COMB: AN ENCODER PLUG-IN FOR NATIVE POSITION-INDEPENDENT CACHING

Anonymous authors
Paper under double-blind review

ABSTRACT

The Key-Value (KV) cache of Large Language Models (LLMs) is prefix-based, making it highly inefficient for processing contexts retrieved in arbitrary order. Position-Independent Caching (PIC) has been proposed to enable KV reuse without positional constraints; however, existing approaches often incur substantial accuracy degradation, limiting their practical adoption. To address this issue, we propose native PIC by reintroducing the encoder to prevalent decoder-only LLMs and explicitly training it to support PIC. We further develop COMB, a PIC-aware caching system that integrates seamlessly with existing inference frameworks. Experimental results show that COMB reduces Time-to-First-Token (TTFT) by 51-94% and increases throughput by 3× with comparable accuracy. Furthermore, the evaluation on DeepSeek-V2-Lite-Chat demonstrates the applicability of COMB to other types of decoder-only LLMs. Our code is available at https://anonymous.4open.science/r/Comb.

1 INTRODUCTION

Large Language Models (LLMs) exhibit strong capabilities across a wide range of complex tasks, such as automatic software engineering Hu et al. (2023); Liang et al. (2025); Wang et al. (2023a), document-question answering Bai et al. (2024), and reasoning-intensive problems Yao et al. (2023). LLMs also offer a user-friendly interface through natural-language-based prompts — sequences of tokens. As LLMs' capabilities and ease of use expand, their usage patterns have shifted from simple conversational tasks to more complex scenarios, such as multi-turn planning, reasoning, tool use, and few-shot learning. This shift has resulted in long prompts with repeated tokens across requests, such as system messages, few-shot learning examples, and documents, whose content is less changing (i.e., static) than user-specific instructions (being dynamic).

Prefix-Based Context Caching (Prefix caching) is an efficient approach that reuses Key-Value (KV) vectors, the intermediate representations of repeated tokens in previous requests, to reduce computation. Prefix caching matches the current request against previous ones to reuse the KV vectors of the longest common prefix. Prefix caching remains the dominant approach in existing systems Deepseek (2024a); Google (2026); Kwon et al. (2023); Zheng et al. (2024), but it requires exact prefix matches across requests, limiting reuse cases in settings such as few-shot learning and Retrieval-Augmented Generation (RAG), where static chunks (e.g., documents) remain unchanged across requests but are preceded by varying prefixes.

To address the limitations of prefix caching, Position-Independent Caching (PIC) Hu et al. (2025); Yao et al. (2025) enables modular reuse of the KV vectors of static tokens, regardless of their prefixes. Specifically, the PIC technique is a two-stage framework analogous to compilation and linking (Figure 1). First, the **compile** stage involves submitting individual static chunks to the LLM to generate and store their respective KV vectors from position zero. Second, the **link** stage loads and concatenates needed KV vectors (in any order/position), with optional computations to recover accuracy. PIC significantly increases reuse opportunities, but it deviates from standard attention mechanisms, resulting in potential accuracy degradation; thus, recovering accuracy becomes its main challenge.

Subsequent studies largely conform to the PIC framework and can be categorized into two paradigms: **post-training PIC** and **training-aware PIC**. **Post-training PIC**, typically recompute a

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

subset of tokens in the link stage to recover accuracy Agarwal et al. (2025); Hu et al. (2025); Yao et al. (2025); Zhang et al. (2025); Zhao et al. (2025). These approaches are non-intrusive (they do not train the model or modify model architectures) and can be deployed as plug-ins, but they generally suffer from reduced accuracy due to training-inference inconsistency. **Training-aware** PIC explicitly trains models to be aware of the PIC usage during the compile stage, thereby achieving near-zero overhead in the link stage Lu et al. (2025); Ma et al. (2025); Yang et al. (2025). These approaches achieve high accuracy, but they require fine-tuning and thus permanently alter model behavior, which may lead to catastrophic forgetting and performance degradation on other tasks (as analyzed in Figure 6).

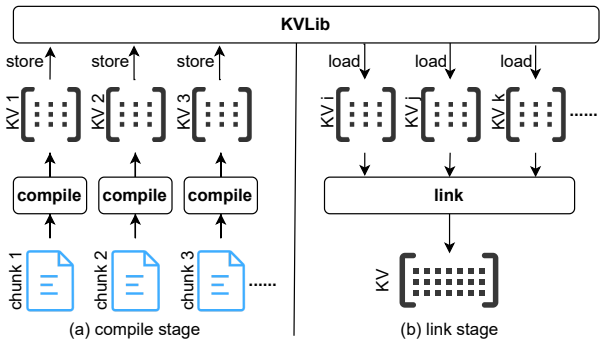


Figure 1: Position-Independent Caching (PIC) workflow. KV_i, KV_j, KV_k indicate KV vectors in arbitrary order/position.

To address these limitations and combine the advantages of both paradigms, we propose COMB, which reintroduces an encoder plug-in into mainstream decoder-only LLMs and trains only the encoder to be PIC-aware. This design offers three key benefits. First, compared to post-training approaches, COMB achieves the highest accuracy by integrating a PIC-specific component — the encoder — and training it explicitly for PIC generation, thereby making PIC a native capability of the model, analogous to native sparse attention in DeepSeek Yuan et al. (2025). Second, compared to training-aware approaches, COMB provides maximal flexibility: the encoder functions as a plug-in whose cross-attention can be entirely removed without affecting the standard decoding workflow. Third, compared to all approaches, despite introducing additional parameters (from the encoder), COMB achieves the lowest Time-To-First-Token (TTFT) due to the efficiency of COMB's encoder-decoder architecture (see details in Section 3).

We implement COMB with two components: a model component and a system component. For the model component, we reintroduce an encoder into standard decoder-only architectures, freeze the decoder parameters, and explicitly train the encoder to accommodate PIC. Input chunks are independently processed by the encoder to generate their KV vectors, and queries over these chunks are supervised using a maximum-likelihood objective against ground-truth outputs. Architecturally, the encoder layers are interleaved with the decoder in a comb-like manner and perform cross-attention only, motivating the name of COMB. For the system component, we implement a PIC management system on top of the existing inference framework, including HuggingFace transformers and vLLM Kwon et al. (2023), comprising approximately 5K lines of Python code. The code and models that we trained are publicly available online and have been anonymized.

Experimental results on LongBench Bai et al. (2024) show that COMB achieves the highest accuracy and the lowest TTFT among all evaluated approaches. When cache hits occur, COMB reduces TTFT by up to 94% and improves throughput by up to 3×, while matching or exceeding the accuracy of prefix-based attention. Importantly, these gains are achieved without permanently modifying the underlying decoder-only model: COMB remains a plug-in mechanism that can be enabled or disabled on demand, reverting to standard prefix caching when PIC is not used. Furthermore, COMB saves memory usage for KV vectors by 75%.

2 BACKGROUND

Context caching techniques can be broadly categorized into prefix-based caching and PIC. In this work, we focus on PIC, which enables the reuse of cached KVs of static prompt segments independent of their positions or surrounding context. Existing PIC approaches generally follow a two-stage framework analogous to compilation and linking (Figure 1), as introduced in Section 1.

Prior PIC approaches can be further classified into *post-training PIC* and *training-aware PIC*. **Post-training PIC** approaches, including EPIC Hu et al. (2025) and CacheBlend Yao et al. (2025), treat the underlying LLMs as fixed and recover accuracy through selective recomputation during the link stage. For example, EPIC recomputes initial tokens of each chunk Hu et al. (2025), while CacheBlend recomputes a subset of tokens with the largest discrepancies Yao et al. (2025). **Training-aware PIC** approaches explicitly incorporate PIC into model training Lu et al. (2025); Ma et al. (2025); Yang et al. (2025), enabling near-zero-overhead reuse while maintaining high accuracy. For example, BlockAttention Ma et al. (2025) enforces PIC by training the model to restrict attention within document blocks, preventing cross-document interactions. KVLink Yang et al. (2025) introduces and trains special linking tokens that reestablish effective self-attention across independently cached documents.

3 Model Design

We now describe the architecture of COMB and analyze its computational and memory complexity. As illustrated in Figure 2, COMB resembles a classical encoder–decoder model but differs in key aspects. The decoder's goal is text generation. During decoding, newly generated tokens are appended to the query sequence and treated identically to the initial query tokens. The encoder is trained to support PIC. Since all decoder parameters are frozen during training, disabling the encoder and the cross-attention layers in Figure 2 recovers the original decoder-only LLM.

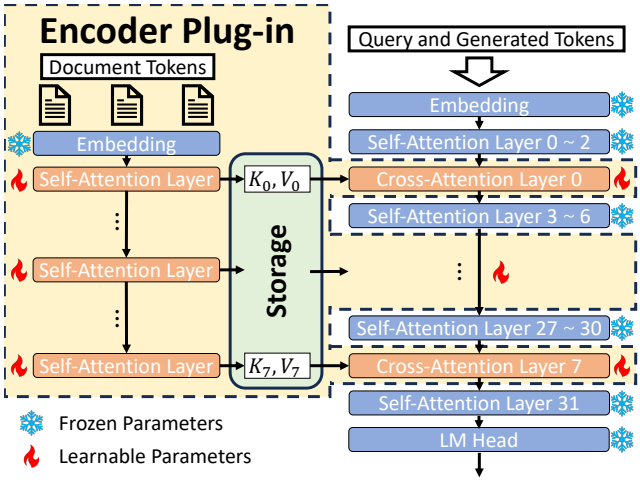


Figure 2: Model architecture.

Throughout, we denote the hidden dimension by d, the document length by l_{doc}, and the query length by l_{query}. We present all attention computations in the single-head case; the extension to multi-head attention is standard and does not affect asymptotic complexity.

3.1 Self-Attention Layer

The decoder (blue self-attention layers in Figure 2) is taken directly from an existing decoder-only LLM that handles only queries and generated tokens. The encoder (orange self-attention layers in Figure 2) shares the same per-layer structure as the decoder (self-attention plus feed-forward), but has *fewer* layers than the decoder (e.g., 8 encoder layers interleaved with 32 decoder layers in Figure 2). Its embedding layer inherits from the decoder and remains frozen. The encoder processes document tokens and produces Position-Independent Caches (PICaches[1]).

Both the encoder and decoder comprise multiple self-attention layers. In a self-attention layer, let the input of this layer be $X \in \mathbb{R}^{l_{\text{query}} \times d}$. $W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$ are the layer's query, key, and value projection matrices. We first compute

$$Q = XW_Q, K = XW_K, V = XW_V, \quad Q, K, V \in \mathbb{R}^{l_{\text{query}} \times d}, \tag{1}$$

$$\tilde{Q}, \tilde{K} = \text{RoPE}(Q, K), \quad \tilde{Q}, \tilde{K} \in \mathbb{R}^{l_{\text{query}} \times d}, \tag{2}$$

[1]In this paper, we use the term *PIC* to refer to the position-independent caching technology, and the term *PICache* to refer to the actual position-independent KV cache.

where RoPE is the rotary position embedding Su et al. (2024). Let $M \in \mathbb{R}^{l_{\text{query}} \times l_{\text{query}}}$ be the causal mask, and then masked self-attention is given by

$$A = \text{softmax}\left(\frac{\tilde{Q}\tilde{K}^\top}{\sqrt{d}} + M\right), H = AV, \quad A \in \mathbb{R}^{l_{\text{query}} \times l_{\text{query}}}, H \in \mathbb{R}^{l_{\text{query}} \times d}, \tag{3}$$

where H is the obtained hidden states to be processed by the next module. The remaining feed-forward, residual, and normalization components also follow the standard decoder-only architecture.

3.2 Cross-attention layer

Encoder–decoder interactions occur in the orange cross-attention layers in Figure 2. In these layers, decoder tokens (query tokens or newly generated tokens) attend to the encoder-produced document KVs. Let the output of an encoder's self-attention layer be $H^E \in \mathbb{R}^{l_{\text{doc}} \times d}$, and the output of the previous decoder's self-attention layer be $X^D \in \mathbb{R}^{l_{\text{query}} \times d}$. The query, key, and value are given by

$$\begin{aligned} Q^{\text{cross}} = X^D W_Q^{\text{cross}}, K^{\text{cross}} = H^E W_K^{\text{cross}}, V^{\text{cross}} = H^E W_V^{\text{cross}}, \\ Q^{\text{cross}} \in \mathbb{R}^{l_{\text{query}} \times d}, \quad K^{\text{cross}}, V^{\text{cross}} \in \mathbb{R}^{l_{\text{doc}} \times d}, \end{aligned} \tag{4}$$

where $W_Q^{\text{cross}}, W_K^{\text{cross}}, W_V^{\text{cross}} \in \mathbb{R}^{d \times d}$ are the query, key, and value projection of the cross-attention layer. To distinguish the order of documents, $\tilde{Q}^{\text{cross}}, \tilde{K}^{\text{cross}} = \text{RoPE}(Q^{\text{cross}}, K^{\text{cross}})$. Cross-attention then computes

$$A^{\text{cross}} = \text{softmax}\left(\frac{\tilde{Q}^{\text{cross}}(\tilde{K}^{\text{cross}})^\top}{\sqrt{d}}\right), H^{\text{cross}} = A^{\text{cross}} V^{\text{cross}}, A^{\text{cross}} \in \mathbb{R}^{l_{\text{query}} \times l_{\text{doc}}}, H^{\text{cross}} \in \mathbb{R}^{l_{\text{query}} \times d}. \tag{5}$$

The cross-attention output H^{cross} is passed through the standard decoder pathway via residual connections inside the decoder blocks. There are two options to implement PIC: (1) store H^E, or (2) store K^{cross} and V^{cross}. The first option saves storage space, but introduces computational overhead after the PICache is loaded. As a result, we store $(K^{\text{cross}}, V^{\text{cross}})$ for efficiency in the current design. The KV vectors form the document-side representations which are stored and reused across requests by COMB. The PICaches of multiple documents can be generated independently from position zero during the compile stage and later concatenated along the l_{doc} during inference (link stage).

3.3 Complexity analysis

We now compare the computational and memory complexity of COMB with a standard decoder-only architecture. Let $N = l_{\text{doc}} + l_{\text{query}}$ denote the total sequence length.

Time Complexity. A standard decoder-only architecture with L_{D} layers (e.g., $L_{\text{D}} = 32$) applies self-attention to all N tokens in each layer, yielding per-layer complexity $O(N^2)$ and total

$$T_{\text{dec-only}} = O\big(L_{\text{D}} N^2\big) = O\big(L_{\text{D}} (l_{\text{query}} + l_{\text{doc}})^2\big). \tag{6}$$

In COMB, we use L_{D} decoder layers and L_{E} encoder layers (e.g., $L_{\text{D}} = 32$, $L_{\text{E}} = 8$ in Figure 2). The encoder attends only over document tokens, with per-layer cost $O(l_{\text{doc}}^2)$ and total $T_{\text{E}} = O\big(L_{\text{E}}\, l_{\text{doc}}^2\big)$. Decoder self-attention operates only on query tokens, with per-layer cost $O(l_{\text{query}}^2)$ and total $T_{\text{D,self}} = O\big(L_{\text{D}}\, l_{\text{query}}^2\big)$. Each cross-attention layer (orange blocks in Figure 2) mixes l_{query} decoder queries with l_{doc} document KVs, for per-layer cost $O(l_{\text{query}}\, l_{\text{doc}})$ and total $T_{\text{cross}} = O\big(L_{\text{cross}}\, l_{\text{query}}\, l_{\text{doc}}\big)$, where $L_{\text{cross}} \le L_{\text{D}}$ is the number of cross-attention layers (e.g., $L_{\text{cross}} = 8$ interleaved layers in Figure 2). The overall attention cost of COMB is

$$T_{\text{comb}} = O\big(L_{\text{E}}\, l_{\text{doc}}^2 + L_{\text{D}}\, l_{\text{query}}^2 + L_{\text{cross}}\, l_{\text{query}}\, l_{\text{doc}}\big). \tag{7}$$

Since L_{E} and L_{cross} are less than L_{D}, $T_{\text{comb}} < T_{\text{dec-only}}$.

Two things are worth noting. First, in practice, each document is independently prefilled by the encoder. When prefilling m documents with lengths $l_{\text{doc},1}, \ldots, l_{\text{doc},m}$ such that $l_{\text{doc}} = \sum_{i=1}^{m} l_{\text{doc},i}$, the

encoder prefill cost scales as $\sum_{i=1}^{m} l_{\text{doc},i}^2$ rather than l_{doc}^2, and we have $\sum_{i=1}^{m} l_{\text{doc},i}^2 \leq l_{\text{doc}}^2$. Second, modern inference engines already employ chunked prefilling for long prompts by processing them in smaller segments iteratively. As a result, the initial PIC compilation ("cold start") of a new document typically aligns with this existing procedure and, in cache-miss scenarios, does not introduce noticeable overhead beyond that of the baseline system.

Memory Complexity. We now compare the memory required for KV vectors. In the standard decoder-only architecture, each of the L_{D} layers stores KVs for all N tokens, for total KV memory

$$M_{\text{dec-only}} = O\big(L_{\text{D}} N d\big) = O\big(L_{\text{D}}(l_{\text{query}} + l_{\text{doc}})d\big). \tag{8}$$

In COMB, encoder KV vectors are stored for document tokens across L_{E} layers, with memory $M_{\text{E}} = O\big(L_{\text{E}}\, l_{\text{doc}} d\big)$. Decoder KV vectors are stored only for query tokens, $M_{\text{D}} = O\big(L_{\text{D}}\, l_{\text{query}} d\big)$. Thus, the total KV memory of COMB is

$$M_{\text{comb}} = O\big(L_{\text{E}}\, l_{\text{doc}} d + L_{\text{D}}\, l_{\text{query}} d\big). \tag{9}$$

Since $L_{\text{E}} < L_{\text{D}}$, $M_{\text{comb}} < M_{\text{dec-only}}$.

Finally, L_{E} can be chosen in the range $[1, L_{\text{D}}]$. Larger L_{E} increases expressivity and potentially accuracy, while even $L_{\text{E}} = L_{\text{D}}$ does not exceed the time or memory complexity of a decoder-only model. In our experiments, we set $L_{\text{E}} = 8$ to balance accuracy and computational resources (Section 5). Moreover, under our limited GPU resources, this configuration avoids out-of-memory errors and accelerates training by enabling larger batch sizes due to reduced parameter counts, optimizer states, and activation memory.

4 IMPLEMENTATION

This section describes how we instantiate, train, and deploy COMB. We first detail the model information, then summarize the training data and recipe, and finally discuss how COMB is integrated into a production serving system.

4.1 MODELS

We implement COMB by augmenting a frozen decoder-only LLM with a trainable encoder described in Section 3. First, the decoder is directly adopted from existing open-source models. Specifically, we use two open-source LLMs: Llama-3.1-8B-Instruct Meta AI (2024a) and DeepSeek-V2-Lite-Chat Deepseek (2024b) as the decoder backbones. Llama represents the most widely used architecture in the current LLM ecosystem, whereas DeepSeek departs from the standard KV-cache design by employing Multi-head Latent Attention (MLA) for improved memory efficiency. Together, these models cover diverse architectural choices and training paradigms. Second, we integrate an encoder into the decoder via cross-attention layers, where only the encoder parameters and the query-side projections in the cross-attention layers are trainable. The resulting models are referred to as COMBLLAMA and COMBDEEPSEEK, respectively. We train all models using the AdamW optimizer with linear warmup followed by cosine decay. Unless otherwise stated, gradient clipping and weight decay follow standard practices for large-scale language model fine-tuning. Additional implementation details are provided in the code released.

4.2 TRAINING DATA AND RECIPE

COMB is trained to emulate the behavior of the underlying decoder-only model under Position-Independent Caching. We select SQuAD Rajpurkar et al. (2018), Natural-Instructions Mishra et al. (2022), XSum Narayan et al. (2018), and Super-Natural-Instructions Wang et al. (2022) as training datasets. Each training example is constructed as a tuple (D, Q, Y), where D is a (possibly multi-document) static context, Q is a user query or instruction, and Y is the ground-truth response. During training, each document in D is first independently encoded by the encoder to obtain document-side KVs $(K^{\text{cross}}, V^{\text{cross}})$, and the frozen decoder then generates outputs conditioned on $(K^{\text{cross}}, V^{\text{cross}})$ and query tokens. We optimize a token-level cross-entropy loss between the decoder outputs and the target sequence Y, using teacher forcing.

The ground-truth sequence Y is generated by the original decoder-only LLM given (D, Q) as input, resulting in on-policy training for the CombLlama and CombDeepseek models. All models are trained on four NVIDIA A100 80GB GPUs with tensor parallelism set to 4.

4.3 System Details

We build Comb system on top of existing LLM inference frameworks (e.g., huggingface transformers and vLLM Kwon et al. (2023)), with 5K lines of code in Python.

Comb consists of four key components: **PIC manager**, **PIC allocator**, **Chunk processor**, and **Inference engine**. When the system starts, the PIC manager reserves a portion (specified by the operator) of GPU memory, initializes the PIC allocator, and launches the Chunk processor and the Inference engine. After initialization, the PIC manager will serve users' requests, which consist of a query and several documents (possibly none or multiple). First, the documents will be looked up to check if their PICaches exist. If not, the PIC allocator will allocate free memory for their PICaches, and the Chunk processor computes the encoder part for them. Then the PICaches are stored asynchronously. After collecting the PICaches, the PIC manager will send their handle to the Inference engine through the CUDA IPC (Inter-Process Communication) API. This enables tensors to pass in place between processes. Finally, the Inference engine generates a response using the query and the related PICache.

Comb's serving workflow is illustrated in Appendix A. Note that Comb can be seamlessly integrated into the existing disaggregated prefill-decode serving paradigm. The chunk processor can be considered the prefill node, while the LLM inference engine can be considered the decode node.

5 Evaluation

We begin by describing the experimental setup, including datasets, models, evaluation metrics, and software/hardware environment. We then present our key evaluation results.

5.1 Experimental Setup

Dataset. Following CacheBlend Yao et al. (2025) and EPIC Hu et al. (2025), we evaluate on the following LongBench datasets Bai et al. (2024): *2WikiMQA* & *HotpotQA* (multi-document question answering), *MuSiQue* (long-document question answering), *SAMSum* (few-shot instruction following), and *MultiNews* (multi-document summarization). All datasets contain 200 test cases, with the distribution of prompt (prefill) lengths and answer (decode) lengths shown in Figure 3. Static tokens constitute approximately 95%-99% of the prompt, while dynamic tokens are fewer than 50.

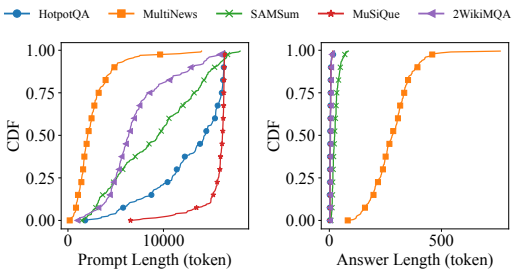
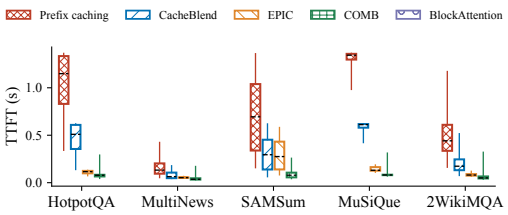


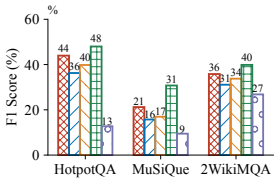
Figure 3: Prefill and decode length distribution.

Metrics. We use the following three metrics to evaluate performance and model accuracy. First, *Time-To-First-Token (TTFT)* Kwon et al. (2023) (lower is better) is used to evaluate all datasets. This metric measures the prefill-stage time: the time from when users send a request to when users receive the first token; this time could be reduced by using context caching. Second, *F1 score* Bai et al. (2024) (higher is better) is used to evaluate *2WikiMQA*, *MuSiQue*, and *HotpotQA*. This metric measures the similarity between LLMs' output and the ground-truth answer based on their common words. Third, *Rough-L score* Lin (2004) (higher is better) is used to evaluate *SAMSum* and *MultiNews*. This metric measures the similarity between LLMs' output and the ground-truth answer by calculating the length of their longest common subsequence.

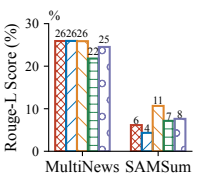
Baselines. We compare Comb against four representative approaches: Prefix caching, CacheBlend Yao et al. (2025), EPIC Hu et al. (2025), and BlockAttention Ma et al. (2025). Prefix caching follows the standard attention mechanism; EPIC and CacheBlend are post-training PIC



(a) TTFT upon cache hit

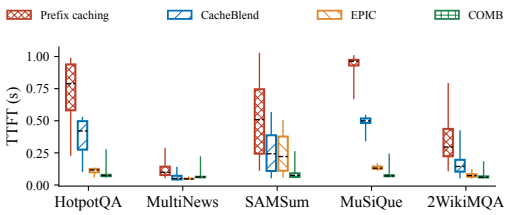


(b) F1 score

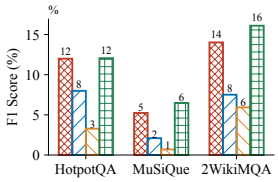


(c) RougeL score

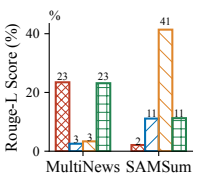
Figure 4: Comparison of TTFT (↓ Better) and accuracy (↑ Better) using Llama-3.1-8B-Instruct. Note that the TTFT of BlockAttention is not shown because it is not integrated with vLLM.



(a) TTFT upon cache hit



(b) F1 score



(c) RougeL score

Figure 5: Comparison of TTFT (↓ Better) and accuracy (↑ Better) using DeepSeek-V2-Lite-Chat. Note that BlockAttention is not shown because it was not trained on DeepSeek-V2-Lite-Chat.

approaches, while BlockAttention is a training-aware PIC approach. We set the recomputation ratio of CacheBlend to 20%, and set the number of recomputed tokens per document to 64 for EPIC.

Environment. We run experiments on a single NVIDIA A100 server with four A100-80GB GPU available. The server has 128-core Intel(R) Xeon(R) Platinum 8358P CPU@2.60GHz with 2 hyper-threading and 1TB DRAM. We use Ubuntu 20.04 with Linux kernel 5.16.7 and CUDA 12.6. The Inference engine that we use for evaluations is vLLM 0.12.0.

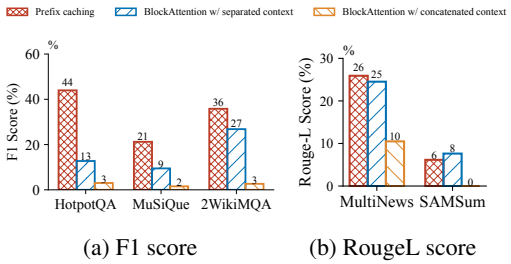
5.2 End-to-end accuracy under PIC

We first examine whether COMB preserves the generation quality of the underlying decoder-only models when PIC is enabled. Figures 4b and 4c report F1 and Rouge-L scores for Llama-3.1-8B-Instruct, and Figures 5b and 5c report the same metrics for DeepSeek-V2-Lite-Chat. Across all datasets and both model families, COMB attains high accuracy among the evaluated PIC-based approaches.

We do not treat absolute accuracy as a primary contribution, as the comparison is not fully controlled. In particular, COMB is explicitly trained to be PIC-aware, whereas EPIC is training-free, and other baselines fine-tune fewer parameters for fewer steps. Rather, these results substantiate our central claim: in PIC usage scenarios, incorporating a PIC-aware component into the model architecture and training it accordingly enables the model to recover — and in some cases exceed — the accuracy of standard prefix-based attention. The strong and consistent performance of COMB across datasets and across two architecturally distinct model families (Llama and DeepSeek with MLA) thus provides empirical evidence for the effectiveness of PIC-aware training. Notably, despite being training-free, EPIC achieves unexpectedly strong performance on the *SAMSum* dataset, in some cases surpassing the prefix-caching baseline. This behavior has been analyzed in detail in the EPIC study Hu et al. (2025), and our results are consistent with their findings.

5.3 Non-intrusiveness of COMB

We emphasize that COMB is non-intrusive. In practical or industrial settings, the encoder can be enabled or disabled on demand. When the encoder is disabled, COMB reduces exactly to the underlying decoder-only model and recovers its original prefix-caching behavior and accuracy, without any degradation (just as the prefix caching shown in Figure 6).



(a) F1 score (b) RougeL score

Figure 6: Comparison of F1 score and RougeL score between using separated context as input to BlockAttention and not using it.

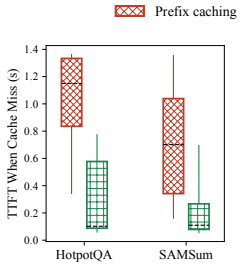


Figure 7: TTFT without KV cache using Llama-3.1-8B.

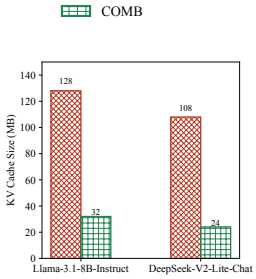


Figure 8: The KV cache size of 1,024 tokens.

In contrast, approaches such as BlockAttention are inherently intrusive. When PIC is not desired, and multiple documents are directly fed into such models, accuracy can collapse. The reason is that BlockAttention is explicitly trained to restrict attention within predefined document blocks. If users do not carefully chunk inputs and apply block attention in its intended manner, the model's assumptions are violated, leading to severe performance degradation (Figure 6a and Figure 6b, blue bars vs. orange bars).

5.4 TTFT UNDER CACHE HITS AND MISSES

We next analyze TTFT with and without PICaches. For each sample in the test set, we first feed it into the inference system and record the TTFT for cache misses as shown in Figure 7. Then, we modify the sample's instruction to simulate another request with the same context but different prompt, thereby recording the TTFT for cache hits as shown in Figures 4a and 5a. The boxes in the figures range from 25th to 75th percentile, and the whiskers go from 5th to 95th percentile. As expected, all PIC approaches substantially reduce TTFT under cache hits, since most document-side computation is amortized across requests. Among existing PIC approaches, EPIC attains particularly low TTFT because it recomputes only several tokens at each chunk boundary during the link stage, incurring minimal overhead. Training-aware approaches such as BlockAttention offer algorithmic advantages, but in practice have not been integrated into production-grade inference engines (e.g., vLLM), and thus are typically evaluated only with vanilla HuggingFace runtime. Therefore, the TTFT of BlockAttention is significantly higher than other approaches, and we do not show it in the figures.

COMB achieves the lowest TTFT due to the lightweight encoder architecture and the time complexity analyzed in Section 3: encoder self-attention is applied only to document tokens, decoder self-attention only to query tokens, and cross-attention couples them in $O(l_{\text{query}}\, l_{\text{doc}})$ time. PICaches for each document are compiled once and reused across subsequent requests, so when caches hit, the decoder only needs to mix relatively short query sequences with precomputed document PICaches via cross-attention.

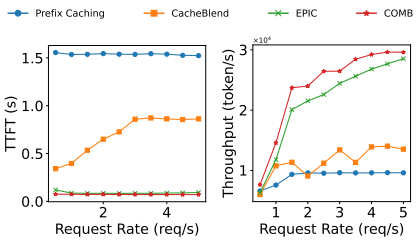
When caches miss, COMB performs a cold-start compilation to generate PICaches for new documents. Even in this regime, TTFT remains competitive with or better than the decoder-only baseline for two reasons. First, the encoder is substantially shallower than the decoder and is applied only to static document tokens, so the additional prefill cost is modest. Second, production-level inference engines such as vLLM already employ chunked prefill: instead of processing extremely long prompts in a single pass, they split inputs into manageable segments and prefill them incrementally to avoid large intermediate activations and out-of-memory errors. PIC compilation in COMB naturally aligns with this chunked prefill behavior, so much of the cold-start work is similar to the computation that the baseline system would perform regardless. As a result, COMB provides low TTFT in both cache-hit and cache-miss scenarios while enabling accurate, model-aware PIC.

5.5 ONLINE LATENCY AND THROUGHPUT UNDER INCREASING LOAD

Finally, we evaluate COMB under increasing request rates in an online serving setting. Figure 10 reports TTFT and throughput as the number of concurrent users and, correspondingly, the amount

of GPU memory used by position-independent caches increase. Across the evaluated load range, COMB maintains the lowest TTFT while simultaneously achieving the highest sustained throughput among all approaches.

These gains are closely tied to the memory complexity in Section 3. In a standard decoder-only architecture, KV memory scales with the *entire* prompt length (documents plus query), quickly exhausting GPU HBM and limiting concurrency. By contrast, COMB stores encoder KVs only for static document tokens across a small number of encoder layers, and decoder KVs only for the much shorter query sequence. This lightweight encoder design reduces per-request KV memory, freeing HBM capacity that can instead be used to batch more requests or admit more users. As shown in Figure 8, COMB saves KV memory by 75% for Llama-3.1-8B-Instruct, and by 78% for DeepSeek-V2-Lite.



(a) TTFT (b) Throughput

Figure 10: The performance of COMB as the request rate increases.

Taken together with the accuracy results above, these findings show that native, PIC-aware training delivers a favorable accuracy–latency–throughput trade-off in realistic serving scenarios.

6 RELATED WORK

Context Caching (CC). CC has two main categories. The first is position-dependent caching, which can be further divided into prefix-based caching Liu et al. (2024); Yu et al. (2025); Zheng et al. (2024) and relative position-dependent caching, such as PromptCache Gim et al. (2024). By mid-2024, vendors such as Gemini Google (2026) and DeepSeek Deepseek (2024a) began incorporating explicit prefix-based CC features into their systems. The second category is Position-Independent Caching (PIC) Hu et al. (2025); Lu et al. (2025); Ma et al. (2025); Yao et al. (2025). Although EPIC Hu et al. (2025) and CacheBlend Yao et al. (2025) address initial aspects of PIC, COMB advances the paradigm by rendering PIC intrinsic to the model architecture.

Post-Training vs. Training-Aware Approaches. A recurring trajectory in LLM research involves initially validating hypotheses via post-training adjustments before integrating them into the model training process. For instance, in the realm of attention sparsity, researchers first explored inference-time heuristics such as H2O Zhang et al. (2023), Quest Tang et al. (2024), and RaaS Hu et al. (2024). Following the validation of these sparsity patterns Jiang et al. (2024), DeepSeek introduced Native Sparse Attention (NSA) Yuan et al. (2025) to incorporate sparse structures directly into the model. This progression from post-training experimentation to training-aware design is also evident in quantization Lin et al. (2024); Xiao et al. (2023), pruning Wang et al. (2023b); Zhang et al. (2021), speculative decoding DeepSeek-AI et al. (2025), and attention sink Qiu et al. (2025). The underlying principle is that once an inductive bias — whether sparsity or position-independent caching — proves beneficial post-training, it merits integration into the model's native architecture. While EPIC Hu et al. (2025) and CacheBlend Yao et al. (2025) represent post-training approaches, Block-Attention Ma et al. (2025), KVLink Yang et al. (2025), TurboRAG Lu et al. (2025), and COMB fall under the training-aware category. Notably, COMB combines the best of both worlds, achieving the accuracy benefits of training-aware approaches while retaining the non-intrusive, plug-in flexibility of post-training approaches.

7 CONCLUSION

In this paper, we have proposed **native PIC** by reintroducing an encoder into decoder-only LLMs and explicitly training it for PIC-compatible generation. We have also presented COMB, a PIC-aware caching system integrated with existing inference frameworks. Extensive experiments demonstrate that COMB achieves the highest accuracy under PIC while substantially improving inference efficiency, reducing TTFT by up to 94% and increasing throughput by up to 3×. In addition, COMB remains a plug-in mechanism that can be enabled or disabled on demand.

AI USE STATEMENT

We have not used generative AI tools for all of the required disclosure tasks. Additionally, we used generative AI tools for debugging issues such as TP configuration and CUDA synchronization. LLM-generated code was verified and tested for correctness by 2 authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The code and model weights are available via the anonymous repository link in the abstract. We have included detailed command instructions in the repository to help reproduce the experiment.

REFERENCES

Shubham Agarwal, Sai Sundaresan, Subrata Mitra, Debabrata Mahapatra, Archit Gupta, Rounak Sharma, Nirmal Joshua Kapu, Tong Yu, and Shiv Saini. Cache-craft: Managing chunk-caches for efficient retrieval-augmented generation. *Proc. ACM Manag. Data*, 3(3), June 2025. doi: 10.1145/3725273. URL https://doi.org/10.1145/3725273.

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. In *Proceedings of the Sixty-Second Annual Meeting of the Association for Computational Linguistics*, pp. 3119–3137, 2024.

Deepseek. Deepseek api introduces context caching on disk, cutting prices by an order of magnitude. https://api-docs.deepseek.com/news/news0802, 2024a.

Deepseek. Deepseek-v2-lite-chat. https://huggingface.co/deepseek-ai/DeepSeek-V2-Lite-Chat, 2024b.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338, 2024.

Google. Gemini context caching. https://ai.google.dev/gemini-api/docs/caching, 2026.

Junhao Hu, Chaozheng Wang, Hailiang Huang, Huang Luo, Yu Jin, Yuetang Deng, and Tao Xie. Predicting compilation resources for adaptive build in an industrial setting. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering*, pp. 1808–1813, 2023.

Junhao Hu, Wenrui Huang, Weidong Wang, Zhenwen Li, Tiancheng Hu, Zhixia Liu, Xusheng Chen, Tao Xie, and Yizhou Shan. RaaS: Reasoning-aware attention sparsity for efficient llm reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 2577–2590, 2024.

Junhao Hu, Wenrui Huang, Haoyi Wang, Weidong Wang, Tiancheng Hu, Qin Zhang, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. EPIC: efficient position-independent caching for serving large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 24391–24402, 2025.

Huiqiang Jiang, YUCHENG LI, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. MInference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=fPBACAbqSN.

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

Qingyuan Liang, Zeyu Sun, Qihao Zhu, Junhao Hu, Yifan Zhao, Yizhou Chen, Mingxuan Zhu, Guoqing Wang, and Lu Zhang. Directional diffusion-style code editing pre-training. *IEEE Trans. Software Eng.*, 51(9):2583–2600, 2025.

Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pp. 74–81, 2004.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of the 7th Annual Conference on Machine Learning and Systems*, 2024.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, pp. 38–56, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706141. doi: 10.1145/3651890.3672274. URL https://doi.org/10.1145/3651890.3672274.

Songshuo Lu, Hua Wang, Yutian Rong, Zhi Chen, and Yaohua Tang. TurboRAG: Accelerating retrieval-augmented generation with precomputed KV caches for chunked text. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 6588–6601, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.334. URL https://aclanthology.org/2025.emnlp-main.334/.

Dongyang Ma, Yan Wang, and Tian Lan. Block-attention for efficient prefilling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=7zNYY1E2fq.

Meta AI. Introducing Llama 3.1: Our most capable models to date. https://ai.meta.com/blog/meta-llama-3-1/, 2024a.

Meta AI. Llama 3.2: Revolutionizing edge AI and vision with open, customizable models. https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/, sep 2024b.

Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. Cross-task generalization via natural language crowdsourcing instructions. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3470–3487, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.244. URL https://aclanthology.org/2022.acl-long.244/.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1797–1807, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1206. URL https://aclanthology.org/D18-1206/.

Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=1b7whO4SfY.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In Iryna Gurevych and Yusuke Miyao (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 784–789, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2124. URL https://aclanthology.org/P18-2124.

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

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 47901–47911, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Chaozheng Wang, Junhao Hu, Cuiyun Gao, Yu Jin, Tao Xie, Hailiang Huang, Zhenyu Lei, and Yuetang Deng. How practitioners expect code completion? In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1294–1306, 2023a.

Yite Wang, Dawei Li, and Ruoyu Sun. NTK-SAP: improving neural network pruning by aligning training dynamics. In *Proceedings of the 11th International Conference on Learning Representations*, 2023b.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, Eshaan Pathak, Giannis Karamanolakis, Haizhi Lai, Ishan Purohit, Ishani Mondal, Jacob Anderson, Kirby Kuznia, Krima Doshi, Kuntal Kumar Pal, Maitreya Patel, Mehrad Moradshahi, Mihir Parmar, Mirali Purohit, Neeraj Varshney, Phani Rohitha Kaza, Pulkit Verma, Ravsehaj Singh Puri, Rushang Karia, Savan Doshi, Shailaja Keyur Sampat, Siddhartha Mishra, Sujan Reddy A, Sumanta Patro, Tanay Dixit, and Xudong Shen. Super-NaturalInstructions: Generalization via declarative instructions on 1600+ NLP tasks. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 5085–5109, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.340. URL https://aclanthology.org/2022.emnlp-main.340/.

Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 38087–38099, 2023.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. KVLink: Accelerating large language models via efficient KV cache reuse. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=oDcAGSXZZP.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving for rag with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, pp. 94–109, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400711961. doi: 10.1145/3689031.3696098. URL https://doi.org/10.1145/3689031.3696098.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The 11th International Conference on Learning Representations*, 2023.

Lingfan Yu, Jinkun Lin, and Jinyang Li. Stateful large language model serving with pensieve. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, pp. 144–158, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400711961. doi: 10.1145/3689031.3696086. URL https://doi.org/10.1145/3689031.3696086.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 23078–23097, 2025.

Zeru Zhang, Jiayin Jin, Zijie Zhang, Yang Zhou, Xin Zhao, Jiaxiang Ren, Ji Liu, Lingfei Wu, Ruoming Jin, and Dejing Dou. Validating the lottery ticket hypothesis with inertial manifold theory. In *Proceedings of the 35th Annual Conference on Neural Information Processing Systems*, pp. 30196–30210, 2021.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. H2O: heavy-hitter oracle for efficient generative inference of large language models. In *Proceedings of the 37th Annual Conference on Neural Information Processing Systems*, pp. 34661–34710, 2023.

Zhisong Zhang, Yan Wang, Xinting Huang, Tianqing Fang, Hongming Zhang, Chenlong Deng, Shuaiyi Li, and Dong Yu. Attention entropy is a key factor: An analysis of parallel context encoding with full-attention-based pre-trained language models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9840–9855, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.485. URL https://aclanthology.org/2025.acl-long.485/.

Shiju Zhao, Junhao Hu, Rongxiao Huang, Jiaqi Zheng, and Guihai Chen. Mpic: Position-independent multimodal context caching system for efficient mllm serving, 2025. URL https://arxiv.org/abs/2502.01960.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=VqkAKQibpq.

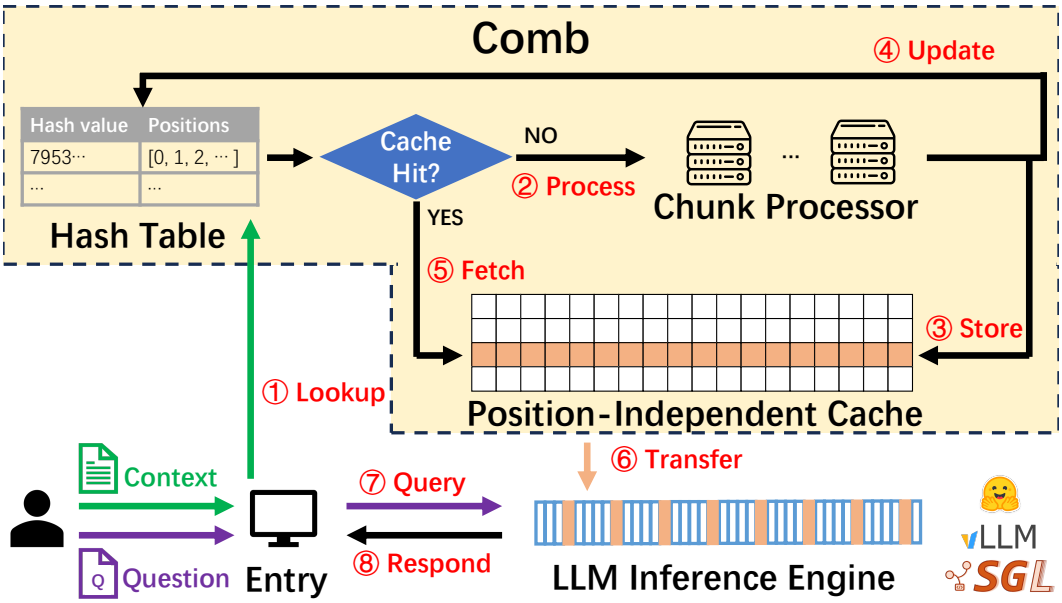


Figure 11: COMB Overview.

A SYSTEM OVERVIEW

The workflow of COMB serving system is shown in Figure 11. The input of COMB consists of a question and possibly none or multiple contexts, which is expected to be reused in an arbitrary position. We let users distinguish between context and query because users know their own needs best. ① For multiple contexts, COMB looks up whether their PICaches exist through a hash table. ② For those contexts without caches, a chunk processor will generate their KV caches, ③ store the KV caches, and ④ update the hash table. Then the PICaches of these contexts are fetched ⑤ and transferred to an LLM inference engine (e.g., HuggingFace transformers, vLLM, or SGLang) ⑥. ⑦ The question is passed directly to the LLM inference engine, and is responded to with the PICaches ⑧. The blue and orange squares in the lower right corner of the figure represent the self-attention layers and cross-attention layers of LLM, as illustrated in Section 3.

B TRAINING OVERHEAD

Table 1 summarizes the parameter counts and training overhead.

Table 1: Model size and training overhead for COMB.

MODEL NAME	# OF FROZEN PARAMETERS	# OF LEARNABLE PARAMETERS	GPU HOURS (A100 * HOUR)
COMBLLAMA	8.0 B	3.5 B	742
COMBDEEPSEEK	15.7 B	3.5 B	1351

C DISCUSSION

C.1 WHY DOES COMB WORK?

Our journey began with optimizing PIC accuracy, driven by the goal of integrating PIC as a native architectural feature rather than an afterthought. This led us to an encoder-decoder design leveraging cross-attention—a conviction we held firmly even before committing substantial resources to large-scale training.

This approach rests on two foundational observations. First, the Transformer architecture was originally conceived as an encoder-decoder framework Vaswani et al. (2017). The encoder's inherent strength in comprehension tasks makes it ideally suited to process arbitrarily ordered contexts, perfectly complementing the decoder's generation capabilities. Second, this paradigm has already been validated in multimodal LLMs such as Whisper Radford et al. (2023) and Mllama Meta AI (2024b). Multimodal tasks are inherently more challenging than pure language tasks—requiring the interpretation of text potentially embedded within images and complex visual reasoning. If this architecture proves effective for such demanding multimodal scenarios, its applicability to pure language modeling follows naturally.

C.2 The potential of PIC

We have now entered the era of AI Agents. What is an Agent, and how does it differ from an LLM? A widely accepted view holds that Agents can formulate short-term actions based on long-term goals and are capable of using tools. Consequently, *retrieval* emerges as a crucial capability for Agents. Beyond consulting references to gather information, Agents must also retrieve which tools are best suited for specific tasks.

Whenever retrieval is involved, PIC becomes essential, because retrieved items can arrive in *arbitrary order* under various permutations and combinations. Using prefix caching in this scenario proves highly inefficient: essentially only the first item can reuse the KV cache, while all subsequent items fail to do so.

We can address this by equipping the Agent with an *Encoder*: all retrieved content—arriving in arbitrary order—is fed into the Encoder, while the question and the model's reasoning remain in the Decoder. When the model needs to retrieve new information, previously retrieved content is discarded and replaced with new items in the Encoder.

Imagine how powerful such an Agent would be if the Decoder's 128K context window contained only the question and the model's own chain of thought, free from noisy reference materials.

D Limitation

To address the training-inference inconsistency induced by post-training PIC methods, COMB requires a modest amount of training cost (thousands of GPU hours), which is between the training cost of low-rank adaptation (dozens of GPU hours) and pre-training (millions of GPU hours).

However, we believe that the benefits outweigh the training cost. Although COMB incurs a one-time offline training cost, it delivers both (1) substantially better accuracy than post-training PIC methods, and (2) substantial serving gains in terms of TTFT, throughput, and KV memory. The training cost is paid once, while the serving benefits accrue over a sustained reuse-heavy deployment. For the large-scale serving scenarios we target, this amortization is exactly what makes the approach attractive. More broadly, we hope this result will encourage big companies to revisit their model architecture choices. If future models incorporate an encoder path earlier in training (e.g., during mid-training or broader domain adaptation), then PIC could become a native capability rather than something retrofitted after the fact. We see COMB as evidence that this direction is promising.