RetrievalAttention: Accelerating Long-Context LLM Inference via Vector Retrieval

Di Liu[◇,*], **Meng Chen**[♠,*], **Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang**[♠], **Chen Chen**[◇], **Fan Yang, Yuqing Yang, Lili Qiu**
Microsoft Research, [◇]Shanghai Jiao Tong University, [♠]Fudan University
{baotonglu, hjiang, qiazh, yuqyang}@microsoft.com

Abstract

Transformer-based Large Language Models (LLMs) have become increasingly important. However, due to the quadratic time complexity of attention computation, scaling LLMs to longer contexts incurs extremely slow inference latency and high GPU memory consumption for caching key-value (KV) vectors. This paper proposes **RetrievalAttention**, a training-free approach to both accelerate attention computation and reduce GPU memory consumption. By leveraging the dynamic sparsity of attention mechanism, RetrievalAttention proposes to use approximate nearest neighbor search (ANNS) indexes for KV vectors in CPU memory and retrieves the most relevant ones with vector search during generation. Unfortunately, we observe that the off-the-shelf ANNS indexes are often ineffective for such retrieval tasks due to the out-of-distribution (OOD) between query vectors and key vectors in attention mechanism. RetrievalAttention addresses the OOD challenge by designing an attention-aware vector search algorithm that can adapt to the distribution of query vectors. Our evaluation shows that RetrievalAttention only needs to access 1–3% of data while maintaining high model accuracy. This leads to significant reduction in the inference cost of long-context LLMs with much lower GPU memory footprint. In particular, RetrievalAttention only needs a single NVIDIA RTX4090 (24GB) for serving 128K tokens in LLMs with 8B parameters, which is capable of generating one token in 0.188 seconds.

1 Introduction

Recent transformer-based Large Language Models [1] have shown remarkable capabilities in processing long contexts. For instance, Gemini 1.5 Pro [2] has supported the context window of up to 10 million tokens. While this is promising for analyzing extensive data, supporting longer context windows also introduces challenges for inference efficiency due to the quadratic complexity of attention computation. To enhance efficiency, KV caching, a technique that retains past Key and Value vectors, has been widely adopted to prevent redundant computations. However, KV caching-based systems face two primary issues: (a) substantial GPU memory requirements particularly for long contexts, e.g., the Llama-2-7B model requires approximately 500GB per million tokens in FP16 format; and (b) inference latency increases linearly to the context size, primarily due to the time needed to access cached tokens — a common issue across various computing devices, including GPUs. Therefore, reducing token access and storage costs is vital for enhancing inference efficiency.

The solution lies in leveraging the dynamic sparsity inherent in the attention mechanism [3]. This refers to the phenomenon where each query vector significantly interacts with only a limited subset of key and value vectors, with the selection of these tokens varying dynamically based on individual queries. Prior work [4–9] has proposed various techniques to capitalize on this observation to improve

Preprint. Under review.

[1] Work during internship at Microsoft.

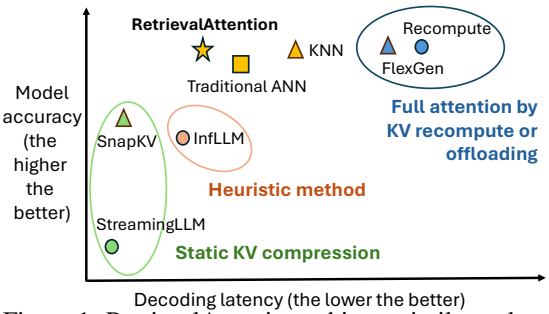


Figure 1: RetrievalAttention achieves similar task accuracy as full attention but exhibits extremely low decoding latency.

Prompt Length	128K	256K	512K	1M
Total Latency (s)	32.8	111	465	1,765
FFN (s)	7.6	15	31	70
Attention (s)	25.2	96	434	1,695
GPU Memory KV Cache (GB)	62.5	125	250	500

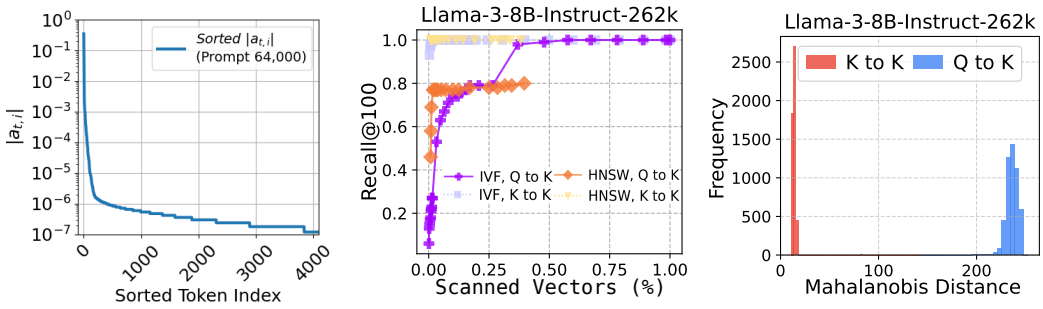
Table 1: Decoding latency (on one A100) and GPU memory required for KV cache of Llama-2-7B across different context lengths.

the efficiency of attention computation. However, most of these methods identify important tokens either statically [10, 11] or heuristically [6, 7, 5], leading to imprecise approximations that often result in a significant performance drop. We observe that the Approximate Nearest Neighbor Search (ANNS) index is particularly effective in this context. When using the inner product as the similarity measurement, ANNS precisely aligns with the attention mechanism, represented as $\mathbf{qK}^T$. It can directly identify the most critical key vectors with maximum inner product value, yielding a higher accuracy compared to previous static or heuristic methods (as illustrated in Figure 1).

Leveraging ANNS for attention mechanisms presents a unique challenge: the out-of-distribution (OOD) problem between query and key vectors. Most ANNS engines operate under the assumption that both query and key vectors are drawn from the same distribution. However, due to the different projection weights for query and key vectors in attention mechanism, this assumption often does not hold in this context. Unfortunately, the effectiveness of ANNS degrades significantly if the query vectors are OOD. In particular, our empirical analysis shows that maintaining an acceptable accuracy level for inference quality can require scanning 30% of all key vectors to identify the critical ones, which fails to exploit the inherent sparsity in attention mechanism. To the best of our knowledge, this is the first work that identifies the challenge of OOD in using ANNS for attention computation.

In this work, we present RetrievalAttention, an efficient method for accelerating long-context LLM generation. RetrievalAttention employs dynamic sparse attention during token generation, allowing the most critical tokens to emerge from the extensive context data. To address the challenge of OOD, RetrievalAttention proposes a vector index tailored for the attention mechanism, focusing on the distribution of queries rather than key similarities. This approach allows for traversal of only a small subset of key vectors (1% to 3%), effectively identifying the most relevant tokens to achieve accurate attention scores and inference results. In addition, RetrievalAttention reduces GPU memory consumption by retaining a small number of KV vectors in the GPU memory following static patterns (e.g., similar to StreamingLLM [10]) and offloading the majority of KV vectors to CPU memory for index construction. During token generation, RetrievalAttention efficiently retrieves critical tokens using vector indexes on the CPU and merges the partial attention results from both the CPU and GPU. This strategy enables RetrievalAttention to perform attention computation with reduced latency and minimal GPU memory footprint.

We evaluate the accuracy and efficiency of RetrievalAttention on both commodity GPUs (4090) and high-end GPUs (A100) on three long-context LLMs across various long-context benchmarks like ∞-Bench [12] and RULER [13]. For the 128K context on the 4090 GPU, RetrievalAttention achieves $4.9\times$ and $1.98\times$ decoding-latency reduction compared to the retrieval method based on exact KNN and traditional ANNS indexing respectively, while maintaining the same accuracy as full attention. To the best of our knowledge, RetrievalAttention is the first solution that supports running 8B-level models on a single 4090 GPU (24GB) with acceptable latency and almost no accuracy degradation.



(a) Attention score distribution. (b) ANNS index performance. (c) Different distribution.

Figure 2: (a) Attention is highly sparse in 64,000 tokens. (b) Off-the-shelf ANNS indexes perform poorly on Q to K searches but performs well on K to K searches. The Q and K are dumped from Llama-3-8B with a prompt length of 128,000 tokens. (c) Query vectors (Q) are distant from key vectors (K) while key vectors (K) themselves are close.

2 Background and Motivation

2.1 LLM and Attention Operation

In the generation of the t-th token, the attention mechanism computes the dot product between the query vector $\mathbf{q}_t \in \mathbb{R}^{1\times d}$ (where d is the hidden dimension) and the key vector of past tokens $\mathbf{k}_i \in \mathbb{R}^{1\times d}$ (for $i \leq t$). This product is scaled by $d^{-\frac{1}{2}}$ and normalized via a `Softmax` function to yield the attention score $a_{t,i}$. These scores then weight the values $\mathbf{v}_i$, resulting in the output $\mathbf{o}_t$.

$$z_i = \frac{\mathbf{q}_t \cdot \mathbf{k}_i^T}{\sqrt{d}}, \quad a_{t,i} = \frac{e^{z_i}}{\sum_{j=1..t} e^{z_j}}, \quad \mathbf{o}_t = \sum_{i=1..t} a_{t,i} \cdot \mathbf{v}_i \tag{1}$$

LLM inference contains two stages: the prefill phase and decoding phase. The prefill phase, which only happens once, computes the keys and values of the prompt with a time-complexity $O(n^2)$. In the decoding (token generation) phase, the newly generated token becomes the new query and computes attention scores with all past key vectors. One common optimization to avoid repetitive calculation is to cache past KV states in the GPU memory, thereby reducing the complexity to $O(n)$.

2.2 Expensive Long-Context Serving

Due to the quadratic time complexity of attention operation, serving long-sequence input incurs extremely high cost. Table 1 shows the inference latency without KV cache. When the prompt length reaches 1 million tokens, generating every token requires 1,765 seconds with over 96% of latency spent on attention operations. Although KV cache can reduce the decoding latency, it demands a huge amount of GPU memory for long contexts. As shown in Table 1, 500 GB memory is necessary for storing the KV cache when the context length reaches 1 million tokens, which is far beyond the GPU memory capacity of a single A100 GPU (80 GB). Offloading and reloading the KV cache between GPU and CPU memory is a potential solution but incurs excessive commutation overhead over PCIe [4], degrading the inference performance especially on commodity GPUs.

2.3 Dynamic and Sparse Attention

Despite the large size of the context, only a small proportion of tokens actually dominate the generation accuracy. Figure 2a shows the distribution of $|a_{t,i}|$ in Equation 1 for a query vector from Llama-2-7B with a prompt of 64,000 tokens. We observe that the top 500 tokens dominate the values of $|a_{t,i}|$, while the remaining tokens contribute approximately zero. A high attention score indicates that two vectors are close, as measured by the inner product. Therefore, the sparsity of attention scores means that the relevant keys to the query are very few. We measure the mean-square-error (MSE) of the attention output if we use the top-k $|a_{t,i}|$ as an approximation. We find that it only

needs 36 tokens to achieve a very low MSE ($< 10^{-6}$) of the full attention, showing a high sparsity ratio ($> 99.9\%$).

Moreover, we observe that as the LLM continues generating new tokens, critical key vectors change dynamically. This means that tokens considered important in previous queries may not be critical in subsequent queries, and vice versa. The dynamic sparsity shows a promising path to approximately compute attention with greatly reduced cost and without sacrificing the model accuracy. For each query, if we can accurately determine the relevant key-value vectors with higher importance, minimum GPU memory and a much lower time complexity can be achieved for attention computation.

2.4 Challenges of Off-the-shelf Vector Search

Finding the most similar vectors using ANNS indexes is a widely studied problem [14, 15], which semantically aligns with the goal of attention to find the nearest key vectors to each query vector in the inner product space. However, we find that naively applying off-the-shelf vector indexes fails to provide good performance because of the OOD issue between query (Q) and key vectors (K).

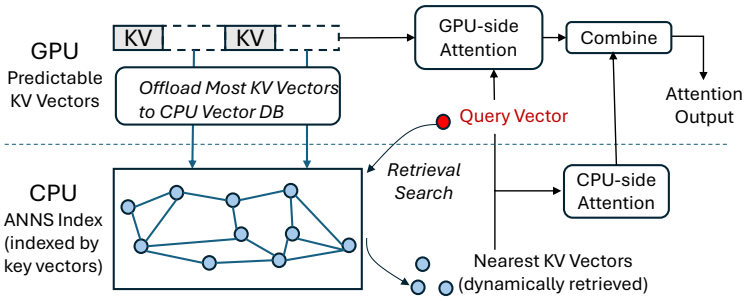
In conventional vector databases, the distribution of vectors between content and query is often well-aligned because they are derived from the same embedding model. However, naively using off-the-shelf vector indexes for attention computation suffers from an inherent distribution gap between queries and keys, which are projected by different model weights. Figure 2b (focus on Q to K for now) demonstrates the performance of widely-used vector indexes supported by Faiss [16] using a query vector to retrieve the most similar key vectors. It compares the percentage of keys scanned and the corresponding recall achieved (i.e., the overlapping ratio between the retrieved top-100 results and the ground truth). IVF [14] requires scanning $\sim 30\%$ data for a recall rate higher than 0.95, and HNSW[15] falls into a local optimum. The results show that traditional vector indexes require scanning a large number of vectors to achieve high recall, highlighting the challenge of performing efficient vector searches for attention.

Fundamentally, the difficulty is due to the OOD between query and key vectors. We quantify this using Mahanobis distance [17], which measures the distance from a vector to a distribution. We sample 5,000 vectors from Q and K respectively as the query set and compute the the Mahanobis distance from the query set to the remaining vectors in K. Figure 2c shows that the queries from Q are significantly distant from the K vectors (OOD) while K themselves are very close. Therefore, traditional index building based solely on the closeness between key vectors does not align with the attention mechanism, which requires to retrieve critical tokens as nearest neighbors from the query vectors' viewpoint. In contrast, Figure 2b shows that using sampled K as the queries (K to K) can easily achieve a high recall by only scanning 1-5% vectors because they are in the same distribution. Similarly, query vectors also follow the same distribution. For efficient vector search, the index must consider the OOD characteristic of the attention computation by design.

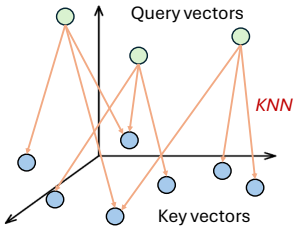
3 RetrievalAttention Design

In this work, we focus on the acceleration of token generation and assume the prefill of the long-context prompt is done in advance, which is widely supported by existing LLM service providers (e.g., context caching [18] or separation of prefill and decoding [19, 20]).

We propose RetrievalAttention that leverages attention-aware vector search to accurately approximate attention computation by CPU-GPU co-execution. Figure 3a shows the overall design of RetrievalAttention. Based on our observation in §2.3, We derive an approximated attention by selectively retrieving relevant key-value vectors while discarding those that are negligible(§3.1). To efficiently supports long context, we offload most KV vectors to the CPU memory, build vector indexes, and use attention-aware vector search to find critical tokens. (§3.2). To better exploit the GPU devices, we leverage the attention scores obtained in the prefill phase to select a proportion of KV cache that are consistently important during the decoding phase and persist them on GPU devices. RetrievalAttention computes partial attention with dynamically retrieved from CPU memory and persistent key-value vectors in GPU memory in parallel, and finally combine them together(§3.3).



(a) Overall design of RetrievalAttention.



(b) Key building procedure of OOD-aware index.

Figure 3: (a) RetrievalAttention offloads most KV tokens to vector databases in CPU, which are retrieved during the decoding phase to find the most relevant KV tokens with queries. (b) During the index construction, we link each query to its exact top-k nearest key vectors (KNN).

3.1 Approximated Attention

Based on the Equation 1, RetrievalAttention approximates the full attention output $\mathbf{o}_t$ by selectively utilizing the KV vectors associated with high attention scores (*i.e.*, $a_{t,i}$). Specifically, we define $\mathcal{I}_{t,\epsilon}$ as a subset of token indices for which the attention score surpasses ϵ. Consequently, a sparse attention mechanism, which only considers tokens located in $\mathcal{I}_{t,\epsilon}$, can be defined as follows:

$$\mathbf{o}_t = \sum_{i \in \mathcal{I}_{t,\epsilon}} a_{t,i} \cdot \mathbf{v}_i + \cancel{\sum_{i \notin \mathcal{I}_{t,\epsilon}} a_{t,i} \cdot \mathbf{v}_i} \approx \sum_{i \in \mathcal{I}_{t,\epsilon}} \tilde{a}_{t,i} \cdot \mathbf{v}_i \quad \text{where} \quad \tilde{a}_{t,i} = \frac{e^{z_i}}{\sum_{j \in \mathcal{I}_{t,\epsilon}} e^{z_j}} \tag{2}$$

Based on the above approximation, we build RetrievalAttention to only consider important key-value vectors (i.e., $\mathcal{I}_{t,\epsilon}$) which are retrieved by vector indexes.

3.2 Attention-aware Vector Search

For each pair of key and value vectors, we first decide whether to hold them in CPU or GPU memory (decision method is elaborated in §3.3). The KV vectors offloaded to CPU memory will be indexed by $\mathbf{k}_i \in \mathcal{R}^d$ and queried by $\mathbf{q}_t$ to find for the most relevant ones.

To accelerate the vector search during token generation, RetrievalAttention leverages the existing query vectors in the prefill phase to guide the index building for key vectors, efficiently mitigating the distribution gap. As shown in Figure 3b, RetrievalAttention explicitly establishes connections from the query vector to its nearest key vectors (i.e., exact k-nearest neighbors, or KNN). The KNN results can be efficiently computed via GPU, forming a mapping from query vector distribution to key vector distribution. Using this structure, the decoding query vector can firstly searches its nearest query vectors and then obtain the most relevant key vectors through the distribution mapping.

Therefore, the KNN connections from query vectors to key vectors serves as a bridge to reconcile their distribution differences. However, this structure still has imperfections in both memory overhead and search efficiency because we need to store and access query vectors besides key vectors. To address this problem, we leverage the projection technique from the state-of-the-art cross-modal ANNS index RoarGraph [21] to eliminate all query vectors. Specifically, we project the KNN connections into key vectors by linking key vectors that are connected to the same query vectors, which efficiently streamlines the search. This process connects key vectors that are perceived close from query vectors' perspective, allowing efficient index traversal for future query vectors.

Our evaluation shows that, by effectively modeling the proximity relationship between the query and key vectors, the vector database only requires scanning $1-3\%$ key vectors to reach a high recall, significantly reducing the index search latency by 74% compared with IVF indexes [14].

3.3 CPU-GPU Co-Execution

To exploit GPU parallelism and accelerate attention computation, RetrievalAttention decomposes the attention computation into two disjoint sets of KV cache vectors: the predictable ones on GPU and the dynamic ones on CPU, and then combine the partial attention outputs together.

We leverage the patterns observed in the prefill phase to predict KV vectors that are consistently activated during token generation. Similar to StreamingLLM [10], our current implementation uses fixed initial tokens and last sliding window of the context as the static pattern, and persist them in the GPU cache. RetrievalAttention can be adapted to utilize more complex static patterns [11, 22], achieving the best trade-off between low inference cost and high accuracy. To minimize data transfer over the slow PCIe interface, RetrievalAttention independently computes the attention results for the CPU and GPU components and then combines them, inspired by the FastAttention [23]. The detailed combination algorithm and overall execution flow of RetrievalAttention is illustrated in §B.

4 Evaluation

In this section, we compare the performance of RetrievalAttention in long-context LLM inference against full attention and other state-of-the-art methods. Through experiments, we mainly explore the following questions: (1) **How does RetrievalAttention affect the model's inference accuracy?** Specifically, we assess the generation accuracy of RetrievalAttention and other methods across various downstream tasks. (§4.2) (2) **Can RetrievalAttention efficiently reduce the token generation latency of long-context LLM inference?** We compare the end-to-end decoding latency of RetrievalAttention with that of other baselines. (§4.3).

4.1 Experimental Setup

Testbed, Models, and Configurations. We conduct experiments on a server equipped with one NVIDIA RTX 4090 GPU (24GB memory) and an Intel i9-10900X CPU with 20 cores and 128GB DRAM. The experiment results using NVIDIA A100 GPU can be found in §A.4. We implement RetrievalAttention on three state-of-the-art long-context LLMs, including Llama-3-8B-Instruct-262k[24], Yi-6B-200K[25], and Yi-9B-200K[26]. To show practical speedup of RetrievalAttention and ensure the CPU memory consumption in long contexts do not exceed the capacity of DRAM, we follow previous work [5] to run the benchmark in real-world single-batch scenarios.

Baselines. We compare RetrievalAttention with the following training-free baselines. (1) Full attention without KV cache as well as the version with KV cache using vLLM [27]. (2) StreamingLLM[10]: it retains initial tokens along with fixed-length recent tokens in the GPU memory and discards remaining tokens. (3) SnapKV[11]: it caches the active tokens observed from the last window of the prompts. (4) InfLLM[6]: it separates KV cache of continuous token sequences into blocks and select representative vectors for each block. In the decoding phase, the current query scans all representative vectors and retrieve top-k blocks with the highest similarity. We do not include Quest [5] for evaluation because its implementation [1] does not support the GQA model.

To better assess the effectiveness of our method, we introduce two additional baselines using traditional vector search methods from Faiss[16]. Specifically, Flat is an exact KNN method that performs a linear scan of all key-value vectors, whereas IVF indexes key vectors through clustering. By default, all indexing-based methods retrieve top-100 nearest key vectors to the current query vector.

Benchmarks. We adopt three representative long-context benchmarks for evaluation.

- ∞-Bench[12]: this benchmark consists of 7 tasks, including three retrieval tasks (PassKey retrieval, Number retrieval, KV retrieval) and four realistic tasks (code debugging, dialogue and multiple-choices questions). The average context length of ∞-Bench is over 100K tokens.
- RULER[13]: a comprehensive long-context benchmark consisting of 4 categories and 13 tasks, including retrieval, multi-hop tracing, aggregation, and QA tasks. It contains prompts of different lengths from 4K to 128K, allowing us to determine the actual context window size of models.
- Needle-in-a-haystack [28]: it challenges the models to accurately retrieve information (the "needle") hidden within a lengthy document (the "haystack").

[1] https://github.com/mit-han-lab/quest

Table 2: Performance (%) of different methods and models on ∞-Bench. The size of the static pattern is consistently 640 (128 initial tokens + 512 tokens in the local window). All indexing based methods including Flat, IVF and RetrievalAttention retrieve top-100 key vectors by default. In the relatively complicated task KV Retrieval, we include the results of retrieving top-2000 key vectors.

	Methods	**Tokens**	**Retr.N**	**Retr.P**	**Retr.KV**	**Code.D**	**Math.F**	**En.QA**	**En.MC**	**Avg.**
Llama-3-8B	FullAttention	128K	100.0	100.0	17.5	19.0	39.5	9.1	68.0	50.4
	StreamingLLM	640	5.0	5.0	0.5	18.5	39.5	5.9	66.5	20.1 (-30.3)
	SnapKV	2K	100.0	100.0	0.5	18.0	40.0	11.8	67.0	48.2 (-2.2)
	InfLLM	640+2K	100.0	100.0	0.5	20.5	48.0	7.0	37.0	44.7 (-5.7)
	Flat	640+100/2K	100.0	100.0	8.5/14.5	19.0	40.0	7.5	67.0	48.9 (-1.5) / 49.7 (-0.7)
	IVF	640+100/2K	94.0	100.0	9.5/14.0	19.0	40.0	7.8	67.0	48.2 (-2.2) / 48.8 (-1.6)
	RetrievalAttention	640+100/2K	100.0	100.0	9.0/14.0	19.0	40.0	7.5	67.0	**48.9 (-1.5) / 49.6 (-0.8)**
Yi-9B	FullAttention	128K	100.0	100.0	30.5	25.5	36.5	9.8	67.0	52.8
	StreamingLLM	640	5.0	5.0	0.5	23.5	30.5	6.3	69.5	20.0 (-32.8)
	SnapKV	2K	63.0	100.0	0.5	23.0	33.0	10.3	68.5	42.6 (-10.2)
	InfLLM	640+2K	100.0	100.0	0.5	20.5	43.0	9.4	44.0	45.3 (-7.5)
	Flat	640+100/2K	100.0	100.0	21.0/30.0	23.0	35.0	10.8	68.5	51.2 (-1.6) / 52.5 (-0.3)
	IVF	640+100/2K	99.0	100.0	19.5/29.5	23.0	35.0	10.7	69.0	50.9 (-1.9) / 52.3 (-0.5)
	RetrievalAttention	640+100/2K	99.5	100.0	20.0/30.0	23.0	35.0	9.5	68.5	**50.8 (-2.0) / 52.2(-0.6)**
Yi-6B	FullAttention	128K	98.0	100.0	3.5	31.0	11.0	19.2	55.5	45.5
	StreamingLLM	640	5.0	5.0	0.5	29.5	8.0	11.2	54.0	16.2 (-29.3)
	SnapKV	2K	39.0	100.0	0.0	30.5	8.5	17.1	55.0	35.7 (-9.8)
	InfLLM	640+2K	99.0	100.0	0.5	27.5	18.0	12.7	40.5	42.6 (-2.9)
	Flat	640+100/2K	98.5	100.0	2.5/3.0	30.5	16.0	17.7	54.5	45.7 (+0.2) / 45.7 (+0.2)
	IVF	640+100/2K	98.0	100.0	2.5/3.5	29.5	16.0	17.5	54.5	45.4 (-0.1) / 45.6 (+0.1)
	RetrievalAttention	640+100/2K	95.0	99.0	3.0/3.0	30.0	16.0	17.6	54.5	**45.0 (-0.5) / 45.0 (-0.5)**

4.2 Accuracy on Long Context Tasks

∞-Bench. As shown in Table 2, RetrievalAttention achieves comparable accuracy to the full attention, benefiting from its efficient dynamic retrieval of important tokens. Static methods, such as StreamingLLM and SnapKV, lack this capability and therefore achieve sub-optimal accuracy. During token generation, the critical tokens change dynamically according to the current query, invalidating the previously captured static patterns. Although InfLLM supports dynamic retrieval of relevant blocks, it achieves nearly zero accuracy in complex tasks (i.e., KV retrieval) due to low accuracy of representative vectors. Since RetrievalAttention can accurately identify the most relevant key vectors, it achieves the best accuracy in KV retrieval. Moreover, by retrieving more tokens (i.e., top-2000 shown in the column of Retr.KV) in KV retrieval, RetrievalAttention achieves nearly the same accuracy as full attention, which demonstrating the effectiveness of our method in complex and dynamic tasks.

It is worth noting that Flat and IVF need to scan 100% and 30% of the past key vectors to achieve the same task accuracy as RetrievalAttention. In contrast, RetrievalAttention only requires scan $1 - 3\%$ vectors, resulting in much lower decoding latency.

RULER. Table 3 demonstrates that models utilizing RetrievalAttention achieves nearly the same task accuracy as full attention in different context lengths. In contrast, other training-free methods experience an noticeable reduction in accuracy, particularly for longer context sizes like 128K, as they fail to capture dynamically changed important tokens.

Needle-in-a-haystack. As shown in Figure 4, RetrievalAttention can effectively focus on information at various positions across different context windows, ranging from 4K to 128K. In contrast, methods like StreamingLLM encounter difficulties when critical information lies beyond the range of the static patterns, whose results are shown in §A.2.

4.3 Latency Evaluation

As the context length increases, the decoding latency of full attention significantly increases due to its quadratic time comlexity. Enabling the KV cache (vLLM) incurs out-of-memory (OOM) issues due to limited GPU memory. The latency of StreamingLLM, SnapKV, and InfLLM remains relatively stable because of constant tokens involved in the attention computation but they suffer significant accuracy degradation as demonstrated in §4.2. Due to efficient attention-aware vector

search, RetrievalAttention achieves $4.9\times$ and $1.98\times$ latency reduction compared to Flat and IVF for the 128K context. The detailed latency breakdown can be seen in the Table 6 of §A.3.

Table 3: Performance (%) of different methods and models on RULER.

	Methods	Act. Tokens	Claimed	Effective	4K	8K	16K	32K	64K	128K	Avg.
Llama-3-8B	FullAttention	128K	262K	32K	93.13	90.49	89.27	85.11	82.51	78.74	86.54
	StreamingLLM	640	-	<4K	28.74	16.49	15.46	13.34	11.30	9.45	15.80 (-70.75)
	SnapKV	2K	-	4K	91.51	80.70	75.53	70.84	65.44	58.68	73.78 (-12.76)
	InfLLM	640+2K	-	4K	85.20	52.86	38.29	32.44	27.94	25.71	43.74 (-42.81)
	Flat	640+100	-	16K	92.71	87.93	87.01	84.97	80.99	74.34	84.66 (-1.89)
	IVF	640+100	-	16K	92.73	87.86	87.22	84.74	78.46	68.21	83.20 (-3.34)
	RetrievalAttention	640+100	-	16K	92.64	88.46	86.80	84.78	80.50	74.70	**84.70(-1.85)**
Yi-9B	FullAttention	128K	200K	8K	91.02	86.62	82.85	73.17	67.08	60.51	76.87
	StreamingLLM	640	-	<4K	27.96	16.03	14.89	9.48	10.95	12.05	15.23 (-61.65)
	SnapKV	2K	-	4K	90.39	75.59	64.48	48.70	39.28	32.97	58.57 (-18.30)
	InfLLM	640+2K	-	<4K	82.66	50.36	36.17	28.20	22.65	20.94	40.16 (-36.71)
	Flat	640+100	-	8K	91.09	87.71	84.42	74.58	66.16	59.50	77.24 (+0.37)
	IVF	640+100	-	8K	91.03	91.04	83.85	72.19	65.13	58.04	76.29 (-0.58)
	RetrievalAttention	640+100	-	8K	90.78	86.32	82.95	73.73	65.67	59.15	**76.43(-0.44)**
Yi-6B	FullAttention	128K	200K	<4K	84.52	77.77	69.12	61.64	58.36	55.77	67.86
	StreamingLLM	640	-	<4K	24.77	12.88	11.21	7.33	8.76	9.86	12.47 (-55.40)
	SnapKV	2K	-	<4K	80.94	59.55	45.36	36.11	33.43	29.53	47.49 (-20.38)
	InfLLM	640+2K	-	<4K	76.42	44.38	34.11	27.11	25.28	25.33	38.77 (-29.09)
	Flat	640+100	-	<4K	83.69	77.26	67.28	60.58	57.27	50.63	66.12 (-1.75)
	IVF	640+100	-	<4K	83.25	76.90	67.00	58.94	55.99	50.31	63.33 (-2.53)
	RetrievalAttention	640+100	-	<4K	83.01	76.56	67.49	59.46	57.20	51.44	**65.86(-2.00)**

5 Related Works

To accelerate the long-context LLM inference, some works [29, 30, 10, 31, 32, 11] attempt to compress the size of the KV cache by leveraging the sparsity of attention. However, these methods often suffer from significant model accuracy drops due to the dynamic nature of attention sparsity.

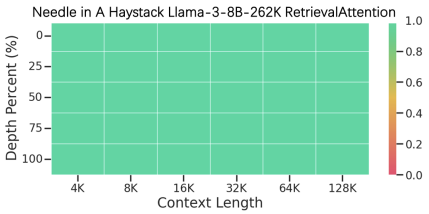


Figure 4: Performance of RetrievalAttention in Needle-in-a-haystack.

Methods	4K	8K	16K	32K	64K	128K
Full (without cache)	0.527	1.167	2.672	6.214	15.263	43.927
vLLM	OOM	OOM	OOM	OOM	OOM	OOM
StreamingLLM	0.029	0.030	0.029	0.030	0.030	0.029
SnapKV	0.029	0.028	0.028	0.029	0.029	0.028
InfLLM	0.058	0.063	0.063	0.065	0.067	0.069
Flat	0.140	0.178	0.226	0.328	0.522	0.922
IVF	0.128	0.140	0.162	0.201	0.253	0.373
RetrievalAttention	0.137	0.144	0.156	0.162	0.169	0.188

Table 4: Per-token generation latency (s) as context length varies from 4K to 128K on Llama-3-8B.

FlexGen [4] and Lamina [33] offload the KV cache to CPU memory, but they struggle with slow and costly full-attention computation. By identifying the dynamic nature of important KV vectors for different queries, recent work chooses to retain all of the KV cache and dynamically attend to different parts of KV vectors based on the current query. Quest [5] partitions the KV cache into blocks and selects a representative key vector for each block. For a given query, it scans all representative key vectors and attends top-k blocks with the highest attention scores. InfLLM [6] adopts a similar strategy as Quest but offloads most KV cache blocks to the CPU memory to support longer contexts. Due to block-based organization and retrieval, the accuracy of representative vectors significantly impacts the effectiveness of those methods for obtaining important tokens. SparQ [7], InfiniGen [8], and LoKi [9] approximate the most relevant top-k keys corresponding to a given query by reducing the head dimension. RetrievalAttention instead organizes the KV cache using ANNS indexes, allowing the retrieval of important tokens with high recalls and low cost. The concurrent work MagicPiG [34] and PQCache [35] employ locality-sensitive-hashing (LSH) and PQ centroids respectively to retrieve critical tokens. However, they fail to address the OOD issue in attention, necessitating retrieving a large portion of KV cache (e.g., 20%) to maintain the model accuracy.

6 Conclusion

We propose RetrievalAttention, a method that offloads most KV vectors to CPU memory and leverages vector search for dynamic sparse attention to minimize inference cost. RetrievalAttention identifies the different distributions of the query and key vectors and employs an attention-aware approach to efficiently find critical tokens for model generation. Experimental results demonstrate that RetrievalAttention effectively achieves $4.9\times$ and $1.98\times$ decoding speedup than exact KNN and traditional ANNS methods, on a single RTX4090 GPU for a context of 128K tokens. RetrievalAttention is the first system that supports running 8B-level LLMs with 128K tokens on a single 4090 (24GB) GPU with an acceptable latency cost and without compromising model accuracy.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 5998–6008, 2017.

[2] Gemini Team. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context, 2024.

[3] Yichuan Deng, Zhao Song, and Chiwun Yang. Attention is naturally sparse with gaussian distributed input, 2024.

[4] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: high-throughput generative inference of large language models with a single gpu. In Proceedings of the 40th International Conference on Machine Learning, ICML'23. JMLR.org, 2023.

[5] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In Forty-first International Conference on Machine Learning, 2024.

[6] Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, Song Han, and Maosong Sun. Infllm: Unveiling the intrinsic capacity of llms for understanding extremely long sequences with training-free memory. ArXiv preprint, abs/2402.04617, 2024.

[7] Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. Sparq attention: Bandwidth-efficient llm inference. In Forty-first International Conference on Machine Learning, 2024.

[8] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. InfiniGen: Efficient generative inference of large language models with dynamic KV cache management. In 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24), pages 155–172, Santa Clara, CA, 2024. USENIX Association.

[9] Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank keys for efficient sparse attention. ArXiv preprint, abs/2406.02542, 2024.

[10] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In The Twelfth International Conference on Learning Representations, 2024.

[11] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. ArXiv preprint, abs/2404.14469, 2024.

[12] Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Hao, Xu Han, Zhen Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. ∞Bench: Extending long context evaluation beyond 100K tokens. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 15262–15277, Bangkok, Thailand, 2024. Association for Computational Linguistics.

[13] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? ArXiv preprint, abs/2404.06654, 2024.

[14] Sivic and Zisserman. Video google: A text retrieval approach to object matching in videos. In Proceedings ninth IEEE international conference on computer vision, pages 1470–1477. IEEE, 2003.

[15] Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE transactions on pattern analysis and machine intelligence, 42(4):824–836, 2018.

[16] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. 2024.

[17] Prasanta Chandra Mahalanobis. On the generalized distance in statistics. Sankhyā: The Indian Journal of Statistics, Series A (2008-), 80:S1–S7, 2018.

[18] Google Cloud. Context caching overview. `https://cloud.google.com/vertex-ai/generative-ai/docs/context-cache/context-cache-overview`, 2024. Accessed: 2024-07-01.

[19] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA), pages 118–132. IEEE, 2024.

[20] Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Kimi's kvcache-centric architecture for llm serving. ArXiv preprint, abs/2407.00079, 2024.

[21] Meng Chen, Kai Zhang, Zhenying He, Yinan Jing, and X. Sean Wang. Roargraph: A projected bipartite graph for efficient cross-modal approximate nearest neighbor search. Proc. VLDB Endow., 17(11):2735–2749, aug 2024.

[22] 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, abs/2407.02490, 2024.

[23] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In Advances in Neural Information Processing Systems, 2022.

[24] Gradient AI. Llama-3-8b-instruct-262k. `https://huggingface.co/gradientai/Llama-3-8B-Instruct-262k`, 2024. Accessed: 2024-07-01.

[25] 01-ai. Yi-6b-200k. `https://huggingface.co/01-ai/Yi-6B-200K`, 2024. Accessed: 2024-07-01.

[26] 01-ai. Yi-9b-200k. `https://huggingface.co/01-ai/Yi-9B-200K`, 2024. Accessed: 2024-07-01.

[27] 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.

[28] Greg Kamradt. Needle in a haystack - pressure testing llms. https://github.com/gkamradt/LLMTest_NeedleInAHaystack, 2023. Accessed: 2024-08-12.

[29] 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 Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, 2023.

[30] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. Advances in Neural Information Processing Systems, 36, 2024.

[31] Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. LM-infinite: Zero-shot extreme length generalization for large language models. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 3991–4008, Mexico City, Mexico, 2024. Association for Computational Linguistics.

[32] Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In The Twelfth International Conference on Learning Representations, 2024.

[33] Shaoyuan Chen, Yutong Lin, Mingxing Zhang, and Yongwei Wu. Efficient and economic large language model inference with attention offloading. ArXiv preprint, abs/2405.01814, 2024.

[34] Zhuoming Chen. Magicpig: sparse inference engine for llm. https://github.com/Infini-AI-Lab/MagicPiG/, 2024. Accessed: 2024-08-01.

[35] Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. Pqcache: Product quantization-based kvcache for long context llm inference. arXiv preprint arXiv:2407.12820, 2024.

[36] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In The 2023 Conference on Empirical Methods in Natural Language Processing, 2023.

[37] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. ArXiv preprint, abs/1904.10509, 2019.

[38] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. ArXiv preprint, abs/2004.05150, 2020.

[39] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020.

[40] Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. ETC: Encoding long and structured inputs in transformers. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 268–284, Online, 2020. Association for Computational Linguistics.

[41] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020.

[42] Amanda Bertsch, Uri Alon, Graham Neubig, and Matthew Gormley. Unlimiformer: Long-range transformers with unlimited length input. Advances in Neural Information Processing Systems, 36, 2024.

[43] Yuzhen Mao, Martin Ester, and Ke Li. Iceformer: Accelerated inference with long-sequence transformers on CPUs. In The Twelfth International Conference on Learning Representations, 2024.

[44] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24), pages 193–210, Santa Clara, CA, 2024. USENIX Association.

[45] Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Leon Song, Samyam Rajbhandari, and Yuxiong He. Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models. ArXiv preprint, abs/2309.14509, 2023.

[46] Hao Liu, Matei Zaharia, and Pieter Abbeel. Ringattention with blockwise transformers for near-infinite context. In The Twelfth International Conference on Learning Representations, 2024.

A Additional Experimental Details and Results

A.1 Model Architecture

Table 5 compares the architecture differences of the three models used in our experimental evaluation. All models supports the grouped query attention (GQA), in which multiple query heads share one KV head. Among them, the Yi-9B model has more transformer layers, while the Llama-3-8B model has more KV heads.

Table 5: Architecture overview of LLMs.

Model	Total Layer	Query Head	KV Head
Yi-6B	32	32	4
Yi-9B	48	32	4
Llama-3-8B	32	32	8

A.2 Additional Results on Needle-in-a-haystack

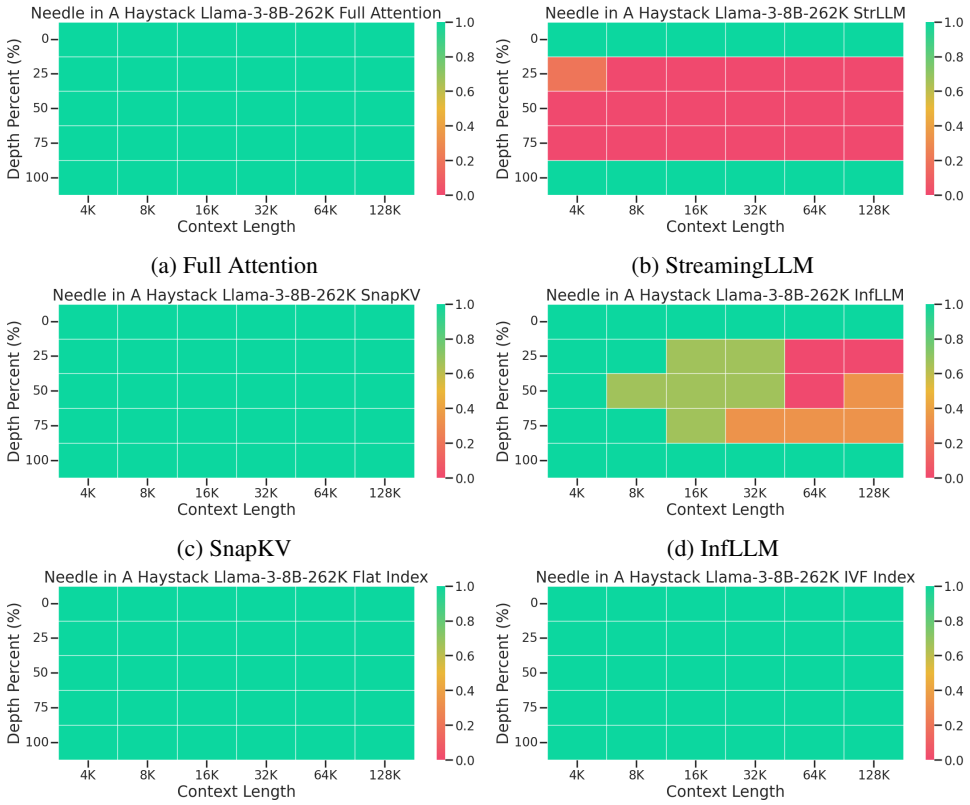
Figure 5 shows the results of other methods on Needle-in-a-haystack benchmark. StreamingLLM can only find the correct answer when the needle's position is within the static pattern. InfLLM maintains high performance with shorter context lengths. However, as the length increases, its performance shows a significant decline. Although SnapKV, Flat, and IVF perform well on this benchmark, we have analyzed their disadvantages in accuracy and latency in the previous evaluation.

Table 6: Latency breakdown of retrieval attention-based algorithm on Llama-3-8B.

Methods	Retrieval	Attention	Others	Total
Flat	0.798	0.083	0.041	0.922
IVF	0.250	0.084	0.039	0.373
RetrievalAttention	0.064	0.081	0.043	0.188

A.3 Latency Breakdown

Table 6 presents the breakdown of end-to-end latency for different retrieval attention-based algorithms on a single RTX 4090 server under the 128K context length. Since RetrievalAttention effectively addresses the issue of out-of-distribution, it only requires 34.0% of the time for vector search. In contrast, Flat and IVF spends 86.6% and 67.0% of time, respectively. This is because RetrievalAttention scans less data, avoiding memory bandwidth contention when multiple heads are performing parallel retrieval in the CPU side. This advantage becomes more pronounced with longer context lengths.



(a) Full Attention (b) StreamingLLM

(c) SnapKV (d) InfLLM

(e) FLAT (f) IVF

Figure 5: Performance of different algorithms and models on Needle-in-a-haystack. The size of the static pattern is consistently 640 (128 initial tokens + 512 tokens in the local window).

A.4 Decoding Latency on A100

We test the generality of RetrievalAttention by measuring its performance on a server with one A100 (80GB) and one AMD EPYC CPU with 24 cores and 220GB DRAM. We show the token-generation latency of different methods on three models in Table 7. Since the KV cache of full attention is disabled, all prompt tokens need to be recalculated during the decoding, incurring a very high decoding latency. By enabling the KV cache with the PageAttention optimization in vLLM, the decoding latency is significantly reduced. However, vLLM suffers from OOM issue with the increase of context length, which we elaborate further later. Other KV cache dropping or block retrieval methods including StreamingLLM, SnapKV, and InfLLM achieve faster decoding speed, but this is at the expense of a significant drop in model accuracy. In contrast, RetrievalAttention does not compromise generation accuracy while achieving much lower decoding latency than IVF and Flat because of the efficient mitigation of out-of-distribution problem.

We also evaluate how the decoding latency changes when the context length varies from 100K to 1M tokens on Llama-3-8B model and the results can be found in Table 8. To make sure there is enough CPU memory to hold the KV cache and indexes especially in the 1M context scenario, we use a powerful machine equipped with an AMD EPYC 7V12 CPU with 48 cores and 1.72 TB of memory. The machine is also equipped with the same 80GB A100 GPU. The decoding latency of full attention with KV state re-computation increases quadratically with the context size. With the KV cache enabled in the the GPU memory, vLLM starts triggering OOM issue when the context size is larger than 200K. Static KV dropping methods such as StreamingLLM have no latency increase due to the constant KV cache involved for attention computation. Different from Flat and IVF whose latency numbers are sensitive to context size, RetrievalAttention only has a minor latency increase (8%) when the context size increases 10× from 100K to 1M.

Table 7: Per-token generation latency (s) of 128K context-length on A100.

Methods	Yi-6B	Yi-9B	Llama-3-8B
Full(without cache)	31.61	47.51	33.38
vLLM	0.030	0.044	0.033
StreamingLLM	0.032	0.047	0.031
SnapKV	0.033	0.05	0.033
InfLLM	0.069	0.11	0.068
Flat	0.541	0.802	0.564
IVF	0.309	0.468	0.345
RetrievalAttention	0.150	0.227	0.155

Table 8: Per-token generation latency (s) as context length varies from 100K to 1M.

Methods	100K	200K	500K	1M
Full (without cache)	25.47	83.03	457	1740
vLLM	0.029	0.046	OOM	OOM
StreamingLLM	0.034	0.035	0.032	0.035
SnapKV	0.035	0.035	0.034	0.034
InfLLM	0.082	0.079	0.082	0.084
Flat	0.489	0.871	1.92	3.69
IVF	0.308	0.476	1.032	1.889
RetrievalAttention	0.159	0.167	0.170	0.172

B RetrievalAttention Algorithm

B.1 Formula of Combining Attention Results from the CPU and GPU Side

RetrievalAttention partitions the KV vectors for attention into two disjoint sets: predictable ones on GPU (denoted as $\mathcal{W}$) and dynamically retrieved ones on CPU (denoted as Ω).

$$\mathcal{I}_{t,\epsilon} = \mathcal{W} \cup \Omega \tag{3}$$

Attention operation is applied to the two sets of KV vectors separately on CPU and GPU, generating two partial attention outputs (denoted as $\mathbf{o}_{\mathcal{W}}$ and $\mathbf{o}_{\Omega}$, respectively). To guarantee the approximated attention output equals to the attention computation on $\mathcal{I}_{t,\epsilon}$, RetrievalAttention uses a similar idea of FlashAttention [23] to combine $\mathbf{o}_{\mathcal{W}}$ and $\mathbf{o}_{\Omega}$ in the following equations:

$$\begin{aligned}
\mathbf{o}_{\mathcal{W}} &= \text{Attn}(\mathbf{q}_t, \mathbf{K}[\mathcal{W},:], \mathbf{V}[\mathcal{W},:]) \\
&= \frac{\sum_{i\in\mathcal{W}} e^{z_i - \tilde{z}_1} \cdot v_i}{\sum_{i\in\mathcal{W}} e^{z_i - \tilde{z}_1}} \\
\mathbf{o}_{\Omega} &= \text{Attn}(\mathbf{q}_t, \mathbf{K}[\Omega,:], \mathbf{V}[\Omega,:]) \\
&= \frac{\sum_{i\in\Omega} e^{z_i - \tilde{z}_2} \cdot v_i}{\sum_{i\in\Omega} e^{z_i - \tilde{z}_2}} \\
\mathbf{o}_t &= \gamma_1 \cdot \mathbf{o}_{\mathcal{W}} + \gamma_2 \cdot \mathbf{o}_{\Omega}
\end{aligned} \tag{4}$$

where $\tilde{z}_1 = \max_{i\in\mathcal{W}} z_i$ and $\tilde{z}_2 = \max_{i\in\Omega} z_i$ are the local maximum dot products in set $\mathcal{W}$ and Ω respectively. And γ_1 and γ_2 are re-scaling factors to guarantee the attention output is the same as that on $\mathcal{I}_{t,\epsilon}$, which are defined as follows:

$$\begin{aligned}
\gamma_1 &= \frac{e^{\tilde{z}_1 - \tilde{z}} \cdot \sum_{i\in\mathcal{W}} e^{z_i - \tilde{z}_1}}{\sum_{i\in\mathcal{I}_{t,\epsilon}} e^{z_i - \tilde{z}}} \\
\gamma_2 &= \frac{e^{\tilde{z}_2 - \tilde{z}} \cdot \sum_{i\in\Omega} e^{z_i - \tilde{z}_2}}{\sum_{i\in\mathcal{I}_{t,\epsilon}} e^{z_i - \tilde{z}}}
\end{aligned} \tag{5}$$

B.2 Overall Execution Flow

Algorithm 1 summarizes the above design of RetrievalAttention and elaborate the procedure in an algorithm. At the beginning of each token generation, RetrievalAttention predicts active KV vectors and move them to GPU memory, and compute partial attention using the FlashAttention [23] kernel (1 - 6). In parallel with GPU computation, RetrievalAttention leverages the specially designed vector database to find the most relevant KV vectors to compute attention on CPU (7 - 8). Finally, RetrievalAttention combines the partial attention outputs on GPU and CPU using 4 and gets the approximated attention output (9).

Algorithm 1: RetrievalAttention

Input: Query vector $\mathbf{q}_t \in \mathcal{R}^{1\times d}$
Data: KV Cache in GPU $\mathbf{K}_{\mathcal{W}}, \mathbf{V}_{\mathcal{W}} \in \mathcal{R}^{|\mathcal{W}|\times d}$
Data: CPU-based Vector Database $\mathcal{H}$
Output: Attention output $\mathbf{o}_t \in \mathcal{R}^{1\times d}$

```
// Find the predictable KV vectors
```
$\mathcal{W}' \leftarrow$ PredictActiveTokens(...);
for $\{i|i \in \mathcal{H} \cup \mathcal{W}'\}$ **do**
  $\mathcal{H}$.remove(i); $\mathcal{W}$.insert(i); `// move to GPU`
for $\{i|i \notin \mathcal{W}' \wedge i \in \mathcal{H}\}$ **do**
  $\mathcal{W}$.remove(i); $\mathcal{H}$.insert(i); `// move to CPU`

```
// Attention on GPU
```
$\mathbf{o}_{\mathcal{W}} \leftarrow$ FlashAttention($\mathbf{q}_t, \mathbf{K}_{\mathcal{W}}, \mathbf{V}_{\mathcal{W}}$)

```
// Attention on CPU
// Search in vector database to retrieve most relevant KV vectors
```
$\Omega \leftarrow$ VectorSearch($\mathbf{q}_t$);
$o_{\Omega} \leftarrow$ AttentionCPU(Ω); `// Combine partial attention outputs`
$\mathbf{o}_t = \gamma_1 \cdot \mathbf{o}_{\mathcal{W}} + \gamma_2 \cdot \mathbf{o}_{\Omega}$; `// Equation 4,5`

C Implementation

RetrievalAttention builds one individual vector index for the KV cache in one attention head. RetrievalAttention did several optimizations to optimize the prompt prefill, accelerate the vector search, and reduce the CPU memory usage.

Optimization for the Prefill Phase. During the prefill phase, the full attention computation of the long context is required for generating the output vector for the next level of the LLM. Simultaneously, we move the KV vectors to the CPU side for the ANNS index building. To accelerate the overall prefill process, we overlap the cache movement to the CPU with the full attention computation on the GPU in a pipeline manner. To minimize peak GPU memory usage during the prefill phase, attention computation is performed sequentially across multiple attention heads. This approach only slightly impacts the attention computation speed, as longer prompts can fully leverage GPU parallelism with FlashAttention.

Multi-head Parallelism on the CPU side. To speed up the dynamic sparse attention computation on the CPU, we exploit the multi-thread parallelism in vector databases by leveraging the multi-core ability of modern CPU architecture. Specifically, since the computation of different attention heads is independent, we launch multiple threads for parallel searching across different vector indexes to reduce the overall latency on the CPU side. For grouped query attention (GQA) [36], although multiple query heads could share the same key-value vectors, we observe that the query vectors from different query heads in the same group exhibit different vector distributions. Therefore, we build one vector index for each query head to leverage the specific query distribution of each head.

Minimize the CPU Memory Usage. To reduce CPU memory consumption, the indexes in the same attention group share one copy of KV vectors by only storing the pointers to KV vectors in each index. In the future, we plan to utilize scalar quantization to further compress the KV vectors, implementing an 8-bit representation in place of the original FP32 format. This compression is promising to reduce memory usage while preserving computational efficiency. Importantly, our initial results demonstrate that this quantization approach does not compromise the accuracy of the model outcomes, maintaining performance equivalent to the full-precision representation.

D Additional Related Work

Sparse Transformers. Since the quadratic complexity of attention has become the bottleneck of LLM efficiency for long context applications, numerous works have studied to design sparse

transformers to reduce the computational and memory complexity of the self-attention mechanism. Some works restrict the attention computation to predefined patterns including sliding windows [37], dilated windows [38], or mixture of different patterns [39, 40]. Some approaches use cluster-based sparsity based on hash value [41] or KNN algorithms [42, 43].These solutions either require pre-training a model from scratch or target limited scenarios like CPU-only, which do not work for our target to out-of-box usage of LLMs on the GPU-CPU architecture. Although some approaches [6, 7] exploit the dynamic sparse nature of LLMs, they often use some estimation using low-rank hidden states or post-statistical approaches, which incurs high overhead but with low accuracy. Moreover, all these approaches have to maintain full KV vectors on GPU with only accelerated inference by reduced memory movement, which does not solve the challenge of commodity GPUs with limited GPU memory.

Additionally, some approaches accelerate the inference by employing dynamically sparse attention patterns [22], separating the prefill and decoding stages [44, 20], and utilizing sequence parallelism [45, 46]. These methods are orthogonal to ours and can be in conjunction with our approach.