b>66.09</b> |
| Method              | RepoBench-P  | HotpotQA     | 2WikiMQA     | MuSiQue      | MultiNews    | Average      |
| LLAMA-3-8B-INSTRUCT | 38.15        | 44.24        | 21.01        | 20.47        | <b>23.63</b> | 31.50        |
| +APE                | <b>49.43</b> | <b>50.11</b> | <b>28.06</b> | <b>25.79</b> | 22.40        | <b>38.11</b> |

## 7 Conclusion

This work explores the potential of parallel encoding in CAG scenarios, which can pre-cache KV states for fast inference and re-use positions for long context but lead to worse performance. To address this, we propose APE, a training-free method to enable accurate, fast, and long CAG systems. APE achieves this by aligning the attention weight distribution of parallel encoding with sequential encoding via three steps: shared prefix, adaptive temperature, and scaling factor. Empirically, we show that APE improves accuracy and efficiency in various RAG and ICL tasks while successfully scaling to process hundreds of chunks in parallel for both settings.

## 8 Limitations

While APE shows the effectiveness and efficiency of parallel encoding with only inference-time modification in the attention distribution, it remains sensitive to hyperparameter selection, particularly the attention temperature  $T$  and scaling factor  $S$ . In real-world applications, where contexts vary in length, quantity, and content, aligning the distribution between sequential and parallel encoding automatically presents a significant challenge.

## 9 Acknowledgement

This work is supported in part by NSF award CNS-2211882 and a gift from Qualcomm. We thank the authors of ChatQA (Liu et al., 2024b), Longbench (Bai et al., 2023), CRAG (Yang et al., 2024), LM Evaluation Harness (Gao et al., 2024), VLLM (Kwon et al., 2023), and MInference (Jiang et al., 2024a) for their useful codebase, benchmark, and models, and Yixin Dong, Hanshi Sun, Zhuoming Chen for their helpful discussions.

## References

- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- Rishabh Agarwal, Avi Singh, Lei M Zhang, Bernd Bohnet, Stephanie Chan, Ankesh Anand, Zaheer Abbas, Azade Nova, John D Co-Reyes, Eric Chu, et al. Many-shot in-context learning. *arXiv preprint arXiv:2404.11018*, 2024.
- Akari Asai, Zexuan Zhong, Danqi Chen, Pang Wei Koh, Luke Zettlemoyer, Hannaneh Hajishirzi, and Wen-tau Yih. Reliable, adaptable, and attributable language models with retrieval. *arXiv preprint arXiv:2403.03187*, 2024.
- Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.
- Mikhail S Burtsev, Yuri Kuratov, Anton Peganov, and Grigory V Sapunov. Memory transformer. *arXiv preprint arXiv:2006.11527*, 2020.
- Harrison Chase. Longchain, 2022. <https://github.com/langchain-ai/langchain>.
- Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*, 2023.
- 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*, 2021a.
- 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*, 2021b.
- Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.
- Michiel De Jong, Yury Zemlyanskiy, Nicholas FitzGerald, Fei Sha, and William Cohen. Mention memory: incorporating textual knowledge into transformers through entity mention attention. *arXiv preprint arXiv:2110.06176*, 2021.
- Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.
- Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvassy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024.
- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- Alexander R Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir R Radev. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. *arXiv preprint arXiv:1906.01749*, 2019.
- Thibault Févry, Livio Baldini Soares, Nicholas FitzGerald, Eunsol Choi, and Tom Kwiatkowski. Entities as experts: Sparse memory access with entity supervision. *arXiv preprint arXiv:2004.07202*, 2020.
- Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 07 2024. <https://zenodo.org/records/12608602>.
- Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.
- AI Gradient. Llama-3-8b-instruct-262k, 2024.

- Aman Gupta, Anup Shirgaonkar, Angels de Luis Balaguer, Bruno Silva, Daniel Holstein, Dawei Li, Jennifer Marsman, Leonardo O Nunes, Mahsa Rouzbahman, Morris Sharp, et al. Rag vs fine-tuning: Pipelines, tradeoffs, and a case study on agriculture. *arXiv preprint arXiv:2401.08406*, 2024.
- Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020a.
- Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020b.
- Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*, 2020.
- Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*, 2021.
- Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.
- Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *arXiv preprint arXiv:2407.02490*, 2024a.
- Ziyan Jiang, Xueguang Ma, and Wenhui Chen. Longrag: Enhancing retrieval-augmented generation with long-context llms. *arXiv preprint arXiv:2406.15319*, 2024b.
- Hongye Jin, Xiaotian Han, Jingfeng Yang, Zhimeng Jiang, Zirui Liu, Chia-Yuan Chang, Huiyuan Chen, and Xia Hu. Llm maybe longlm: Self-extend llm context window without tuning. *arXiv preprint arXiv:2401.01325*, 2024.
- Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.
- Jinhyuk Lee, Anthony Chen, Zhuyun Dai, Dheeru Dua, Devendra Singh Sachan, Michael Boratko, Yi Luan, Sébastien MR Arnold, Vincent Perot, Siddharth Dalmia, et al. Can long-context language models subsume retrieval, rag, sql, and more? *arXiv preprint arXiv:2406.13121*, 2024.
- Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*, 2023.
- Zhenyu Li, Yike Zhang, Tengyu Pan, Yutao Sun, Zhichao Duan, Junjie Fang, Rong Han, Zixuan Wang, and Jianyong Wang. Focusllm: Scaling llm’s context by parallel decoding. *arXiv preprint arXiv:2408.11745*, 2024.
- Jerry Liu. Llamaindex, 11 2022. [https://github.com/jerryjliu/llama\\_index](https://github.com/jerryjliu/llama_index).
- Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024a.
- Zihan Liu, Wei Ping, Rajarshi Roy, Peng Xu, Chankyu Lee, Mohammad Shoeybi, and Bryan Catanzaro. Chatqa: Surpassing gpt-4 on conversational qa and rag. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024b.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.
- Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, et al. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. *arXiv preprint arXiv:2403.12968*, 2024.
- Ofir Press, Noah A Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021.

- Nir Ratner, Yoav Levine, Yonatan Belinkov, Ori Ram, Inbal Magar, Omri Abend, Ehud Karpas, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. Parallel context windows for large language models. *arXiv preprint arXiv:2212.10947*, 2022.
- Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.
- East Sun, Yan Wang, and Lan Tian. Block-attention for efficient rag. *arXiv preprint arXiv:2409.15355*, 2024.
- Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.
- AI Together. Llama-2-7b-32k-instruct, 2023.
- Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.
- A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.
- Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.
- Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.
- Xiao Yang, Kai Sun, Hao Xin, Yushi Sun, Nikita Bhalla, Xiangsen Chen, Sajal Choudhary, Rongze Daniel Gui, Ziran Will Jiang, Ziyu Jiang, et al. Crag—comprehensive rag benchmark. *arXiv preprint arXiv:2406.04744*, 2024.
- Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.
- Howard Yen, Tianyu Gao, and Danqi Chen. Long-context language modeling with parallel context encoding. *arXiv preprint arXiv:2402.16617*, 2024.
- André Zayarni, Andrey Vasnetsov, et al. Qdrant, 2024. <https://qdrant.tech/>.

# Appendix

## A Detailed Experimental Setups for Section 3.1

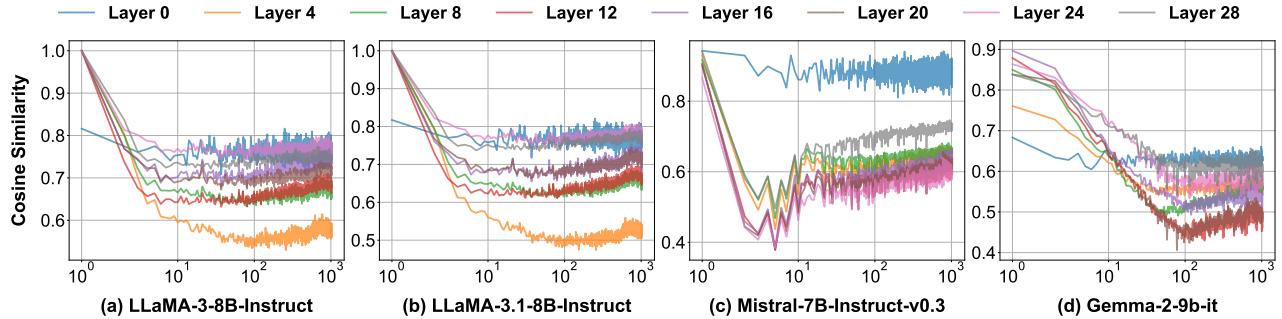
**RAG.** We select four tasks that require processing multiple input documents from the LongBench dataset (Bai et al., 2023), including HotpotQA (Yang et al., 2018), 2WikiMultihopQA (Ho et al., 2020), MuSiQue (Trivedi et al., 2022), and MultiNews (Fabbri et al., 2019). The F1 score is used as the evaluation metric for the three QA tasks, while Rouge-L is used for the summarization task. Both parallel encoding and CEPED process each document independently using  $\Theta_{E_{nc}}$ . For documents that exceed the length limitation of  $\Theta_{E_{nc}}$ , we split them into multiple chunks for encoding. In sequential encoding, we will truncate lengthy inputs from the middle.

**ICL.** We select three few-shot learning tasks from LM Evaluation Harness (Gao et al., 2024) to evaluate the ICL ability of different encoding methods, involving GSM8K (Cobbe et al., 2021b), TriviaQA Joshi et al. (2017), and MMLU (Hendrycks et al., 2020b). In parallel encoding and CEPED, we will encode each example separately and input all the resulting KV states to  $\Theta_{Dec}$ . For sequential encoding, we use variants with different numbers of shots to further measure the effectiveness of other methods, including 0-shot, 1-shot, half-shot, and full-shot.

## B More Visualization Results for Section 3.2

### B.1 Similarity between Tokens from Different Samples in Each Position for Key States.

In Figure 11, we showcase that key states in different layers maintain consistently high cosine similarity values for various initial tokens, with only the first layer exhibiting slightly lower similarities. Our analysis reveals that LLaMA-3-8B-INSTRUCT and LLaMA-3.1-8B-INSTRUCT exhibit almost the same direction (approximately 1.0) for different tokens beyond the first layer, while MISTRAL-7B-INSTRUCT-v0.3 and GEMMA-2-9B-IT show substantial but lower similarities ranging from 0.8 to 0.9. These findings indicate inherent alignments across contexts while highlighting the potential for further improvements through the shared prefix in Section 4.1.



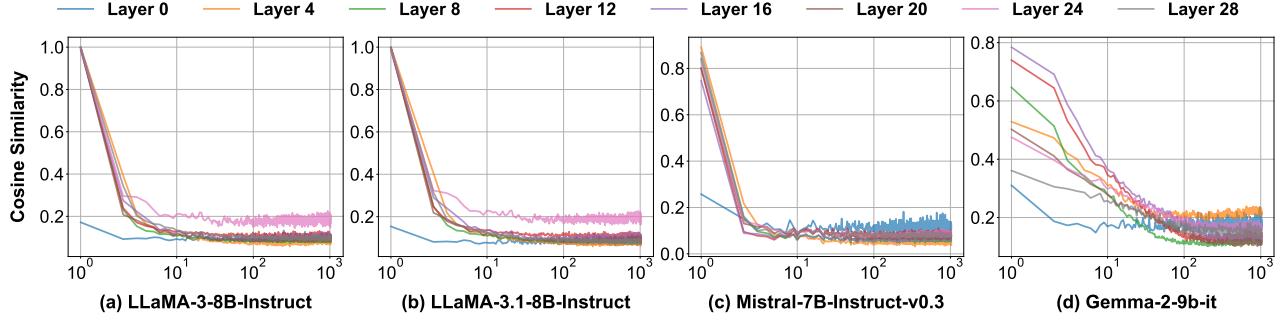
**Figure 11** For all base models, key states from distinct initial tokens exhibit a large cosine similarity than the following positions, where the LLaMA family even approaches 1. The X-axis shows positions of key states on a logarithmic scale.

### B.2 Similarity between Tokens from Different Samples in Each Position for Value States.

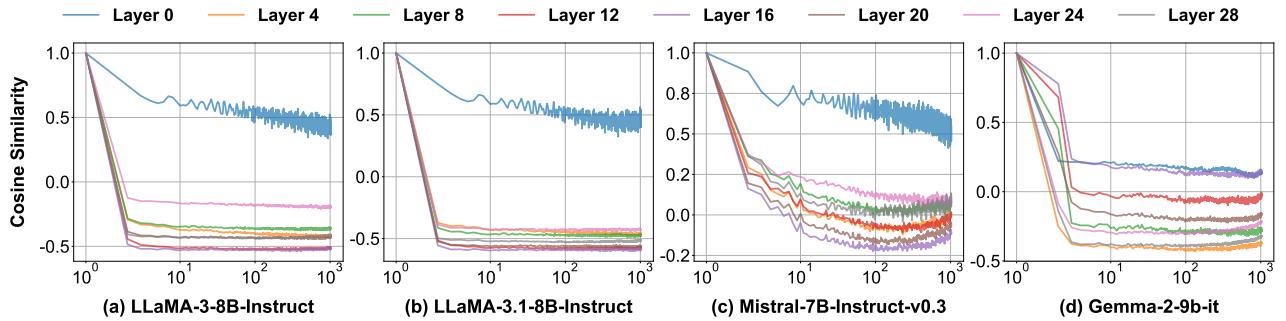
Similarly, Figure 12 shows that value states maintain high cosine similarity across different layers for various initial tokens. There are two notable exceptions: the first layer and the GEMMA-2-9B-IT model. This distinctive pattern in GEMMA-2-9B-IT aligns with the model’s requirement for a system prompt to function correctly.

### B.3 Similarity between the Initial Token and Following Tokens for Key States.

Figure 13 illustrates how the cosine similarity between the initial and subsequent key states stabilizes as position increases. This similarity converges to a near-constant value for all base models after 10 tokens.



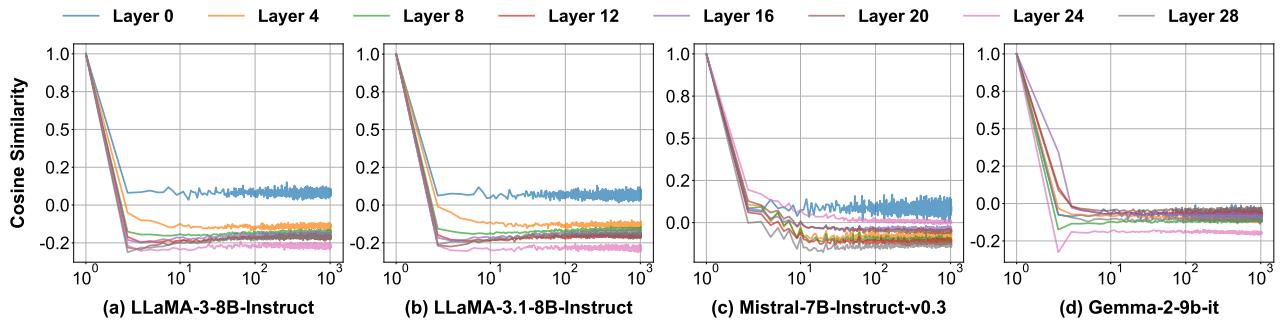
**Figure 12** Among four models, value states from distinct initial tokens exhibit a large cosine similarity than the following positions, except the first layer and GEMMA-2-9B-IT. The X-axis shows positions of value states on a logarithmic scale.



**Figure 13** For all base models, the similarity between the initial key state and subsequent key states stabilizes as the position increases. The X-axis shows positions of key states on a logarithmic scale.

#### B.4 Similarity between the Initial Token and Following Tokens for Value States.

Similar to key states, the value states exhibit a stable similarity between the initial token and subsequent tokens in Figure 14, with all models convergent to a nearly constant value after approximately 10 tokens.



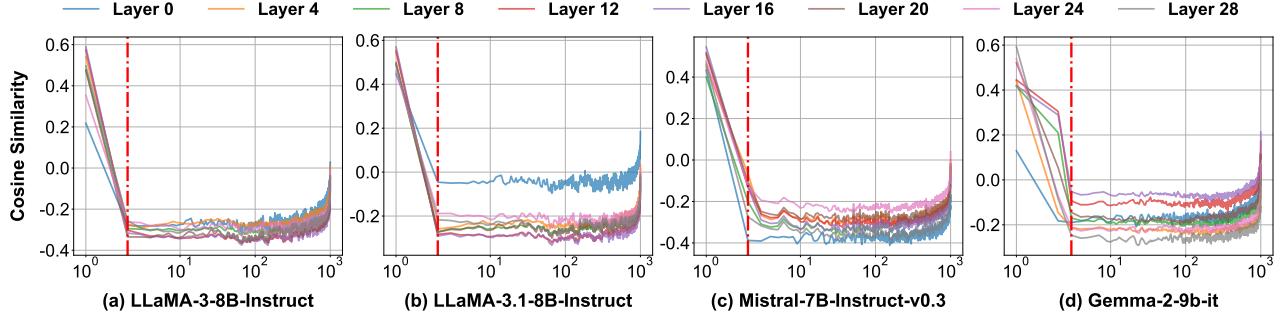
**Figure 14** For all base models, the similarity between the initial value state and subsequent value states stabilizes as the position increases. The X-axis shows positions of value states on a logarithmic scale.

#### B.5 Similarity between the Query State and Past Key States.

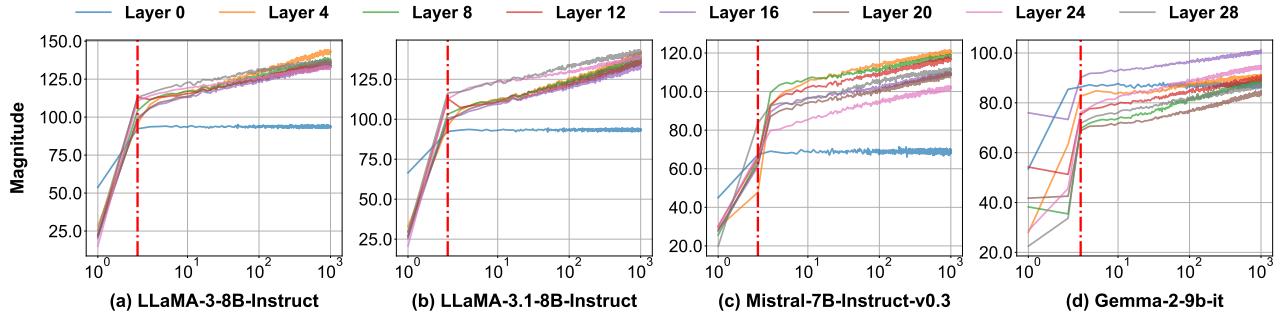
In Figure 15, the query states across all layers, and base models exhibit higher cosine similarity with the initial tokens. Additionally, neighboring positions tend to receive higher cosine similarity.

#### B.6 Magnitude of Key States from Different Positions.

Figure 16 illustrates that the magnitude of key states gradually increases with position, except for the first few tokens, which exhibit significantly smaller magnitudes.

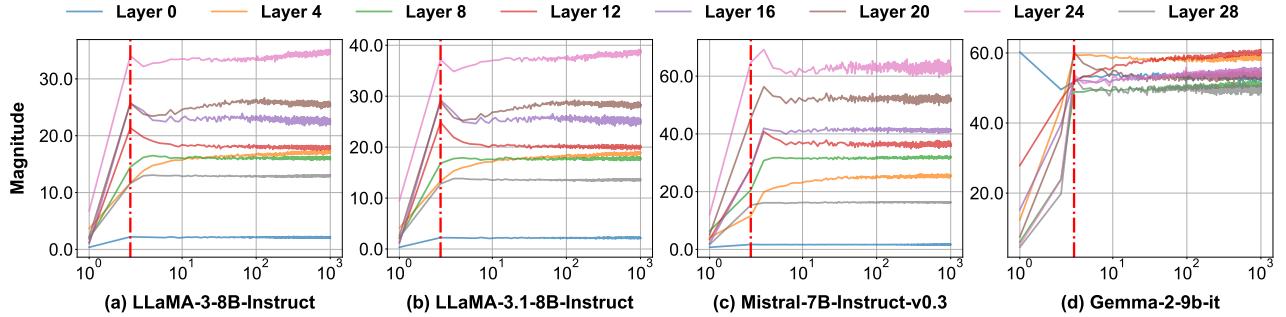


**Figure 15** For all base models, the cosine similarity between the query state and past key states stabilizes for most positions, except for the initial and recent key states. The X-axis shows positions of key states on a logarithmic scale.



**Figure 16** For all models, key states show a slowly upward trend in magnitude as position increases. A red dashed line marks the anomalous region for the first few tokens. The X-axis shows positions of key states on a logarithmic scale.

## B.7 Magnitude of Value States from Different Positions.

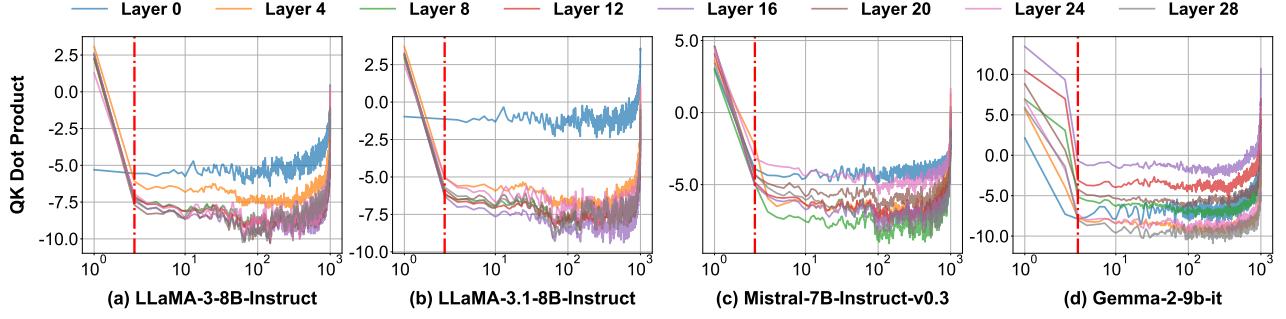


**Figure 17** For all models, the magnitude of value states remains consistent for most positions, except for the first few positions highlighted by a red dashed line. The X-axis represents the positions of value states on a logarithmic scale.

In Figure 17, the value states across all positions exhibit a similar magnitude, except for the first few positions, which show a noticeable deviation. We indicate this region with a red dashed line.

## B.8 Dot Product between the Query State and Past Key States.

In Figure 18, the query states across all layers, and base models exhibit larger dot product values with the initial tokens. Additionally, neighboring positions also tend to receive larger values.



**Figure 18** For all base models, the dot product values between the query state and past key states stabilizes for most positions, except for the initial and recent key states. The X-axis shows positions of key states on a logarithmic scale.

## C Formal Derivation of APE

### C.1 Hierarchical Formula for Softmax Attention.

Here, we begin with the standard Softmax attention, where  $Q$ ,  $K$ , and  $V$  are the query, key, and value states from the input, respectively. To distinguish different sources, we use the subscript  $C_i$  for elements originating from the context, while those without a subscript correspond to user queries or generated texts.

$$O = \text{Softmax} \left( \frac{Q[K_{C_1}^\top, \dots, K_{C_N}^\top, K^\top]}{\sqrt{d}} \right) \times [V_{C_1}, \dots, V_{C_N}, V] \quad (6)$$

$$= \frac{[A_{C_1}, \dots, A_{C_N}, A]}{\sum_{i=1}^N \sum_{j=1}^{l_{C_i}} a_{C_i,j} + \sum_{j=1}^l a_j} \times [V_{C_1}, \dots, V_{C_N}, V], \quad (7)$$

where  $K_{C_i} = [k_{C_i,1}, \dots, k_{C_i,l_{C_i}}]$ ,  $V_{C_i} = [v_{C_i,1}, \dots, v_{C_i,l_{C_i}}]$ ,  $A_{C_i} = [\exp \frac{Qk_{C_i,1}^\top}{\sqrt{d}}, \dots, \exp \frac{Qk_{C_i,l_{C_i}}^\top}{\sqrt{d}}]$ ,

$$A = [\exp \frac{Qk_1^\top}{\sqrt{d}}, \dots, \exp \frac{Qk_l^\top}{\sqrt{d}}], a_{C_i,j} = \exp \frac{Qk_{C_i,j}^\top}{\sqrt{d}}, \text{ and } a_j = \exp \frac{Qk_j^\top}{\sqrt{d}}.$$

We can restructure the computation hierarchically, first computing  $V_{C_i}^h$  and  $A_{C_i}^h$  for each context  $C_i$ :

$$V_{C_i}^h = \text{Softmax} \left( \frac{Q[k_{C_i,1}^\top, \dots, k_{C_i,l_{C_i}}^\top]}{\sqrt{d}} \right) \times [V_{C_i,1}, \dots, V_{C_i,l_{C_i}}], A_{C_i}^h = \text{LogSumExp} \left( \frac{Q[k_{C_i,1}^\top, \dots, k_{C_i,l_{C_i}}^\top]}{\sqrt{d}} \right) \quad (8)$$

Similarly, for the non-context tokens, we compute:

$$V^h = \text{Softmax} \left( \frac{Q[k_1^\top, \dots, k_l^\top]}{\sqrt{d}} \right) \times [V_1, \dots, V_l], A^h = \text{LogSumExp} \left( \frac{Q[k_1^\top, \dots, k_l^\top]}{\sqrt{d}} \right) \quad (9)$$

After we get all these values, we can combine them while renormalizing with  $A^h$ :

$$O = \text{Softmax}(A_{C_1}^h, \dots, A_{C_N}^h, A^h) \times [V_{C_1}^h, \dots, V_{C_N}^h, V^h] \quad (10)$$

### C.2 Hierarchical Formula for APE.

After incorporating all components in APE, we have a new  $V_{C_i}^{h'}$  and  $A_{C_i}^{h'}$  for each context  $C_i$ :

$$V_{C_i}^{h'} = \text{Softmax} \left( \frac{Q[k_{C_i,1}^\top, \dots, k_{C_i,l_{C_i}}^\top]}{T \cdot \sqrt{d}} \right) \times [V_{C_i,1}, \dots, V_{C_i,l_{C_i}}], A_{C_i}^{h'} = S \cdot \text{LogSumExp} \left( \frac{Q[k_{C_i,1}^\top, \dots, k_{C_i,l_{C_i}}^\top]}{T \cdot \sqrt{d}} \right) \quad (11)$$

For the non-context tokens, including our shared prefix, the formulas of  $V^{h'}$  and  $A^{h'}$  remain unchanged. Here, we introduce separate terms  $V_P^{h'}$  and  $A_P^{h'}$  for the shared prefix. Combining them, we have:

$$O = \text{Softmax} \left( A_P^{h'}, A_{C_1}^{h'}, \dots, A_{C_N}^{h'}, A^{h'} \right) \times [V_P^{h'}, V_{C_1}^{h'}, \dots, V_{C_N}^{h'}, V^{h'}] \quad (12)$$

### C.3 Relation with Equation 5.

Finally, we show that it can be rewritten as Equation 5, with the only difference being that all contexts are treated as a single context. For a token from the position  $j$  in context  $C_i$ , the final attention score  $a''_{C_i,j}$  is

$$\begin{aligned} a''_{C_i,j} &= \frac{\exp(Qk_{C_i,j}^\top / T\sqrt{d})}{\sum_{n=1}^N \sum_{t=1}^{l_{C_n}} \exp(Qk_{C_i,t}^\top / T\sqrt{d})} \\ &\cdot \frac{\exp \left( S \cdot \text{LogSumExp} \left( \frac{Qt[k_{C_1,1}^\top, \dots, Qt[k_{C_1,l_{C_1}}, \dots, k_{C_n,1}^\top, \dots, k_{C_n,l_{C_n}}^\top]}{T \cdot \sqrt{d}} \right) \right)}{\exp \left( S \cdot \text{LogSumExp} \left( \frac{Qt[k_{C_1,1}^\top, \dots, Qt[k_{C_1,l_{C_1}}, \dots, k_{C_n,1}^\top, \dots, k_{C_n,l_{C_n}}^\top]}{T \cdot \sqrt{d}} \right) \right) + \exp \left( \text{LogSumExp} \left( \frac{Q[k_1^\top, \dots, k_l^\top]}{T \cdot \sqrt{d}} \right) \right)} \end{aligned} \quad (13)$$

$$= \frac{\exp(Qk_{C_i,j}^\top / T\sqrt{d})}{\sum_{n=1}^N \sum_{t=1}^{l_{C_n}} \exp(Qk_{C_n,t}^\top / T\sqrt{d})} \cdot \frac{(\sum_{n=1}^N \sum_{t=1}^{l_{C_n}} \exp(Qk_{C_n,t}^\top / T\sqrt{d}))^S}{\sum_{n=1}^N (\sum_{t=1}^{l_{C_n}} \exp(Qk_{C_n,t}^\top / T\sqrt{d}))^S + \sum_{t=1}^l \exp(Qk_t^\top / \sqrt{d})} \quad (14)$$

$$= \frac{\exp(Qk_{C_i,j}^\top / T\sqrt{d}) \cdot (\sum_{t=1}^{l_{C_i}} \exp(Qk_{C_i,t}^\top / T\sqrt{d}))^{(S-1)}}{(\sum_{n=1}^N \sum_{t=1}^{l_{C_n}} \exp(Qk_{C_n,t}^\top / T\sqrt{d}))^S + \sum_{t=1}^l \exp(Qk_t^\top / \sqrt{d})} = \frac{a'_{C_i,j}}{(\sum_{n=1}^N \sum_{t=1}^{l_{C_n}} a'_{C_i,t})^S + \sum_{t=1}^l a_t} \quad (15)$$

This formula is equivalent to Equation 5, except it combines the prefix and other non-context tokens for simplicity. Similarly, for the non-context tokens from position  $j$ , we can derive  $a''_j$  as

$$a''_j = \frac{\exp(Qk_j^\top / \sqrt{d})}{\sum_{n=1}^N (\sum_{t=1}^{l_{C_n}} \exp(Qk_{C_n,t}^\top / T\sqrt{d}))^S + \sum_{t=1}^l \exp(Qk_t^\top / \sqrt{d})} = \frac{a_j}{(\sum_{n=1}^N \sum_{t=1}^{l_{C_n}} a'_{C_i,t})^S + \sum_{t=1}^l a_t} \quad (16)$$

Combining these two components, we obtain the final formula presented in Equation 5.

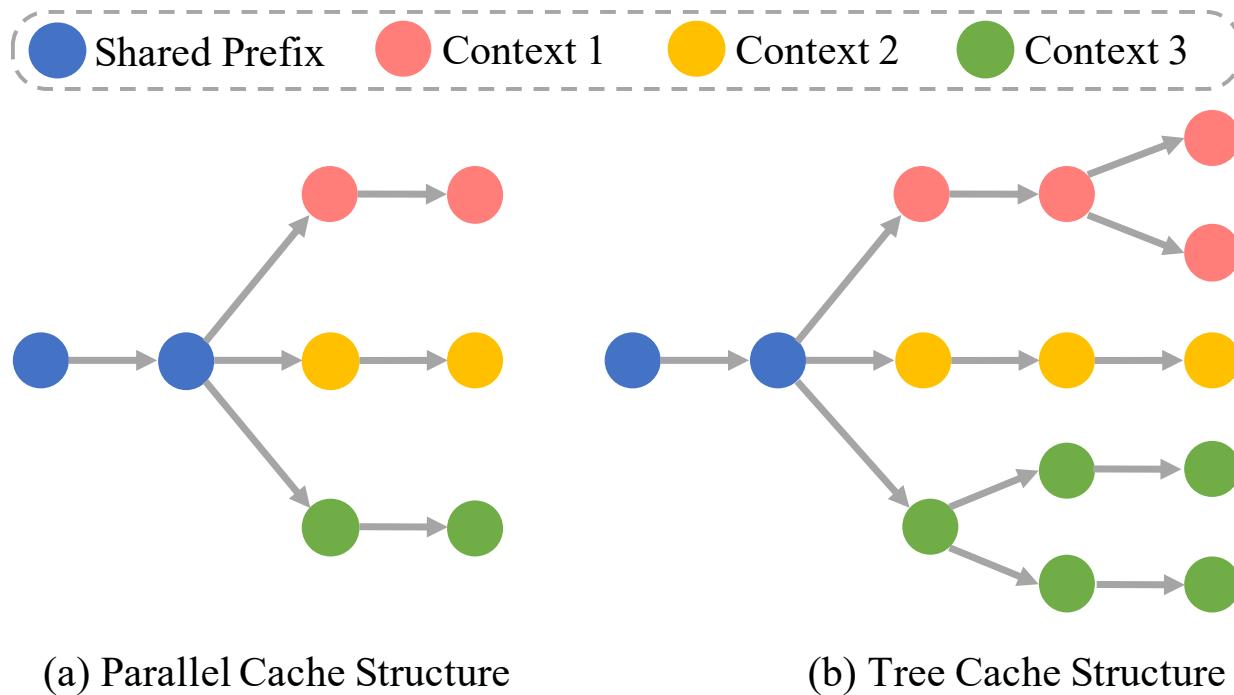
### C.4 Efficient Implementation.

To combine the computation for context and non-context tokens, we employ flash attention twice—once for each part—and then merge the results. This only introduces a marginal computational overhead, as shown below.

```
def ape_attention(query, key, value, temperature, scale):
    # split key and value states into context and non-context parts
    key_context, key_other = key
    value_context, value_other = value
    attn_output_context, lse_context = flash_attn(query, key, value, temperature = temperature)
    attn_output_other, lse_other = flash_attn(query, key, value)
    lse_context = lse_context*(scale)
    attn_weights = [lse_context, lse_other]
    attn_weights = Softmax(attn_weights)
    value_states = [attn_output_context, attn_output_other]
    attn_output = attn_weights @ value_states
```

### C.5 Future Directions.

The hierarchical formulation of APE can naturally extend to more complex tree structures, as illustrated in Figure 19. This flexibility allows each user query to be enriched with external knowledge organized in such structures, demonstrating APE’s capability to handle structured external data effectively.



**Figure 19** Beyond the parallel cache structure discussed in the main paper, APE can be extended to handle more complex cache structures from external sources, where each context forms a tree-like hierarchy. In this setup, computations can be performed hierarchically along each branch, progressively merging intermediate results into the final value state.

## D Comparing APE with Long-context LLMs.

In Table 7, we further compare APE with Long-context LLM, including: (i) *Prompt Compression*: Truncation, LLMLingua2 (Pan et al., 2024), (ii) *KV Cache Eviction*: StreamingLLM (Xiao et al., 2023), (iii) *Long-context FT*: Llama-3-8B-Instruct-262K (Gradient, 2024), Llama-2-7B-Instruct-32K (Together, 2023), (iv) *length extrapolation*: Self-Extend (Jin et al., 2024). Experimental results show that APE consistently outperforms all existing long-context LLM methods. We hypothesize that this improvement stems from APE enabling queries to access all past contexts, enhancing retrieval ability. However, since APE has limitations in identifying relationships between contexts, we do not emphasize its performance on current long-context tasks.

## E APE Cache versus Prefix Cache

Finally, we compare the APE cache with the prefix cache to highlight our advantages in serving multiple queries within the CAG setting. Figure 20 illustrates an example with four contexts where both caching strategies are allocated the same budget. Each query retrieves three contexts. Under these conditions, the prefix cache can only match a limited number of combinations, achieving an average hit rate of 41.7%, whereas the APE cache ensures a 100% hit rate. This gap will become even more pronounced as the number of contexts increases.

Table 7 Performance comparison between APE and long-context LLMs on LongBench (Bai et al., 2023).

| Method              | NarratQA     | Qasper       | MultiFQA     | GovReport    | QMSum        | LCC          |
|---------------------|--------------|--------------|--------------|--------------|--------------|--------------|
| LLAMA-3-8B-INSTRUCT | 19.32        | 32.83        | 43.38        | 27.89        | 22.40        | 53.22        |
| LLMLingua2          | 21.00        | 25.78        | 48.92        | 27.09        | 22.34        | 16.41        |
| StreamingLLM        | 16.99        | 28.94        | 11.99        | 25.65        | 19.91        | 40.02        |
| Long-context FT     | 14.88        | 21.70        | 47.79        | <b>32.65</b> | <b>24.76</b> | 55.12        |
| Self-Extend         | 24.82        | 37.94        | 50.99        | 30.48        | 23.36        | 58.01        |
| +APE                | <b>26.87</b> | <b>39.14</b> | <b>59.12</b> | 29.10        | 23.08        | <b>66.09</b> |

| Method              | RepoBench-P  | HotpotQA     | 2WikiMQA     | MuSiQue      | MultiNews    | Average      |
|---------------------|--------------|--------------|--------------|--------------|--------------|--------------|
| LLAMA-3-8B-INSTRUCT | 38.15        | 44.24        | 21.01        | 20.47        | 23.63        | 31.50        |
| LLMLingua2          | 20.56        | 40.16        | 24.72        | 20.85        | 21.34        | 26.29        |
| StreamingLLM        | 26.16        | 32.76        | 20.12        | 17.32        | 21.49        | 23.76        |
| Long-context FT     | 43.05        | 15.89        | 10.49        | 8.74         | 24.28        | 27.21        |
| Self-Extend         | 41.83        | <b>51.09</b> | 24.17        | <b>28.73</b> | <b>24.11</b> | 35.96        |
| +APE                | <b>49.43</b> | 50.11        | <b>28.06</b> | 25.79        | 22.40        | <b>38.11</b> |

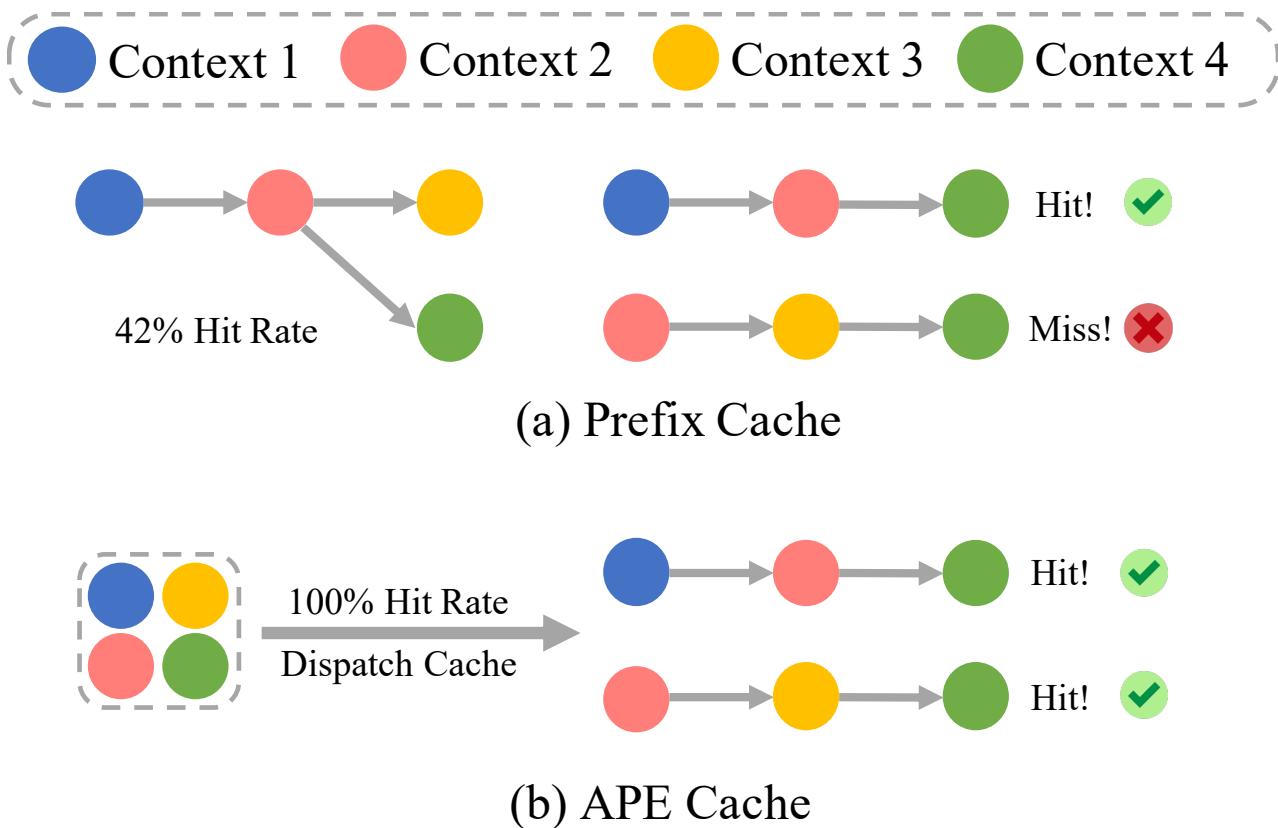


Figure 20 Prefix Cache vs. APE Cache. Our cache can keep a 100% hit rate while the prefix cache only has 42%.