RetroInfer: A Vector Storage Engine for Scalable Long-Context LLM Inference

Yaoqi Chen[1,2*], Jinkai Zhang[3,2*], Baotong Lu[2†], Qianxi Zhang[2], Chengruidong Zhang[2], Jing Liu[2], Jingjia Luo[4,2*], Di Liu[5], Huiqiang Jiang[2], Qi Chen[2], Bailu Ding[2], Xiao Yan[3,6], Jiawei Jiang[3], Chen Chen[5], Mingxing Zhang[4], Cheng Li[1,7], Yuqing Yang[2], Fan Yang[2], Mao Yang[2]
[1]University of Science and Technology of China [2]Microsoft Research [3]Wuhan University
[4]Tsinghua University [5]Shanghai Jiao Tong University [6]Institute for Math & AI, Wuhan
[7]Institute of Artificial Intelligence, Hefei Comprehensive National Science Center
[2]{baotonglu, qianxi.zhang, chengzhang, jingliu3, hjiang, cheqi, Yuqing.Yang, fanyang, maoyang}@microsoft.com
[1]yaoqi_chen@mail.ustc.edu.cn, chengli7@ustc.edu.cn [3]{zhangjinkai, yanxiaosunny, jiawei.jiang}@whu.edu.cn
[4]{luojj22, zhang_mingxing}@mail.tsinghua.edu.cn [5]{liu-di, chen-chen}@sjtu.edu.cn [2]bailuding@gmail.com

ABSTRACT

Recent large language models (LLMs) are rapidly extending their context windows, yet inference throughput lags due to increasing GPU memory and bandwidth demands. This is because the key-value (KV) cache, an intermediate structure storing token representations, grows linearly with context length and requires an iterative linear scan for attention computation. A promising direction to accelerate long-context inference is to exploit attention's inherent sparsity by offloading the KV cache to CPU memory and retrieving only a small subset of tokens important to the current generation step. However, prior sparse attention approaches struggle to balance accuracy and retrieval cost due to varying sparsity patterns and inefficient GPU-CPU memory management.

We present RetroInfer, a vector storage engine that realizes a sparsity-based KV cache for long-context inference. RetroInfer introduces an Attention-aWare VEctor index (*wave index*), which fundamentally improves the tradeoff between attention accuracy and retrieval cost through tripartite attention approximation, accuracy-bound attention estimation, and segmented clustering. We also design the *wave buffer*, a GPU-CPU buffer manager that assigns computation and manages data across heterogeneous hardware. We evaluate RetroInfer across a range of models and workloads, demonstrating up to 4.4× decoding throughput over full attention at 120K context and up to 12.2× over sparse attention baselines at 1 million tokens—all while preserving full-attention-level accuracy.

PVLDB Reference Format:
Yaoqi Chen, Jinkai Zhang, Baotong Lu, Qianxi Zhang, Chengruidong Zhang, Jing Liu, Jingjia Luo, Di Liu, Huiqiang Jiang, Qi Chen, Bailu Ding, Xiao Yan, Jiawei Jiang, Chen Chen, Mingxing Zhang, Cheng Li, Yuqing Yang, Fan Yang, Mao Yang. RetroInfer: A Vector Storage Engine for Scalable Long-Context LLM Inference. PVLDB, 19(5): 1016-1031, 2026.
doi:10.14778/3796195.3796212

*Work performed during the internship while at Microsoft.
†Corresponding author: Baotong Lu.
This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 19, No. 5 ISSN 2150-8097.
doi:10.14778/3796195.3796212

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/microsoft/RetrievalAttention.

1 INTRODUCTION

Transformer-based LLMs that rely on attention mechanisms [91] have experienced rapid adoption and enabled a range of new applications. Recently, their context windows have expanded dramatically to support use cases like multi-turn conversations [27], code and data analysis [8, 48], and reasoning [98]. Many popular LLMs now support context windows of 128K tokens [7, 70], and leading models such as Gemini Pro [31] and Llama 4 [62] push this further to 10 million tokens.

Scaling inference throughput with increasing context length presents significant challenges to GPU resources. During inference, the memory consumption of an LLM's key-value (KV) cache [73] – a core component for accelerating attention – grows linearly with the sequence length [43], straining memory capacity. For instance, serving a single 1M-token request with Llama3-8B [32] requires up to 125GB of memory. In addition, generating a new token requires iterating over all vectors in the KV cache, resulting in excessive memory access that quickly drains GPU bandwidth. Even with multiple GPUs, which offer more aggregate memory, large models with large KV caches continue to stress both capacity and bandwidth.

A promising direction is to exploit the inherent sparsity of attention to build a sparsity-based KV cache [16, 17, 22]. Recent systems [15, 53, 89] show that a small subset of tokens dominates attention output for a given query. Using only important tokens for attention calculation significantly reduces memory access and enables offloading the KV cache to high-capacity, slower CPU memory. However, identifying important tokens remains challenging.

Vector index [25, 39, 57, 59, 79, 84, 95, 119], traditionally used for approximate nearest neighbor search (ANNS), has shown promise in identifying these important tokens [23, 53, 115]. ANNS seeks the most similar vectors to a given query using inner product, which closely aligns with how attention computes relevance (Section 2).

Yet, sparsity-based KV cache systems face a fundamental tradeoff between attention accuracy and retrieval cost. Existing vector indexes fall short due to the variable nature of attention sparsity. Important tokens differ across layers, attention heads, decoding

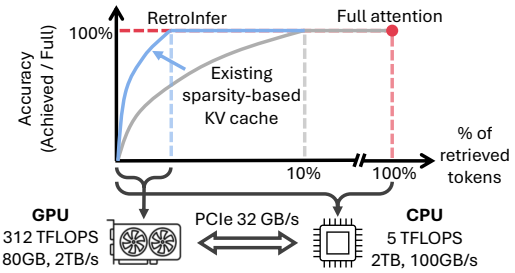

Figure 1: RetroInfer raises the trade-off ceiling between accuracy and retrieval cost, and manages data across hardware.

steps, and input tasks [9, 15]; thus, standard indexes struggle to retrieve a small number of tokens while maintaining high accuracy. As shown in Figure 1, current systems reduce data volume but still incur high retrieval costs (e.g., 10%) to preserve accuracy. We argue that a holistic, attention-aware approach for KV cache is needed to support long-context inference effectively.

We present RetroInfer, a vector storage engine that holistically addresses the challenges of sparsity-based KV cache systems. Central to RetroInfer are two components: an *Attention-aWare VEctor index* (wave index) and an *accuracy-agnostic GPU-CPU buffer manager* (wave buffer). Our design follows two principles: (1) be attention-aware, and (2) separate the concern of attention accuracy from the system efficiency.

Wave index is a new vector index design that improves the trade-off between attention accuracy and retrieval cost. Drawing inspiration from the classic vector indexes [85], wave index introduces three novel attention-aware techniques. First, a *tripartite attention approximation* logically partitions tokens into steady, retrieval, and estimation zones; token importance decreases across these zones. The steady zone computes precise attention on a small number of consistently important tokens, the retrieval zone uses index-guided selection to compute attention precisely, and the estimation zone approximates attention for the remaining tokens based on the index structure. As we move from steady to estimation, attention becomes more approximate but cheaper to compute, reducing sensitivity to the number of tokens retrieved and enabling control over compute cost without sacrificing accuracy.

Second, an *accuracy-bound attention estimation mechanism* approximates the contribution of less important but non-negligible tokens with guaranteed accuracy bounds in the estimation zone. Third, a *segmented clustering* redesigns traditional vector index clustering for parallelism and low overhead, reducing both the index construction and update cost.

With wave index, the key remaining challenge is to distribute computation – index traversal, construction, and attention calculation – and to coordinate data movement between CPU and GPU memory. The inherent ratio of important tokens is not low enough to fully hide PCIe transfer latency, which can stall GPU computation. The asymmetry between CPU and GPU leads to mismatches between computation type and available resources. As shown in Figure 1, the GPU provides high compute throughput but has limited memory capacity, while the CPU offers much larger memory but slower floating-point performance and memory access. The CPU is also better suited for a broader range of computation. Great care must be taken to fully utilize available resources, avoid GPU stalls, control software overhead, and reduce PCIe bandwidth contention.

We design the wave buffer to address these challenges; it acts as the metadata and control plane, similar to a buffer manager in a database system [45, 46, 126]. We observe that access to important tokens exhibits strong temporal locality, so the wave buffer incorporates a GPU-side block cache of the KV cache to further reducing PCIe pressure. Attention calculation and block cache access are performed synchronously on the GPU, while block cache replacement is handled asynchronously. The wave buffer overlaps computation, reduces overhead, and sustains overall inference efficiency. The wave buffer is designed to be accuracy-agnostic, focusing solely on hardware efficiency without affecting attention accuracy; it leverages classic database techniques such as caching and pipelining.

We evaluate RetroInfer on four popular models across a range of tasks from advanced long-context benchmarks (e.g., RULER [38]) with varying context lengths and batch sizes. We also evaluate RetroInfer on two reasoning models that generate long outputs. Our experiments show significant improvements in inference efficiency while maintaining the accuracy of full attention. For the same retrieval budget (i.e., the number of KVs retrieved), RetroInfer outperforms existing sparsity-based baselines by 1.40–46.47% in task accuracy and is the only system that achieves accuracy comparable to full attention. RetroInfer also significantly improves inference throughput by supporting large batch sizes and context lengths. Specifically, RetroInfer achieves up to 4.4× throughput over full attention when the context length fits in GPU memory, and up to 12.2× throughput over other sparse-attention systems when the KV cache is extended to CPU memory. For end-to-end comparison, RetroInfer achieves 2.2×–3.3× throughput compared to vLLM. We make the following contributions:

- We present a novel vector index design – wave index – that improves the trade-off between attention accuracy and retrieval cost, enabling efficient and accurate long-context inference. Wave index introduces three novel techniques: tripartite attention approximation, accuracy-bound attention estimation, and segmented clustering.
- We introduce wave buffer, an accuracy-agnostic GPU-CPU buffer manager that efficiently manages data movement and computation across heterogeneous hardware.
- We design and implement RetroInfer, a long-context inference system that integrates wave index and wave buffer. We evaluate RetroInfer thoroughly on popular LLMs and tasks, demonstrating both high throughput and high accuracy.

2 BACKGROUND AND MOTIVATION

2.1 Transformer-based LLMs and Attention

Transformers [91] are the dominant architecture for large language models (LLMs), relying on a multi-layer attention mechanism. As shown in Figure 2, given an input of n tokens, each represented as a high-dimensional vector, it is linearly transformed into three matrices: queries (Q), keys (K), and values (V). Within each layer's attention module, the model relates each token to others using multiple attention heads in parallel. The outputs from all heads are

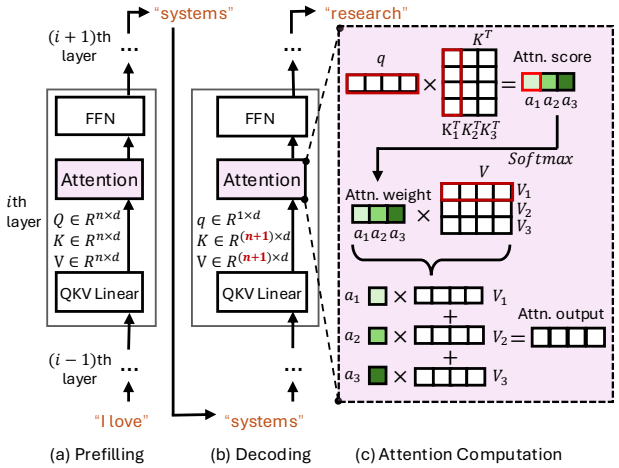

Figure 2: Transformer architecture with KV cache. In (c), deeper colors indicate higher attention weights (more important tokens). "I love systems" are the three input tokens.

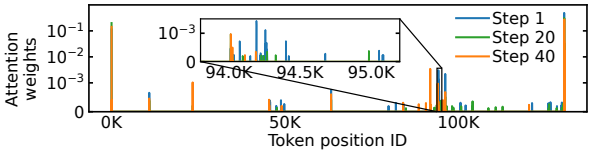

Figure 3: Dynamic sparsity in attention (Llama3-8B-1048K, *qa_1* task [38], layer 0, head 24). The distribution of top-100 attention weights varies across decoding steps.

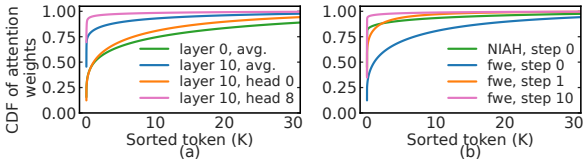

Figure 4: Variety of attention sparsity across (a) model layers and attention heads on *fwe* task [38] and (b) decoding steps and tasks on Llama3-8B-1048K.

then aggregated and passed to a feed-forward neural network (FFN) layer. This process is repeated for each layer of the model.

During inference, the transformer operates in two distinct phases: prefilling and decoding. In the prefilling phase, the model processes the input prompt as a whole; in the decoding phase, it generates tokens one at a time based on both the prompt and previously generated tokens.

To accelerate decoding, modern LLMs use a *key-value cache* (KV cache) [43, 73], which stores the key and value vectors of all previously seen tokens. This avoids redundant computation and significantly reduces decoding latency. However, the KV cache introduces substantial memory and bandwidth requirements, since its size and memory access grow linearly with both context length and batch size. Thus, KV cache makes long-context inference increasingly constrained by memory capacity and bandwidth limitations.

The attention output in decoding is computed as a weighted sum over value vectors, where the weights are given by the softmax of the inner product between the current query q and all cached keys:

$$\mathbf{o} = \mathbf{aV} = \text{Softmax}(\frac{\mathbf{qK}^T}{\sqrt{d}})\mathbf{V} = \sum_{i=1..n}(\frac{e^{\mathbf{q}\cdot\mathbf{K}_i^T/\sqrt{d}}}{\sum_{j=1..n} e^{\mathbf{q}\cdot\mathbf{K}_j^T/\sqrt{d}}} \cdot \mathbf{V}_i) \quad (1)$$

KV cache has become so essential that it has influenced model architecture changes in modern LLMs. Notably, Grouped Query Attention (GQA) [5] has become a standard in recent models [1, 2, 61, 78]. In GQA, several query heads within a group share a key-value head, which reduces KV cache memory consumption proportionally to the group size. However, the memory consumption remains substantial, especially in the long-context scenarios.

2.2 Limits to Scaling Long-Context Inference

Despite growing demand for long-context inference in use cases such as multi-turn conversations [27], code understanding [8], and reasoning [98], achieving scalable performance is challenging. While the KV cache reduces computation cost, the widening gap between GPU memory capacity and memory requirement of KV cache severely limits batch size and throughput.

GPU Memory Capacity Limits. The KV cache size grows linearly with both the context length and the batch size, quickly draining GPU memory. For instance, an A100 GPU (80GB memory) [66] can support a maximum batch size of 4 at a 128K context length for Llama3-8B. Exceeding this limit results in out-of-memory (OOM) errors. Similarly, the maximum context length the GPU can support is 512K[1]. Leveraging the aggregated memory from multiple GPUs is plausible but less cost-effective. Thus, many systems offload the KV cache to larger and cheaper CPU memory [44, 83, 102], but this incurs overhead due to PCIe transfers between CPU and GPU.

GPU Memory Bandwidth Bottlenecks. Memory bandwidth is another key limiting factor. With KV cache, each query at decoding phase linearly accesses all previous KV vectors. As the context length and batch size increase, the resulting access volume saturates memory bandwidth, a limitation widely recognized in recent work [10, 81]. In our experiments, scaling the batch size beyond 3 (at 128K context) leads to marginal throughput improvement due to GPU memory bandwidth saturation.

2.3 Sparsity-based KV Cache

A promising direction to alleviate the memory and bandwidth bottlenecks of the KV cache is to selectively access KV vectors, enabled by the inherent *sparsity* of the attention mechanism [16, 17, 22]. This sparsity arises naturally from the structure of the attention equation (Equation 1) and the exponential nature of the softmax function. Intuitively, value vectors associated with high attention weights[2] (i.e., elements of $\mathbf{a}$ in Figure 2) dominate the attention output $\mathbf{o}$. That is, a small subset of value vectors suffices to reconstruct the attention output, provided that the most important tokens can be accurately identified from the vast pool of KV vectors.

[1]Unless specified, the analysis is done in Llama3-8B-1048K [32] on an A100 GPU. Experiments on more models and multiple GPUs are available in Section 5.

[2]In this paper, we refer to $\mathbf{qK}^T$ in attention as the attention scores, Softmax$(\mathbf{qK}^T/\sqrt{d})$ as the attention weights.

Leveraging sparsity decreases the GPU computation, reduces memory access and helps alleviate the bandwidth bottleneck. Moreover, sparsity makes it feasible to offload the KV cache to CPU memory to address the capacity limitation, as the inference system can selectively access a subset of KV vectors over PCIe.

Challenges to Leverage Sparsity. Identifying important tokens is challenging for three reasons. First, as shown in Figure 3, important tokens (i.e., those with high attention weights) are scattered across the context, making their positions unpredictable. Second, tokens that are important at one decoding step (i.e., one query vector) may not remain so in subsequent steps. As shown in Figure 3, the overlap of the top-100 KV vectors across three decoding steps is only 31%, highlighting token importance dynamics. Third, the sparsity exhibits high variability from two sources: the architecture of the model itself and the nature of the decoding query. As shown in Figure 4, attention distributions differ significantly across model layers and attention heads (Figure 4(a)), potentially reflecting the varying semantic roles [49, 101]. Different queries within the same context or across different tasks exhibit substantially different sparsity ratios on the same attention head (Figure 4(b)).

Sparsity alone is not a panacea. A gap exists between sparsity ratio (i.e., negligible tokens percentage) and the bandwidth constraints imposed by the PCIe interconnect. GPU memory bandwidth (e.g., HBM) is about 60× higher than PCIe bandwidth on common GPUs (e.g., A100, H100, and H200). To hide PCIe transfer latency and prevent GPU stalls, the sparsity ratio must exceed 98% for the GPUs. Recent studies report lower sparsity ratios of 87.5% [89] and 90% [44], highlighting the need for careful system design that accounts for hardware constraints to achieve high throughput.

Existing Sparsity-based KV Cache Systems. Recent systems [15, 44, 49, 89, 102, 104] have explored the use of sparsity to facilitate long-context inference. However, they often fall short along two key axes: accuracy and efficiency.

Some systems rely on fixed-position heuristics to discard KV vectors [49, 104], often resulting in significant accuracy loss due to static assumptions about token importance. Others estimate importance by partitioning the KV cache into equal-sized chunks and using representative vectors [89, 102]. While GPU-friendly, this coarse-grained approximation reduces inference accuracy.

Systems that leverage sparsity to reduce attention computation but retain all KV vectors in GPU memory [89] are constrained by GPU capacity, limiting both context length and batch size. Works such as InfiniGen [44] and MagicPIG [15] offload the KV cache to CPU memory and selectively fetch vectors, either via speculative prediction or locality-sensitive hashing (LSH). However, inaccurate selection can degrade quality, and throughput is bottlenecked by PCIe bandwidth or lower CPU compute power.

3 VECTOR INDEX FOR SPARSITY-BASED KV CACHE

As pointed out in recent work [23, 53, 115], vector index – a classic technique for approximate nearest neighbor search (ANNS) [25, 39, 57, 59, 95, 119] – is a natural fit for retrieving important tokens and serves as the backbone of a sparsity-based KV cache. However, directly applying vector index to the KV cache is not sufficient and introduces several challenges.

Opportunities of Vector Index. The key observation is that Maximum Inner Product Search (MIPS) [79, 84], a variant of nearest neighbor search (NNS), can be seamlessly applied to identify critical tokens in attention mechanisms [53, 115]. High attention scores (i.e., important tokens) indicate that their key vectors have a large inner product with the query vector (i.e., they are similar).

ANNS aims to identify a desired number of vectors most similar to a given query in high-dimensional space, using a similarity metric such as inner product. Vector index [25, 57, 59, 95, 119] is a widely used technique for ANNS that organizes key vectors so that similar vectors are placed nearby; it greatly reduces the number of vectors accessed during search while maintaining near-optimal results.

Vector index is promising because it naturally handles the dynamic nature of sparsity (Figure 3): each decoding token (i.e., query vector) retrieves different results from the index based on importance, rather than relying on fixed positions in the context.

Challenges. The fundamental trade-off between accuracy and retrieval cost in ANNS persists and becomes more pronounced when applied to sparsity-based KV caching. Existing vector indexes are inadequate to address the accuracy challenge due to the high variability in attention sparsity (Figure 4). As in prior work [115], the retrieval cost remains substantial relative to the limited PCIe bandwidth, because it must account for this variability to retrieve more tokens for desired accuracy.

Moreover, efficiency challenges arise when using a vector index for sparsity-based KV cache, as it introduces index traversal and selective data access into an inference system optimized for dense GPU execution. A careful co-design of compute and memory across the hardware stack is essential.

First, sparse attention with vector indexing involves two types of data (index structures and KV vectors) and three types of computation (index traversal, index construction, and attention). Mapping this onto hardware introduces fundamental challenges due to CPU-GPU asymmetry. GPUs offer high-throughput compute and fast memory access but have limited capacity; CPUs provide abundant memory but lower compute throughput. This raises key questions about data placement, task assignment, and how to leverage parallelism and overlap to utilize both processors efficiently.

A natural starting point is to perform attention calculation on the GPU while storing high-volume KV vectors in CPU memory. Under this setup, efficient data movement is critical. Without careful orchestration, the GPU can be starved for data, especially under tight latency constraints.

Second, the system must reduce index traversal and construction costs while preserving retrieval quality. Index traversal can be costly on GPU due to irregular memory access and fine-grained operations (e.g., top-k selection in ANNS) while CPU-based traversal is constrained by weak compute power. Index construction adds additional overhead, and prior work [23, 36, 53] assumes offline construction, limiting the usability in online inference.

Finally, the control logic for managing index structures is often better suited for CPU execution. This necessitates a memory management layer that runs on the CPU and coordinates efficient paging and data transfers to the GPU. These challenges are further complicated by the distinct programming models of CPUs and GPUs, requiring careful coordination to ensure high performance and low overhead.

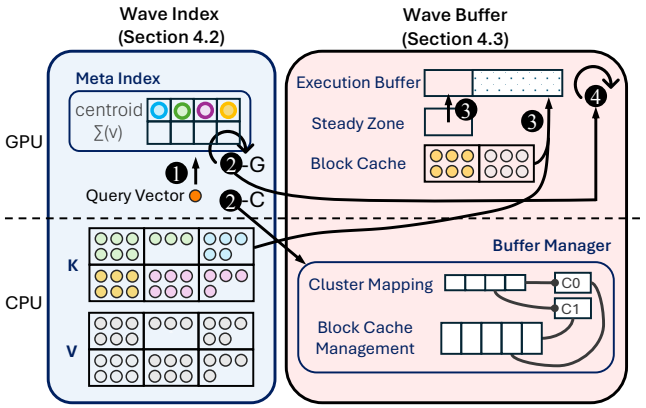

Figure 5: Architecture of RetroInfer. Circles with numbers represent the steps of attention computation, and steps with the same number occur in parallel. Centroids (bold-edged circles) are stored in the meta index.

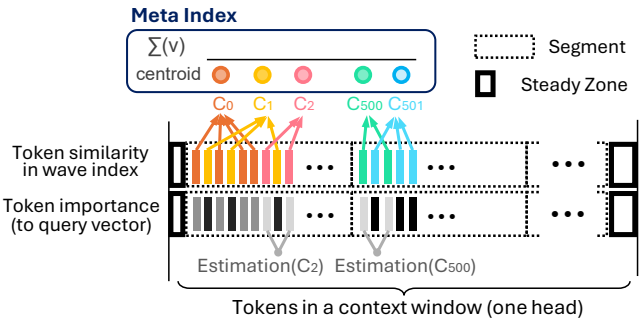

Figure 6: Attention-aware design of wave index. The wave index partitions the context into segments (dotted boxes), where each vertical bar denotes a token. In the first row, tokens with the same color form a cluster, and similar colors (e.g., yellow and orange) indicate key vector similarity. In the second row, darker bars indicate higher importance to the query. For example, a purple query may be close to a red vector (1st segment) and a blue vector (2nd segment).

4 RETROINFER

We present RetroInfer, a vector storage engine that builds a sparsity-based KV cache to scale long-context inference. We begin with a high-level overview, then describe the design of the *wave index* and *wave buffer*, and also discuss implementation.

4.1 Overview

The design of RetroInfer follows two principles: (1) be attention-aware, and (2) separate the concern of attention accuracy from the system efficiency.

We introduce an Attention-aWare VEctor index called *wave index* that retrieves the important KV vectors accurately and efficiently. We also introduce a *wave buffer* which manages memory across GPU and CPU to facilitate wave index, also in an attention-aware manner. Figure 5 shows the architecture of RetroInfer.

To make judicious use of the GPU memory, the wave index employs a cluster-based vector index design. Specifically, it partitions KV vectors into clusters based on their similarity, storing cluster centroids in a *meta index* as their representatives in GPU memory. Each meta index entry contains additional information (i.e., $\sum V$), which allows us to perform the precise attention computation on retrieved clusters, while approximating attention for others to cover varying sparsity ratios. All of the KV vectors are organized in contiguous KV blocks in CPU memory. We dive into the internals of wave index in Section 4.2.

The wave buffer serves two purposes. First, it contains several buffers in GPU memory to accelerate inference throughput. These include a *block cache* for KV vectors and an *execution buffer*, a dedicated memory region that sequentially arranges needed KV vectors for attention computation. The content of the execution buffer is copied from the steady zone, block cache, and directly from the CPU memory in case of a cache miss (details in Section 4.3). Second, it acts as a CPU-resident buffer manager that manages the block cache and data movement between GPU and CPU memory.

During decoding, RetroInfer computes the attention for each head in parallel, following the steps in Figure 5: ❶: The centroids are sorted according to the similarity to the query vector, determining a subset of more critical clusters to retrieve for precise attention computation, and the clusters to perform estimation. ❷: The GPU performs attention estimation (2-G), and a request is sent to the buffer manager to retrieve the needed clusters (2-C). ❸: The buffer manager ensures that the KV blocks are ready in the execution buffer through parallel data copying. ❹: The GPU computes the precise attention using the KV vectors in the execution buffer, while the attention estimation (2-G) result is merged.

4.2 Wave Index

Wave index employs a cluster-based vector index and designs for the attention mechanism to ensure accurate and efficient computation.

Index Organization. Wave index partitions the KV vectors in the context into clusters using spherical k-means [37], based on the similarity between key vectors. We choose spherical k-means because it is widely used for inner-product-based clustering [24], aligning with the similarity metric between query and key vectors in attention mechanisms. Each cluster is represented by its centroid, computed as the mean of its key vectors. We design the wave index based on a cluster-based vector index for two reasons. First, centroids serve as compact representatives and can reside in fast GPU memory, aligning well with CPU-GPU asymmetry. Second, clustering improves access locality by reducing random accesses over PCIe. As illustrated in Figure 6, although several similar tokens (i.e., 2nd, 4th, and 8th) are irregularly distributed in the context, wave index groups them into the same cluster (i.e., yellow).

When serving a query vector, wave index sorts the centroids based on the inner product between the query vector and the centroids, as the inner product determines the attention score. Classic vector indexes have addressed the complexity of high-dimensional vector spaces to effectively perform clustering and compute centroids [24, 37]. However, the goal of classic ANNS is to retrieve the top-k vectors by selecting candidates from clusters and performing pairwise comparisons to the vectors for further ranking, which is

costly on GPU. In contrast, wave index retrieves all vectors from the critical clusters and incorporates them directly into the attention computation, avoiding the selection cost. Since the clusters are highly coherent, this design also improves the attention accuracy by potentially including more tokens that are also important.

To accelerate index construction, we introduce a technique called *segmented clustering*, where the input sequence is divided into segments, and clustering is performed within each segment, instead of clustering over the entire input sequence as in classic vector index. For example, in Figure 6, vectors in the first segment (i.e., red, yellow, and orange ones) are clustered independently of vectors in the second segment. We describe its rationales and details later.

For physical placement, the meta index, which requires only a small memory footprint, resides in GPU memory. The meta index contains a table of centroids, the sum of the value vectors and the cluster size, as shown in Figure 5. The large volume of key and value vectors resides in CPU memory, organized into contiguous KV blocks. This block-based organization facilitates efficient data movement between GPU and CPU memory.

Tripartite Attention Approximation. We introduce *tripartite attention approximation* to ensure the accuracy of attention computation with reduced computation cost by leveraging both the properties of the attention mechanism and the advantage of vector retrieval. The final KV vectors used for attention computation consist of three parts: *steady zone*, *retrieval zone*, and *estimation zone*, and we denote the attention output as o^0, o^1, and o^2 respectively.

Steady, Retrieval, and Estimation Zones. Recent studies have revealed that a range of tokens located at the beginning and end of the context are consistently important [104]. They are categorized as steady zone in wave index, as shown in Figure 6. Therefore, tokens in the steady zone are directly included based on their positions, rather than being organized into the vector index. The KV vectors retrieved by the centroids and clusters are categorized as retrieval zone; these tokens are directly used for attention computation.

Due to the high variety of sparsity ratios across model layers, attention heads, queries, and task types, determining the optimal number of clusters for the retrieval zone is challenging, if not unattainable. To compensate for the potential accuracy loss of non-retrieved clusters, we introduce an accuracy-bounded attention estimation mechanism. Tokens in the non-retrieved clusters are categorized as the estimation zone. This approach effectively handles sparsity variation without increasing data transfer over PCIe.

Combining the three zones, we have p KV vectors in the steady zone, and m clusters in the retrieval and estimation zones, with r clusters assigned to the retrieval zone and the remaining assigned to the estimation zone. The $m - r$ clusters in the estimation zone enhance the robustness of the chosen r clusters for retrieval in the face of varying sparsity ratios, thus improving attention accuracy. Intuitively, our tripartite attention approximation aligns with the inherent sparsity of attention. From o^0 to o^2, the attention weight computation becomes progressively lighter and more approximate, while the resulting values decrease in magnitude.

The clusters exhibit high coherence and strong alignment with attention scores, along with highly representative centroids (Figure 7), enabling a robust distinction between the retrieval and the estimation zone. Precise attention computation is critical for top-ranked centroids, which have a greater impact on the output. In contrast, the estimation zone targets clusters with lower attention scores, where lightweight approximations can be applied to significantly accelerate computation. This design ensures the efficient retrieval of important tokens while maintaining accuracy by bounding the influence of tokens in the estimation zone.

Sensitivity analysis in Section 5.4 shows that using a small fixed steady zone, a small retrieval zone and a larger estimation zone consistently delivers both high throughput and high accuracy across different tasks and models.

Accuracy-Bounded Attention Estimation. The key idea for accuracy bounded attention estimation is to use the centroids to estimate the attention weights for all KV vectors within a cluster, thereby avoiding the computation of attention for each individual KV vector, striking a balance between accuracy and efficiency.

Consider an attention head with p KV vectors in the steady zone and m centroids, denoted as $\mathbf{C} \in \mathbb{R}^{m \times d}$, with corresponding cluster sizes $\mathbf{s} \in \mathbb{R}^{m \times 1}$. The attention weight for the KV vectors in the i-th cluster is estimated by:

$$\tilde{a}_i = \frac{e^{\mathbf{q} \cdot \mathbf{C}_i^T / \sqrt{d}}}{\sum_{j=1..p} e^{\mathbf{q} \cdot \mathbf{K}_j^T / \sqrt{d}} + \sum_{j=1..m} \left(s_j \cdot e^{\mathbf{q} \cdot \mathbf{C}_j^T / \sqrt{d}} \right)}, \; i = 1..m \quad (2)$$

The accuracy of this estimation is assured because the centroid estimation acts as a lower bound for the sum of exponential inner product values within a cluster. For the i-th cluster with s_i key vectors, the centroid is the average of all key vectors in the cluster. By applying Jensen's inequality [41], we derive the following bound:

$$\mathbf{C}_i = \frac{\sum_{j=1..s_i} \mathbf{K}_j}{s_i}, \quad e^{\mathbf{q} \cdot \mathbf{C}_i^T / \sqrt{d}} \leq \frac{\sum_{j=1..s_i} e^{\mathbf{q} \cdot \mathbf{K}_j^T / \sqrt{d}}}{s_i} \quad (3)$$

Figure 7 illustrates the accuracy bound of the estimation. Notably, while clusters are ranked by centroids ($\mathbf{q} \cdot \mathbf{C_i}^T$), the estimation is not monotonically decreasing because the sum of attention scores within a cluster is influenced by the non-uniform cluster sizes (s_i).

To further reduce the overhead of accessing corresponding value vectors, we sum the value vectors of each cluster during the index construction and store cluster size and summed value vectors ($VS \in \mathbb{R}^{m \times d}$) in the meta index. This approach avoids individual value access during inference and is based on the following equality for the partial attention output $\tilde{o}_i$ derived from a cluster C_i and the attention output o_2 for all non-retrieved clusters:

$$\tilde{\mathbf{o}}_\mathbf{i} = \sum_{j=1}^{s_i} (\tilde{a}_i \cdot \mathbf{V}_j) = \tilde{a}_i \sum_{j=1}^{s_i} \mathbf{V}_j = \tilde{a}_i \cdot VS_i, \quad \mathbf{o_2} = \sum_{i=1}^{m-r} \tilde{\mathbf{o}}_\mathbf{i} \quad (4)$$

As such, the time complexity of attention estimation for non-retrieved clusters is $O(m - r)$, which is substantially lower than accessing individual KV vectors, ensuring the efficiency of our approach, particularly for longer contexts.

Lightweight Index Construction and Updates. Index construction occurs during the prefilling phase, where the full KV vectors is offloaded to CPU memory, and clustering is performed on the GPU to produce the meta index. The main challenges in index construction are reducing computation overhead to prevent slowing down the prefilling. While cluster-based vector index incurs lower index construction cost among vector index approaches, performing k-means across the entire input sequence is still expensive.

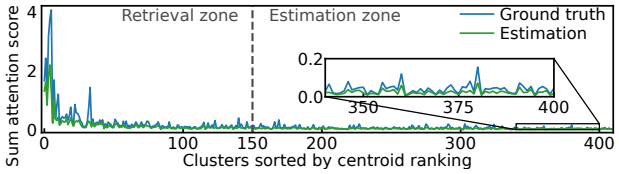

Figure 7: Centroid representativeness and estimation accuracy (Llama3-8B-1048K, layer 30, head 24, *fwe*, 128K context). The x-axis is the ranking of centroids during index traversal. The blue line indicates that top-ranked centroids have higher cumulative attention scores, showing the effectiveness of clustering. Attention estimation (green) follows the same trend of ground truth with a tight bound.

To save construction cost, wave index introduces *segmented clustering*, performing k-means clustering within each segment independently, reducing the time complexity of clustering by a factor of s, where s is the number of segments. This design is based on the observation that key vectors exhibit coarse-grained spatial locality within the input sequence[3]. Key vectors within a broader region (i.e., a single segment) tend to be similar to each other but dissimilar to those in other segments, which negates the need to include distant keys during clustering. As shown in Figure 6, tokens from the three clusters (i.e., yellow, red, and orange) are relatively similar. Though clustering results may vary across different contexts or attention heads, tokens in the first segment are unlikely to be similar to those in the second segment (e.g., green and blue tokens). Section 5.4 presents the analysis of segmented clustering, where an 8K segment size achieves index quality comparable to global k-means, while significantly reducing the build cost.

Such coarse-grained spatial locality does not contradict the fact that important tokens (i.e., clusters) are usually scattered throughout the context (Figure 6). This is due to the fact that similarity-based clustering is conducted in a high-dimensional vector space and the query vector in attention behaves as a special type of "out-of-distribution query vector." [53] In this context, the key vectors similar to the query vector can possibly be distant from each other in the vector space [12]. We address this with a classic centering technique [64] (inspired by MagicPIG [15]) to ensure the clustering effectively captures the attention importance.

The spatial locality also helps in managing newly generated tokens during decoding. New tokens are appended to the steady zone, and k-means clustering is applied once the local window reaches the size of a segment. This design appends new clusters into the index, avoiding full re-clustering and significantly reducing index update overhead. We empirically set the update segment size to 1K tokens through extensive experiments, a choice that well balances the inference accuracy and efficiency. Since clustering is performed once every 1024 tokens, the clustering cost is effectively amortized among generated tokens. Our evaluation shows that index update contributes only a 0.2% latency overhead during decoding. A larger segment size increases GPU memory consumption for buffering tokens, reducing batch size and limiting decoding throughput.

[3]We attribute the spatial locality to RoPE [87], which encodes positional information in key vectors. This effect is confirmed by observing significantly less locality when performing k-means clustering on Pre-RoPE versus Post-RoPE key vectors.

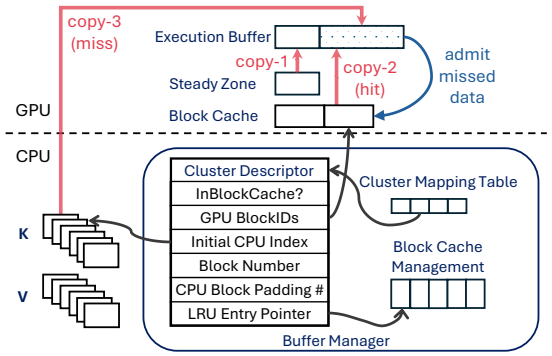

Figure 8: Design of wave buffer. Black arrows denote pointer relationships between data structures. Red arrows indicate the three possible sources for assembling the execution buffer. Missed data is admitted into the block cache by copying from the execution buffer (blue arrow).

4.3 Wave Buffer

With the wave index, RetroInfer significantly reduces the number of KV vectors required per query, lowering PCIe traffic to less than 2% of that in full attention. However, the inference throughput could still suffer due to the limited PCIe bandwidth for long-context inference. Drawing on the classic wisdom of computer systems, the performance and capacity disparity between GPU memory and CPU memory, akin to the von Neumann bottleneck [92], necessitates the use of buffer caches in the fast, low-capacity GPU memory. Thanks to the high-quality retrieval provided by the wave index, adjacent decoding steps for neighboring query vectors show strong temporal locality, as only important tokens are loaded into GPU memory. Thus, caching is highly beneficial.

The organization of the wave index, combined with the hardware characteristics, poses unique challenges in managing the memory used in RetroInfer to achieve better throughput. Specifically, what should be cached, how to manage data movement and the control plane of caching, and finally, how can all these movements be effectively parallelized? We address these challenges with the wave buffer, a comprehensive memory management component that aligns well with the attention mechanism.

Buffer Organization. Within GPU, wave buffer contains a block cache for the KV vectors in the retrieval zone, a small buffer for the steady zone, and an attention-layer-share execution buffer. The control plane of the wave buffer resides in CPU memory, with its data structures managed by the buffer manager running on a CPU thread pool. Figure 8 shows the organization of wave buffer.

KV Block Cache. We observe a strong temporal locality of important tokens: neighboring decoding steps (i.e., query vectors) have a high overlap of critical tokens due to topic continuity and syntactic proximity. With a GPU cache size equal to 5% of the total KV vectors, the average cache hit ratio across tasks ranges from 0.79 to 0.94, indicating high cache ability of important KV vectors. This observation is also supported by recent works [100, 105].

However, exploiting this temporal locality in the KV Block Cache is challenging for two reasons. First, a mismatch exists between the cluster operations and memory management units. Clusters are

suitable as the *logical unit* of cache access to align with the wave index's cluster-based operations, while fixed-size KV blocks are preferred as the *physical unit* of storage for simplicity in memory management. Since a single cluster can span multiple blocks, a semantic gap arises between these two units. To bridge it, we introduce a cluster mapping table as a layer of indirection, implemented as an array indexed by cluster ID for fast access. Each entry is a cluster descriptor (Figure 8), mapping the logical cluster to its physical block addresses in GPU cache and CPU memory. This design enables efficient translation from cluster-level retrieval to block-level data movement while preserving management simplicity.

Second, cache management overhead must remain far below the per-layer decoding latency to maintain the inference efficiency. Specifically, at a 128K context length, one layer of Llama3-8B executes in 720 μs on the GPU. In contrast, integrating a standard LRU cache implementation with PyTorch-based data transfer introduces roughly 1.5 ms of additional overhead per layer, which offsets any potential benefit from reduced PCIe traffic. To meet this stringent latency requirement, wave buffer introduces two optimizations. First, we implement dedicated GPU kernels for efficient GPU–CPU data transfers (detailed in Section 4.6), reducing the copy time from 1.3 ms to 100 μs under cache misses. Second, we move the cache replacement logic to the CPU, decoupling cache access on the critical path from expensive cache updates. This is facilitated by the hybrid GPU-CPU structure–after the lookup, the KV vectors must be immediately ready for the GPU in the execution buffer, whereas the cache update can be performed asynchronously by the CPU, in parallel with the data copy and attention computation. Compared to GPU-native cache management [96], our CPU-managed, asynchronously updated cache achieves low latency requirement, simplicity, and better extensibility for various caching policies.

Synchronous Cache Access. The mapping table associates each cluster ID with its block IDs in either GPU memory (cached) or CPU memory (missed). Upon obtaining the cluster IDs in the retrieval zone for a query, the wave index sends them to the CPU thread running the buffer manager, which looks up the mapping table to determine the clusters' locations and issues the memory copy accordingly. Table lookup is accelerated using CPU multi-threading, without synchronization overhead, as this process is read-only. As shown in Figure 5, we parallelize the mapping table lookup with attention estimation on the GPU to reduce inference latency.

Asynchronous Cache Update. Following the mapping table lookup, the wave buffer immediately submits a cache update request to the CPU thread pool for asynchronous execution. As a result, cache updates are decoupled from cache access, allowing the wave buffer to issue memory copy operations via GPU threads while concurrently making cache replacement decisions and metadata updates (e.g., LRU lists) on the CPU. The replacement decision is then enacted by issuing a request to the GPU to schedule a copy from the execution buffer to the block cache, thereby admitting the data into GPU memory and completing the cache update.

Decoupling cache access and update allows CPU latency to overlap with GPU execution, accelerating the overall process.

Assemble the Execution Buffer. The execution buffer contains the final key-value vectors in the steady zone and the retrieval zone, arranged in contiguous memory. The sequential nature of the execution buffer makes them directly usable by FlashAttention [18] to compute the partial attention output. This output is combined with the estimated attention to produce the final attention output.

Three types of data copy may be needed to assemble the execution buffer, as shown in Figure 8: 1) from the steady zone (GPU to GPU), 2) from the KV block cache (GPU to GPU), and 3) from KV blocks (CPU to GPU). We have developed a highly-optimized copy operator to accelerate the copy process (details in Section 4.6).

4.4 Constrain Prefilling Latency

Because prefilling is compute-intensive, special care is required to perform all buffer-related computation (e.g., construct the mapping table) alongside the lightweight index construction. We carefully constrain prefilling latency by fully utilizing GPU-CPU parallelism.

After obtaining the Q, K, and V during prefilling, RetroInfer asynchronously offloads the KV vectors to CPU memory while the GPU performs segmented clustering. The clustering results are then sent to the CPU, allowing the wave buffer to asynchronously construct its data structures, including the mapping table and metadata for cache replacement, in parallel with attention calculation. With this parallelism, only segmented clustering remains on the critical path of the prefilling phase, and its latency is negligible (less than 5%, as shown in Section 5.3).

4.5 Scale to Multiple GPUs

Serving inference for larger LLMs typically adopts model partitioning across multiple GPUs and model-parallel execution [50, 122]. RetroInfer extends naturally to this setup as the wave index and wave buffer are modular components co-located with each attention head, requiring no coordination across layers or heads. In multi-GPU setting, RetroInfer integrates with partitioned layers or heads and manages the corresponding index and buffer on each GPU. Section 5.3 demonstrates its effectiveness.

4.6 Implementation

We implement a custom Triton [90] kernel for segmented clustering in the wave index. The kernel accepts KV vectors and executes k-means in parallel across attention heads and segments. It then computes the value sums and stores the results in the meta index.

GPU-side memory copies for accessing and updating the KV block cache and execution buffer are complicated, as source and destination addresses are often non-contiguous and may involve both CPU–GPU and GPU–GPU transfers. We implement CUDA kernels (~1,000 LoC) for efficient execution buffer transfers and block cache updates, skipping over fragmented regions within blocks to mitigate fragmentation. Since attention heads may exhibit varying copy patterns due to differing cache hit ratios, number of threads are dynamically changed to maintain performance.

Attention in the estimation zone differs from standard attention, as cluster sizes are required to compute attention weights. We modify FlashAttention [18] kernel to support weighted attention, which also efficiently merges attention outputs from three zones.

5 EVALUATION

We thoroughly evaluate RetroInfer, comparing it with full attention and state-of-the-art sparse-based KV cache systems across various models and tasks. The results demonstrate that:

- RetroInfer outperforms sparsity-based baselines in model accuracy, and is the only system that matches full attention accuracy.
- RetroInfer achieves high throughput while scaling the context lengths and batch size.
- The techniques incorporated into the wave buffer, such as GPU caching and asynchronous cache updates, effectively minimize data transfer over PCIe and boost throughput.

5.1 Experimental Setup

We run experiments on a VM server with NVIDIA A100 GPUs (80GB memory) and an AMD EPYC 7V12 CPU (1.7TB memory). It has 4 NUMA nodes (12 cores each). The GPU and CPU are linked via PCIe 4.0 (×16), with a unidirectional bandwidth of 32GB/s. The server runs Ubuntu 22.04 with CUDA 12.4 and PyTorch 2.5.

Models. We evaluate RetroInfer on four open-source LLMs: Llama3.1-8B [61], Qwen2.5-7B [78], Llama3-8B-1048K [32] and Qwen2.5-72B [77]. Llama3.1-8B, Qwen2.5-7B and Qwen2.5-72B support up to 128K tokens, while Llama3-8B-1048K support a 1048K context window. Unless otherwise specified, all models are evaluated on a single A100 GPU except that Qwen2.5-72B is evaluated on eight A100 GPUs by evenly partitioning the model layers across GPUs. We include two advanced reasoning models DeepSeek-R1-Distill-Llama-8B [19] and DeepSeek-R1-Distill-Qwen-7B [20], to evaluate RetroInfer on long-generation reasoning scenarios.

Baselines. We compare with a highly-optimized full attention baseline that uses FlashInfer [111] in both accuracy and efficiency. vLLM [43], which internally relies on full attention, is included for end-to-end latency evaluation in Section 5.3. We also compare RetroInfer with four state-of-the-art dynamic sparsity-based KV cache systems: Quest [89], MagicPIG [15], InfiniGen [44], and PQCache [115]. Quest is GPU-only and employs chunk-based retrieval based on representative vectors, while other three systems are GPU-CPU frameworks which offload KV cache to CPU memory. MagicPIG employs LSH to sample relevant tokens and relies on CPU for most computation to reduce PCIe transfer. In contrast, InfiniGen speculatively prefetches important tokens to GPU based on similarity between adjacent layers. PQCache, a recent system that applies ANNS to attention, leveraging product quantization (PQ) to identify important tokens and load them to GPU for computation. We use their open-sourced codes [40, 58, 74, 76] for evaluation.

Parameters. Each sparsity-based baseline uses the parameters from its original paper[4]. RetroInfer uses one centroid per 16 tokens on average, with segments of 8K tokens and 10 iterations for k-means to balance index quality and build time. The retrieval budget is set as 1.8%[5] across different sparsity-based systems and tasks, aligning with the budget used in [15] for fair comparison. For RetroInfer, the steady zone contains 4+64 tokens for initial and local windows while the estimation zone is set as 23.2% of total clusters so that three zones cover ~25% of the context. We study the impact of different zone sizes in Section 5.4. For wave buffer, we explored several cache policies and selected LRU as default due to its best performance. The GPU cache size is empirically set as 5% of all KV vectors and the block size is set as 2KB. We use 24 logical threads in one NUMA node. MagicPIG uses all cores from two NUMA nodes, following its original setting [58].

[4]Quest uses a chunk size of 16 and applies full attention to the first two model layers; MagicPIG employs 10 hash functions and 150 hash tables and applies full attention in selected layers; InfiniGen uses 32 partial channels; PQCache uses 2 partitions and 6-bit PQ codes for context ≤ 16K and switches to (4, 8) for context > 16K.

[5]For RetroInfer, in the 128K context with 8192 clusters, we set 150 clusters in the retrieval zone, so the retrieval budget is $\frac{150}{8192} \approx 1.8\%$.

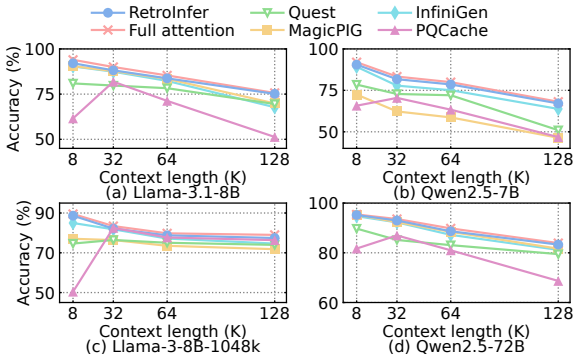

Figure 9: RULER accuracy under different context lengths and models. RetroInfer is the only solution that matches full attention accuracy across different context lengths.

Benchmarks. We utilize two representative long-context benchmarks for end-to-end accuracy evaluation. (1) RULER [38]: a comprehensive benchmark covering 13 different tasks such as aggregation and question answering. Each task contains 200 examples, under context lengths from 8K to 128K. (2) Needle-in-a-haystack (NIAH) [33]: it challenges LLMs to retrieve critical information ("needle") from a context ("haystack"). We use NIAH to stress model's accuracy up to 1 million tokens. End-to-end task accuracy is used to show the inference quality of RetroInfer and baselines.

We also utilize two widely used reasoning benchmarks, AIME24 [68] and GPQA [80], to evaluate long-generation ability, which feature inputs with a few hundred tokens and up to 32K output tokens. For GPQA, we uniformly sample 50 examples for evaluation.

We evaluate RetroInfer and other baselines' inference efficiency with different batch sizes and context lengths. By default, we use Llama3-8B-1048K and NIAH to test RetroInfer up to 1 million tokens. Additional models and tasks and end-to-end evaluations are also covered in Section 5.3.

5.2 Inference Accuracy

We first analyze the end-to-end accuracy on different benchmarks.

RULER and NIAH. Figure 9 shows the average task accuracy of different systems under different context lengths on RULER. For all context lengths and models, RetroInfer consistently matches the accuracy of full attention and outperforms all sparsity-based baselines. At 128K context, RetroInfer has only a drop of 0.66%/1.50%/1.97%/0.73% compared to full attention on Llama3.1-8B/Qwen2.5-7B/Llama3-8B-1048K/Qwen2.5-72B, respectively. Compared to the sparsity-based systems, RetroInfer achieves accuracy improvements ranging from 1.40% to 46.47%. These results indicate that RetroInfer is able to effectively retrieve important tokens using wave index and cover the varying sparsity by estimation robustly.

Figure 10 further shows that RetroInfer achieves 100% accuracy with context lengths up to 1 million tokens. This demonstrates our system's capability to support million-token with high accuracy.

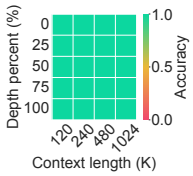

Figure 10: Needle-in-a-haystack results on Llama3-8B-1048K.

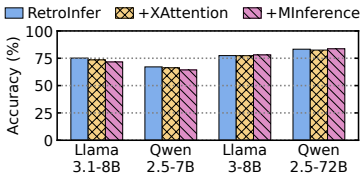

Figure 11: RULER accuracy (128K) of RetroInfer combined with XAttention and MInference.

Table 1: Reasoning tasks accuracy (pass@8 [65]). RetroInfer achieves comparable accuracy to full attention.

Systems	AIME24	GPQA	Systems	AIME24	GPQA
DeepSeek-Llama	*73.30*	*68.00*	*DeepSeek-Qwen*	*76.70*	*72.00*
Quest	46.70	50.00	Quest	46.70	62.00
InfiniGen	36.70	62.00	InfiniGen	60.00	66.00
PQCache	30.00	58.00	PQCache	13.33	46.00
RetroInfer	**76.70**	**68.00**	RetroInfer	**73.30**	**68.00**

Reasoning Tasks. We evaluate RetroInfer on two reasoning benchmarks [68, 80] using reasoning models [19, 20]. The evaluation follows the standard settings [21] and sets the maximum generation length as 32K. Since the input length is relatively short, we do not construct indexes during the prefilling stage. As the generation progresses, once the combined length of prompt and generated tokens exceeds 1K, the index is initialized and then updated incrementally with an update segment size of 1K tokens. We exclude MagicPIG from the comparison, as it does not support index updates for long-generation tasks. Table 1 shows that RetroInfer matches the accuracy of full attention and achieves higher accuracy compared to the baselines, demonstrating RetroInfer's strong capability for long generation. Notably, RetroInfer achieves higher accuracy than full attention in some tasks. This is because the sparsity mechanism helps filter out less informative tokens generated by the reasoning process [14, 109], allowing the model to attend to more relevant context and improve generation quality.

Compatibility with Sparse Prefilling. RetroInfer and other baselines focus on optimizing the decoding phase. Now we study RetroInfer's compatibility with the state-of-the-art sparse prefilling methods XAttention [106] and MInference [42], which aim at reducing prefilling latency. We evaluate RetroInfer on RULER with and without them, as shown in Figure 11. The accuracy RetroInfer combined with XAttention or MInference only drops by 1.52% on average. This demonstrates that RetroInfer is compatible with prefilling acceleration techniques.

5.3 Inference Efficiency

We first evaluate decoding throughput across context lengths and batch sizes, followed by prefilling latency analysis, and lastly stress test the end-to-end request latency and throughput of all systems under different loads. vLLM is not suitable for direct comparison in decoding throughput, as it employs internal request scheduling policy [4], making it challenging to explicitly control the batch size. Thus we only include it in end-to-end evaluation.

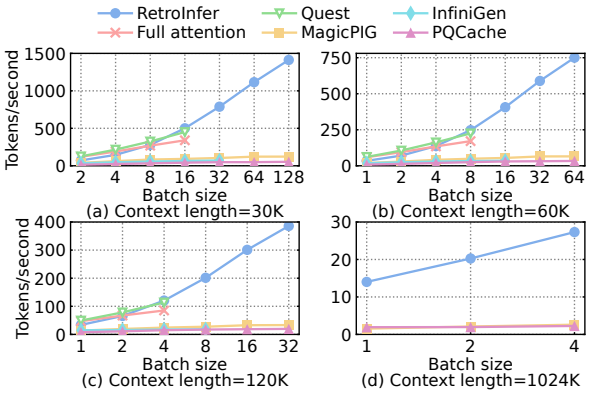

Figure 12: Decoding throughput at different context lengths (Llama3-8B-1048K). RetroInfer scales well with the batch size and achieves the best throughput on all lengths.

Decoding Throughput. Figure 12 shows the decoding throughput of RetroInfer and other systems across four different context lengths ranging from 30K to 1M. For the context length of 30K, 60K and 120K shown in Figure 12 (a–c), RetroInfer outperforms full attention by 4.1×, 4.4×, and 4.4× respectively. The speedup over sparsity-based systems ranges from 3.3× to 22.8×. The efficiency of RetroInfer stems from wave buffer, significantly extending the batch sizes while maintaining low overhead. When the batch size is small, full attention and Quest show comparable or slightly higher throughput than RetroInfer due to their fully GPU processing, but they cannot scale beyond GPU memory. MagicPIG's throughput is constrained by limited CPU compute, while InfiniGen suffers from low throughput due to relatively expensive speculative operations and frequent PCIe data transfers. PQCache falls behind mainly due to its inefficient GPU-CPU memory management and increasing overhead of fetching PQ codebook as the context length increases.

At 1M context, full attention and Quest incur OOM. InfiniGen face the same issue as the key cache retrained on GPU for speculation exceeds GPU memory. Therefore, we compare only with the remaining systems. RetroInfer outperforms MagicPIG and PQCache by 10.5× and 12.2×, respectively. This demonstrates RetroInfer's efficiency for supporting extremely long-context.

We also evaluate the decoding throughput on different tasks and models. As shown in Figure 13(a), RetroInfer outperforms full attention by 3.4–4.6× across different tasks. The throughput variation across tasks is due to differing cache hit ratios. The advantage over sparsity-based systems still holds, with RetroInfer achieving 2.6–3.5× throughput than the best-performing baseline Quest on four tasks. Figure 13(b) shows that RetroInfer outperforms baselines by 2.4–24.4× on four models. Thus, RetroInfer is effective across different model architectures and model sizes (7B to 72B). The 72B model scales well across multi-GPUs, maintaining an advantage ranging from 2.5× to 24.4× over baselines.

Prefilling Latency. We evaluate the prefilling latency across context lengths. To avoid OOM errors at 1024K context, we offload the KV cache to CPU memory and measure the prefilling latency. Offloading introduces only 0.4% overhead for the prefilling. As illustrated in Figure 14, RetroInfer achieves only 6% and 3% higher

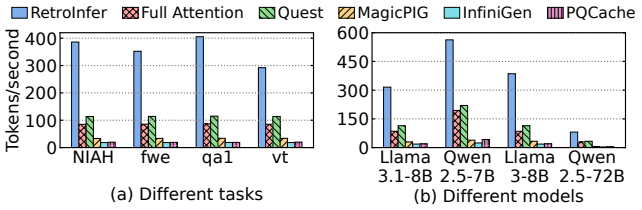

Figure 13: Maximum decoding throughput across tasks and models. RetroInfer consistently outperforms others.

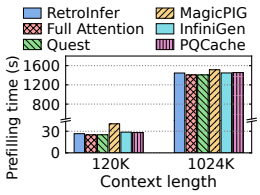

Figure 14: Prefilling latency under different context lengths. RetroInfer's prefilling latency is only slightly higher than full attention due to lightweight index building.

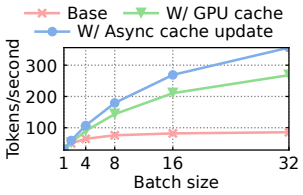

Figure 15: Effect of buffer design. "Base" offloads KV cache to CPU (no GPU cache), showing low scalability; "W/ GPU cache" improves throughput; "W/ Async cache update" further boosts the speed.

prefilling latency than full attention at 120K and 1M context lengths, respectively. We attribute this to the lightweight segmented clustering and asynchronous wave buffer construction. To reduce prefilling latency, RetroInfer can be combined with sparse prefilling methods; for example, at 1M context, XAttention and MInference help reduce the prefilling time to about 600*s* and 220*s*, respectively.

End-to-end Efficiency. We further evaluate end-to-end request latency and throughput including both prefilling and decoding phases under varying loads. We choose two representative workloads: (1) long input, consisting of 120K input tokens and 4K output tokens; and (2) long output, consisting of 512 input tokens and 32K output tokens. Figure 16 shows the latency-throughput curves.

For long-input workloads, under low load (e.g., two requests), RetroInfer is slightly slower than GPU-only full attention, Quest, and vLLM because they do not incur PCIe transfers. To better utilize GPU resources under lighter loads, we implement a GPU-only version of RetroInfer denoted as "RetroInfer-GPU", which achieves latency comparable to Quest. When the load increases, RetroInfer scales well and achieves 1.8–7.8× higher throughput than baselines at the same latency. We further integrate a sparse prefilling technique XAttention with all baselines for lower prefilling latency, but only show "RetroInfer+XAttention" in Figure 16 for better readability. With XAttention, RetroInfer achieves even greater gains, delivering 2.1–10.5× throughput speedup.

For long-output workloads, we exclude MagicPIG due to lack of index update support. As shown in Figure 16(b), with the relatively small context (≤ 32K), the KV cache can fit into GPU memory under low loads. At this load, RetroInfer does not outperform GPU-only baselines, but switching to RetroInfer-GPU achieves latency

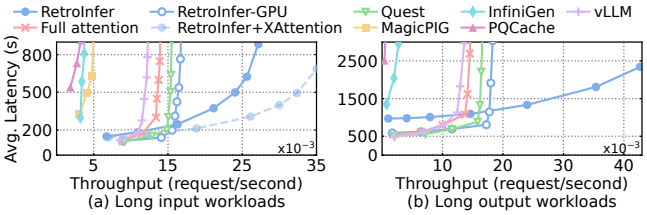

Figure 16: End-to-end request throughput vs. average latency at two workloads. RetroInfer scales well on both.

comparable to Quest. As load increases, RetroInfer outperforms baselines by 2.7–70.8× by supporting larger batch sizes.

5.4 Micro Analysis

We quantify the impact of individual design decisions of RetroInfer including different components of wave buffer, impact of different zone sizes and segment sizes and effect of attention estimation.

Effect of Design Decisions in Wave Buffer. We assess the impact of two design choices in the wave buffer: GPU caching and asynchronous cache updates. As shown in Figure 15, without GPU caching (Base), throughput cannot scale with increasing batch sizes, primarily due to PCIe bandwidth constraints. Adding GPU caching substantially reduces data transfer, enabling scalable throughput. Furthermore, the asynchronous update of the mapping table further boosts throughput by reducing latency on the critical path.

Impact of Different Zone Sizes. We study the impact of different zone sizes on model accuracy and efficiency using two representative tasks from RULER. The retrieval task *s3_niah* exhibits high sparsity, whereas *qa_1* shows more sparsity variation depending on the questions posed to the context. When varying the size of one zone, we keep other zone sizes fixed as in the evaluation setting. We report the decoding throughput on A100 and A6000 GPU (48 GB memory) [67] respectively to evaluate efficiency across hardware.

As shown in Figure 17(a-b), increasing the retrieval budget improves task accuracy but greatly reduces throughput due to increasing PCIe data transfers. RetroInfer reaches full-attention accuracy at a 1.8% retrieval budget across tasks, with diminishing returns beyond this point. The 1.8% budget is robust across tasks due to RetroInfer's estimation zone, which efficiently captures varying sparsity. We analyze this in detail below.

Figure 17(c-d) shows that increasing estimation budget improves accuracy, reaching to full-attention accuracy at ~23.2%, with greater gains for tasks with higher sparsity, like *qa_1*. Unlike retrieval, estimation imposes lower overhead on decoding throughput thanks to its lower computational cost and avoidance of PCIe transfers. This motivates our choice to keep the retrieval zone small (1.8%) and allocate a relatively larger estimation zone (23.2%) to capture varying sparsity patterns, for both high throughput and accuracy.

We empirically set the steady zone size to 4+64 based on common practice [15, 104]. More configurations (e.g., sink tokens only) are evaluated in Figure 17(e-f). Both sink and local window tokens are important, with sink tokens contributing more to accuracy. Larger steady zone sizes yield marginal accuracy gains, making 4 + 64 sufficient. Moreover, steady zone size has minimal impact on throughput due to its relatively small size.

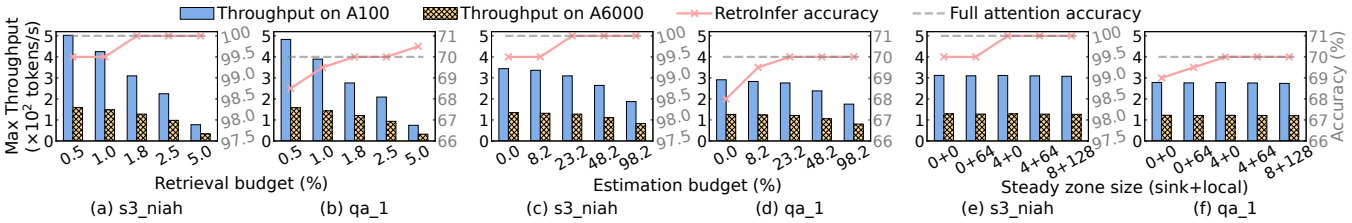

Figure 17: Impact of three zone sizes on maximum decoding throughput and task accuracy (Llama3.1-8B, 128K context).

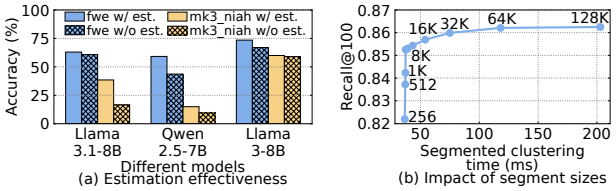

Figure 18: (a) Estimation helps improve the model accuracy. (b) Index build time vs. accuracy at 128K context. The number next to the circle indicates the segment size. Segment size = 8K well balances index build time and clustering accuracy.

Impact of Attention Estimation. We quantify its effect by comparing RetroInfer with and without attention estimation. As depicted in Figure 18(a), estimation enhances task accuracy by up to 20%. Importantly, this improvement is achieved without compromising efficiency, as it can be overlapped with wave buffer accesses.

Impact of Different Segment Sizes. We study the impact of different segment sizes on both clustering execution time and index accuracy, using KV data from layer 10 of Llama3-8B-1048K with *mv_niah* task under 128K context. Recall@100 is used as the metric for clustering accuracy. As shown in Figure 18(b), reducing segment size from 128K (standard global k-means) to 8K results in less than 1% drop in recall, benefiting from spatial locality, while reducing index build time by 80%. However, further decreasing the segment size worsens clustering quality. Experiments across models and tasks show consistent trends. Based on this, we set the segment size to 8K as a balanced configuration.

6 RELATED WORK

LLM Serving Systems. Recent research has explored various strategies for LLM serving efficiency [3, 26, 29, 43, 51, 82, 88, 112, 123, 124]. LoongServe [99] proposes an elastic sequence parallelism scheduling for long-context requests while Helix [60] focuses on scheduling requests across heterogeneous GPU clusters. They are complementary to RetroInfer, which focuses on efficiently managing long-context KV cache. Many works [13, 27, 28, 75, 113] offload KV cache to external storage for reuse across requests. RetroInfer can be combined with these works to reduce prefilling latency.

Sparse Attention. To accelerate long-context inference, many works exploit attention sparsity to reduce computation and memory pressure. Static sparsity methods [49, 103, 104] use fixed patterns during decoding but compromise model accuracy. This motivates dynamic sparsity systems [15, 44, 89, 102, 108, 118] to heuristically select important tokens per step for better accuracy. Some works [23, 36, 53] explore vector indexes for KV cache retrieval but restrict their use to prefix caching or offline inference due to the high index construction cost. ClusterKV [54] employs clustering but lacks adaptability to varying sparsity, leading to suboptimal performance. Different from these works, RetroInfer designs wave index and wave buffer holistically for efficient KV cache management.

PyramidKV [9] and D2O [93] adapt retrieval budgets across layers but ignore sparsity variation across queries and tasks. Tactic [125] adjusts budgets via distribution fitting, though its accuracy depend on fitting precision. Our work focuses more on system-level design than on pure algorithmic improvements. Some sparse attention approaches [30, 114] require model retraining to learn dynamic sparsity, while RetroInfer is training-free. Other techniques include KV cache quantization [35, 56, 116, 117] and prefill sparsity acceleration [42, 106] are complementary to our work.

LLM Weight Sparsity. Model weight compression or offloading [6, 52, 83, 86, 107, 121] reduces the memory requirement for the LLM itself. They are orthogonal to RetroInfer, which targets for long-context scenarios where the KV cache is the memory bottleneck.

Vector Data Management. Managing vector data is an important problem in data management domains, including vector database systems [11, 34, 71, 94, 110], vector indexing and retrieval techniques [25, 57, 59, 63, 69, 72, 95, 97, 119]. Retrieval-augmented generation (RAG) [47, 55, 120] retrieves external knowledge to enhance LLMs, which is orthogonal to KV cache management in RetroInfer that focuses on the vector management inside the LLM context.

7 CONCLUSION

We presented RetroInfer, a vector storage engine for high throughput inference on long-context LLMs. To efficiently manage the KV cache and exploit attention sparsity, RetroInfer introduces a co-designed software stack integrating an attention-aware vector index (*wave index*) with a runtime control plane (*wave buffer*) for heterogeneous memory systems. The wave index improves the trade-off between attention accuracy and retrieval cost, while the wave buffer decouples attention approximation from system efficiency, enabling scalable execution across GPUs and CPUs. We demonstrate that RetroInfer not only achieves significant speedups over baselines, but also preserves model accuracy.

ACKNOWLEDGMENTS

We thank all anonymous reviewers for their insightful comments. This work is supported in part by the National Key R&D Program of China under Grant No. 2024YFB4505701.

REFERENCES

[1] 01-ai. 2024. Yi-6B-200K. https://huggingface.co/01-ai/Yi-6B-200K. Accessed: 2024-11-11.

[2] 01-ai. 2024. Yi-9B-200K. https://huggingface.co/01-ai/Yi-9B-200K. Accessed: 2024-11-11.

[3] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, Santa Clara, CA, USA, 117–134. https://www.usenix.org/conference/osdi24/presentation/agrawal

[4] Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, and Ramachandran Ramjee. 2023. SARATHI: Efficient LLM Inference by Piggybacking Decodes with Chunked Prefills. *CoRR* abs/2308.16369 (2023). https://doi.org/10.48550/ARXIV.2308.16369

[5] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Singapore, 4895–4901. https://doi.org/10.18653/V1/2023.EMNLP-MAIN.298

[6] Keivan Alizadeh, Iman Mirzadeh, Dmitry Belenko, S. Khatamifard, Minsik Cho, Carlo C. del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. 2024. LLM in a flash: Efficient Large Language Model Inference with Limited Memory. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Bangkok, Thailand, 12562–12584. https://doi.org/10.18653/V1/2024.ACL-LONG.678

[7] Anthropic. 2025. Claude. https://www.anthropic.com/claude. Accessed: 2025-08-01.

[8] Ramakrishna Bairi, Atharv Sonwane, Aditya Kanade, Vageesh D. C., Arun Iyer, Suresh Parthasarathy, Sriram K. Rajamani, Balasubramanyan Ashok, and Shashank Shet. 2024. CodePlan: Repository-Level Coding using LLMs and Planning. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 675–698. https://doi.org/10.1145/3643757

[9] Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. 2024. PyramidKV: Dynamic KV Cache Compression based on Pyramidal Information Funneling. *CoRR* abs/2406.02069 (2024). https://doi.org/10.48550/ARXIV.2406.02069

[10] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating Large Language Model Decoding with Speculative Sampling. *CoRR* abs/2302.01318 (2023). https://doi.org/10.48550/ARXIV.2302.01318

[11] Cheng Chen, Chenzhe Jin, Yunan Zhang, Sasha Podolsky, Chun Wu, Szu-Po Wang, Eric Hanson, Zhou Sun, Robert Walzer, and Jianguo Wang. 2024. SingleStore-V: An Integrated Vector Database System in SingleStore. *Proc. VLDB Endow.* 17, 12 (2024), 3772–3785. https://doi.org/10.14778/3685800.3685805

[12] Meng Chen, Kai Zhang, Zhenying He, Yinan Jing, and X. Sean Wang. 2024. RoarGraph: A Projected Bipartite Graph for Efficient Cross-Modal Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 17, 11 (2024), 2735–2749. https://doi.org/10.14778/3681954.3681959

[13] Weijian Chen, Shuibing He, Haoyang Qu, Ruidong Zhang, Siling Yang, Ping Chen, Yi Zheng, Baoxing Huai, and Gang Chen. 2025. IMPRESS: An Importance-Informed Multi-Tier Prefix KV Storage System for Large Language Model Inference. In *23rd USENIX Conference on File and Storage Technologies*. USENIX Association, Santa Clara, CA, USA, 187–201. https://www.usenix.org/conference/fast25/presentation/chen-weijian-impress

[14] Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. 2024. Do NOT Think That Much for 2+3=? On the Overthinking of o1-Like LLMs. *CoRR* abs/2412.21187 (2024). https://doi.org/10.48550/ARXIV.2412.21187

[15] Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Léon Bottou, Zhihao Jia, and Beidi Chen. 2025. MagicPIG: LSH Sampling for Efficient LLM Generation. In *The Thirteenth International Conference on Learning Representations*. OpenReview.net, Singapore. https://openreview.net/forum?id=ALzTQUgW8a

[16] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating Long Sequences with Sparse Transformers. *CoRR* abs/1904.10509 (2019). http://arxiv.org/abs/1904.10509

[17] Gonçalo M. Correia, Vlad Niculae, and André F. T. Martins. 2019. Adaptively Sparse Transformers. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*. Association for Computational Linguistics, Hong Kong, China, 2174–2184. https://doi.org/10.18653/V1/D19-1223

[18] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *The Thirty-Sixth Annual Conference on Neural Information Processing Systems*. New Orleans, LA, USA. http://papers.nips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html

[19] DeepSeek. 2025. DeepSeek-R1-Distill-Llama-8B. https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Llama-8B. Accessed: 2025-08-01.

[20] DeepSeek. 2025. DeepSeek-R1-Distill-Qwen-7B. https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-7B. Accessed: 2025-08-01.

[21] DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *CoRR* abs/2501.12948 (2025). https://doi.org/10.48550/ARXIV.2501.12948

[22] Yichuan Deng, Zhao Song, Jing Xiong, and Chiwun Yang. 2024. How Sparse Attention Approximates Exact Attention? Your Attention is Naturally n^C-Sparse. *arXiv preprint arXiv:2404.02690* (2024).

[23] Yangshen Deng, Zhengxin You, Long Xiang, Qilong Li, Peiqi Yuan, Zhaoyang Hong, Yitao Zheng, Wanting Li, Runzhong Li, Haotian Liu, Kyriakos Mouratidis, Man Lung Yiu, Huan Li, Qiaomu Shen, Rui Mao, and Bo Tang. 2025. AlayaDB: The Data Foundation for Efficient and Effective Long-context LLM Inference. In *Companion of the 2025 International Conference on Management of Data, SIGMOD/PODS 2025*. ACM, Berlin, Germany, 364–377. https://doi.org/10.1145/3722212.3724428

[24] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. *CoRR* abs/2401.08281 (2024). https://doi.org/10.48550/ARXIV.2401.08281

[25] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph. *Proc. VLDB Endow.* 12, 5 (2019), 461–474. https://doi.org/10.14778/3303753.3303754

[26] Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. 2024. ServerlessLLM: Low-Latency Serverless Inference for Large Language Models. In *18th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, Santa Clara, CA, USA, 135–153. https://www.usenix.org/conference/osdi24/presentation/fu

[27] Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. 2024. Cost-Efficient Large Language Model Serving for Multi-turn Conversations with CachedAttention. In *Proceedings of the 2024 USENIX Annual Technical Conference*. USENIX Association, Santa Clara, CA, USA, 111–126. https://www.usenix.org/conference/atc24/presentation/gao-bin-cost

[28] Shiwei Gao, Youmin Chen, and Jiwu Shu. 2025. Fast State Restoration in LLM Serving with HCache. In *Proceedings of the Twentieth European Conference on Computer Systems*. ACM, Rotterdam, The Netherlands, 128–143. https://doi.org/10.1145/3689031.3696072

[29] Shihong Gao, Xin Zhang, Yanyan Shen, and Lei Chen. 2025. Apt-Serve: Adaptive Request Scheduling on Hybrid Cache for Scalable LLM Inference Serving. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 130:1–130:28. https://doi.org/10.1145/3725394

[30] Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Hayden Kwok-Hay So, Ting Cao, Fan Yang, and Mao Yang. 2024. SeerAttention: Learning Intrinsic Sparse Attention in Your LLMs. *CoRR* abs/2410.13276 (2024). https://doi.org/10.48550/ARXIV.2410.13276

[31] Google. 2025. Gemini. https://gemini.google.com/app. Accessed: 2025-08-01.

[32] gradientai. 2024. Llama-3-8B-Instruct-Gradient-1048k. https://huggingface.co/gradientai/Llama-3-8B-Instruct-Gradient-1048k. Accessed: 2024-10-29.

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

[34] Rentong Guo, Xiaofan Luan, Long Xiang, Xiao Yan, Xiaomeng Yi, Jigao Luo, Qianya Cheng, Weizhi Xu, Jiarui Luo, Frank Liu, Zhenshan Cao, Yanliang Qiao, Ting Wang, Bo Tang, and Charles Xie. 2022. Manu: A Cloud Native Vector Database Management System. *Proc. VLDB Endow.* 15, 12 (2022), 3548–3561. https://doi.org/10.14778/3554821.3554843

[35] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. In *The Thirty-Eighth Annual Conference on Neural Information Processing Systems*. Vancouver, BC, Canada. http://papers.nips.cc/paper_files/paper/2024/hash/028fcbcf85435d39a40c4d61b42c99a4-Abstract-Conference.html

[36] Coleman Richard Charles Hooper, Sehoon Kim, Hiva Mohammadzadeh, Monishwaran Maheswaran, Sebastian Zhao, June Paik, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. 2025. Squeezed Attention: Accelerating Long Context Length LLM Inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Vienna, Austria, 32631–32652. https://aclanthology.org/2025.acl-long.1568/

[37] Kurt Hornik, Ingo Feinerer, Martin Kober, and Christian Buchta. 2012. Spherical k-means clustering. *Journal of statistical software* 50 (2012), 1–22.

[38] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. RULER: What's the Real Context Size of Your Long-Context Language Models? *CoRR* abs/2404.06654

(2024). https://doi.org/10.48550/ARXIV.2404.06654

[39] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*. ACM, Dallas, Texas, USA, 604–613. https://doi.org/10.1145/276698.276876

[40] InfiniGen. 2024. InfiniGen Code. https://github.com/snu-comparch/InfiniGen. Accessed: 2025-04-01.

[41] Johan Ludwig William Valdemar Jensen. 1906. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta mathematica* 30, 1 (1906), 175–193.

[42] 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. 2024. 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*. Vancouver, BC, Canada. http://papers.nips.cc/paper_files/paper/2024/hash/5dfbe6f5671e82c76841ba687a8a9ecb-Abstract-Conference.html

[43] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*. ACM, Koblenz, Germany, 611–626. https://doi.org/10.1145/3600006.3613165

[44] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. 2024. InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management. In *18th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, Santa Clara, CA, 155–172. https://www.usenix.org/conference/osdi24/presentation/lee

[45] Viktor Leis. 2024. LeanStore: A High-Performance Storage Engine for NVMe SSDs. *Proc. VLDB Endow.* 17, 12 (2024), 4536–4545. https://doi.org/10.14778/3685800.3685915

[46] Viktor Leis, Adnan Alhomssi, Tobias Ziegler, Yannick Loeck, and Christian Dietrich. 2023. Virtual-Memory Assisted Buffer Management. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 7:1–7:25. https://doi.org/10.1145/3588687

[47] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *The Thirty-fourth Annual Conference on Neural Information Processing Systems*. virtual. https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[48] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as A Database Interface? A Big Bench for Large-Scale Database Grounded Text-to-SQLs. In *The Thirty-Seventh Annual Conference on Neural Information Processing Systems*. New Orleans, LA, USA. http://papers.nips.cc/paper_files/paper/2023/hash/83fc8fab1710363050bbd1d4b8cc0021-Abstract-Datasets_and_Benchmarks.html

[49] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. SnapKV: LLM Knows What You are Looking for Before Generation. In *The Thirty-Eighth Annual Conference on Neural Information Processing Systems*. Vancouver, BC, Canada. http://papers.nips.cc/paper_files/paper/2024/hash/28ab418242603e0f7323e54185d19bde-Abstract-Conference.html

[50] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. AlpaServe: Statistical Multiplexing with Model Parallelism for Deep Learning Serving. In *17th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, 663–679. https://www.usenix.org/conference/osdi23/presentation/li-zhouhan

[51] Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. 2024. Parrot: Efficient Serving of LLM-based Applications with Semantic Variable. In *18th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, Santa Clara, CA, USA, 929–945. https://www.usenix.org/conference/osdi24/presentation/lin-chaofan

[52] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. In *Proceedings of the Seventh Annual Conference on Machine Learning and Systems*. mlsys.org, Santa Clara, CA, USA. https://proceedings.mlsys.org/paper_files/paper/2024/hash/42a452cbafa9dd64e9ba4aa95cc1ef21-Abstract-Conference.html

[53] 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, and Lili Qiu. 2024. RetrievalAttention: Accelerating Long-Context LLM Inference via Vector Retrieval. *CoRR* abs/2409.10516 (2024). https://doi.org/10.48550/ARXIV.2409.10516

[54] Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. 2025. ClusterKV: Manipulating LLM KV Cache in Semantic Space for Recallable Compression. In *62nd ACM/IEEE Design Automation Conference*. IEEE, San Francisco, CA, USA, 1–7. https://doi.org/10.1109/DAC63849.2025.11132479

[55] Shige Liu, Zhifang Zeng, Li Chen, Adil Ainihaer, Arun Ramasami, Songting Chen, Yu Xu, Mingxi Wu, and Jianguo Wang. 2025. TigerVector: Supporting Vector Search in Graph Databases for Advanced RAGs. In *Companion of the 2025 International Conference on Management of Data*. ACM, Berlin, Germany, 553–565. https://doi.org/10.1145/3722212.3724456

[56] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen (Henry) Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache. In *Forty-first International Conference on Machine Learning*. OpenReview.net, Vienna, Austria. https://openreview.net/forum?id=L057s2Rq8O

[57] Kejing Lu, Mineichi Kudo, Chuan Xiao, and Yoshiharu Ishikawa. 2021. HVS: Hierarchical Graph Structure Based on Voronoi Diagrams for Solving Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 15, 2 (2021), 246–258. https://doi.org/10.14778/3489496.3489506

[58] MagicPIG. 2024. MagicPIG Code. https://github.com/Infini-AI-Lab/MagicPIG. Accessed: 2025-04-01.

[59] Yury A. Malkov and Dmitry A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2020), 824–836. https://doi.org/10.1109/TPAMI.2018.2889473

[60] Yixuan Mei, Yonghao Zhuang, Xupeng Miao, Juncheng Yang, Zhihao Jia, and Rashmi Vinayak. 2025. Helix: Serving Large Language Models over Heterogeneous GPUs and Network via Max-Flow. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. ACM, Rotterdam, The Netherlands, 586–602. https://doi.org/10.1145/3669940.3707215

[61] Meta. 2024. Llama-3.1-8B-Instruct. https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct. Accessed: 2024-09-25.

[62] Meta. 2025. The Llama 4 herd: The beginning of a new era of natively multimodal AI innovation. https://ai.meta.com/blog/llama-4-multimodal-intelligence/. Accessed: 2025-04-05.

[63] Jason Mohoney, Anil Pacaci, Shihabur Rahman Chowdhury, Ali Mousavi, Ihab F. Ilyas, Umar Farooq Minhas, Jeffrey Pound, and Theodoros Rekatsinas. 2023. High-Throughput Vector Similarity Search in Knowledge Graphs. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 197:1–197:25. https://doi.org/10.1145/3589777

[64] Jiaqi Mu and Pramod Viswanath. 2018. All-but-the-Top: Simple and Effective Postprocessing for Word Representations. In *The Sixth International Conference on Learning Representations*. OpenReview.net, Vancouver, BC, Canada. https://openreview.net/forum?id=HkuGJ3kCb

[65] Ansong Ni, Jeevana Priya Inala, Chenglong Wang, Alex Polozov, Christopher Meek, Dragomir Radev, and Jianfeng Gao. 2023. Learning Math Reasoning from Self-Sampled Correct and Partially-Correct Solutions. In *The Eleventh International Conference on Learning Representations*. OpenReview.net, Kigali, Rwanda. https://openreview.net/forum?id=4D4TSJE6-K

[66] NVIDIA. 2020. NVIDIA A100 Tensor Core GPU. https://www.nvidia.com/en-us/data-center/a100/. Accessed: 2025-04-01.

[67] NVIDIA. 2020. NVIDIA RTX A6000 Graphics Card. https://www.nvidia.com/en-us/products/workstations/rtx-a6000/. Accessed: 2025-10-01.

[68] Art of Problem Solving. 2024. AIME Problems and Solutions. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions. Accessed: 2025-08-01.

[69] Hiroyuki Ootomo, Akira Naruse, Corey Nolet, Ray Wang, Tamas Feher, and Yong Wang. 2024. CAGRA: Highly Parallel Graph Construction and Approximate Nearest Neighbor Search for GPUs. In *40th IEEE International Conference on Data Engineering*. IEEE, Utrecht, The Netherlands, 4236–4247. https://doi.org/10.1109/ICDE60146.2024.00323

[70] OpenAI. 2025. ChatGPT. https://chat.chatbotapp.ai/. Accessed: 2025-08-01.

[71] James Jie Pan, Jianguo Wang, and Guoliang Li. 2024. Survey of vector database management systems. *VLDB J.* 33, 5 (2024), 1591–1615. https://doi.org/10.1007/S00778-024-00864-X

[72] Liana Patel, Peter Kraft, Carlos Guestrin, and Matei Zaharia. 2024. ACORN: Performant and Predicate-Agnostic Search Over Vector Embeddings and Structured Data. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 120. https://doi.org/10.1145/3654923

[73] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently Scaling Transformer Inference. In *Proceedings of the Sixth Conference on Machine Learning and Systems*. mlsys.org, Miami, FL, USA. https://proceedings.mlsys.org/paper_files/paper/2023/hash/c4be71ab8d24cdfb45e3d06dbfca2780-Abstract-mlsys2023.html

[74] PQCache. 2024. PQCache. https://github.com/HugoZHL/PQCache. Accessed: 2025-04-01.

[75] Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 2025. Mooncake: Trading More Storage for Less Computation - A KVCache-centric Architecture for Serving

LLM Chatbot. In *23rd USENIX Conference on File and Storage Technologies*. USENIX Association, Santa Clara, CA, USA, 155–170. https://www.usenix.org/conference/fast25/presentation/qin

[76] Quest. 2024. Quest Code. https://github.com/mit-han-lab/Quest. Accessed: 2025-04-01.

[77] Qwen. 2024. Qwen2.5-72B-Instruct. https://huggingface.co/Qwen/Qwen2.5-72B-Instruct. Accessed: 2025-01-12.

[78] Qwen. 2024. Qwen2.5-7B-Instruct. https://huggingface.co/Qwen/Qwen2.5-7B-Instruct. Accessed: 2025-01-12.

[79] Parikshit Ram and Alexander G. Gray. 2012. Maximum inner-product search using cone trees. In *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Beijing, China, 931–939. https://doi.org/10.1145/2339530.2339677

[80] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. GPQA: A Graduate-Level Google-Proof Q&A Benchmark. *CoRR* abs/2311.12022 (2023). https://doi.org/10.48550/ARXIV.2311.12022

[81] Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. 2024. SparQ Attention: Bandwidth-Efficient LLM Inference. In *Forty-first International Conference on Machine Learning*. OpenReview.net, Vienna, Austria. https://openreview.net/forum?id=OS5dqxmmtl

[82] Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E. Gonzalez, and Ion Stoica. 2024. Fairness in Serving Large Language Models. In *18th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, Santa Clara, CA, USA, 965–988. https://www.usenix.org/conference/osdi24/presentation/sheng

[83] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. In *Fortieth International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Vol. 202. PMLR, Honolulu, Hawaii, USA, 31094–31116. https://proceedings.mlr.press/v202/sheng23a.html

[84] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search (MIPS). In *The Twenty-eighth Annual Conference on Neural Information Processing Systems*. Montreal, Quebec, Canada, 2321–2329. https://proceedings.neurips.cc/paper/2014/hash/310ce61c90f3a46e340ee8257bc70e93-Abstract.html

[85] Josef Sivic and Andrew Zisserman. 2003. Video Google: A Text Retrieval Approach to Object Matching in Videos. In *The 9th IEEE International Conference on Computer Vision*. IEEE Computer Society, Nice, France, 1470–1477. https://doi.org/10.1109/ICCV.2003.1238663

[86] Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2024. PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*. ACM, Austin, TX, USA, 590–606. https://doi.org/10.1145/3694715.3695964

[87] Jianlin Su, Murtadha H. M. Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. RoFormer: Enhanced transformer with Rotary Position Embedding. *Neurocomputing* 568 (2024), 127063. https://doi.org/10.1016/J.NEUCOM.2023.127063

[88] Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. 2024. Llumnix: Dynamic Scheduling for Large Language Model Serving. In *18th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, Santa Clara, CA, USA, 173–191. https://www.usenix.org/conference/osdi24/presentation/sun-biao

[89] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. QUEST: Query-Aware Sparsity for Efficient Long-Context LLM Inference. In *Forty-first International Conference on Machine Learning*. OpenReview.net, Vienna, Austria. https://openreview.net/forum?id=KzACYw0MTV

[90] Philippe Tillet, Hsiang-Tsung Kung, and David D. Cox. 2019. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*. ACM, Phoenix, AZ, USA, 10–19. https://doi.org/10.1145/3315508.3329973

[91] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *The Thirty-first Annual Conference on Neural Information Processing Systems*. 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[92] John von Neumann. 1993. First draft of a report on the EDVAC. *IEEE Annals of the History of Computing* 15, 4 (1993), 27–75. https://doi.org/10.1109/85.238389

[93] Zhongwei Wan, Xinjian Wu, Yu Zhang, Yi Xin, Chaofan Tao, Zhihong Zhu, Xin Wang, Siqi Luo, Jing Xiong, and Mi Zhang. 2024. D2O: Dynamic Discriminative Operations for Efficient Generative Inference of Large Language Models. *CoRR* abs/2406.13035 (2024). https://doi.org/10.48550/ARXIV.2406.13035

[94] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. 2021. Milvus: A Purpose-Built Vector Data Management System. In *Proceedings of the 2021 International Conference on Management of Data*. ACM, Virtual Event, China, 2614–2627. https://doi.org/10.1145/3448016.3457550

[95] Runhui Wang and Dong Deng. 2020. DeltaPQ: Lossless Product Quantization Code Compression for High Dimensional Similarity Search. *Proc. VLDB Endow.* 13, 13 (2020), 3603–3616. https://doi.org/10.14778/3424573.3424580

[96] Zehuan Wang, Yingcan Wei, Minseok Lee, Matthias Langer, Fan Yu, Jie Liu, Shijie Liu, Daniel G. Abel, Xu Guo, Jianbing Dong, Ji Shi, and Kunlun Li. 2022. Merlin HugeCTR: GPU-accelerated Recommender System Training and Inference. In *Sixteenth ACM Conference on Recommender Systems*. ACM, Seattle, WA, USA, 534–537. https://doi.org/10.1145/3523227.3547405

[97] Jiuqi Wei, Botao Peng, Xiaodong Lee, and Themis Palpanas. 2024. DET-LSH: A Locality-Sensitive Hashing Scheme with Dynamic Encoding Tree for Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 17, 9 (2024), 2241–2254. https://doi.org/10.14778/3665844.3665854

[98] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *The Thirty-Sixth Annual Conference on Neural Information Processing Systems*. New Orleans, LA, USA. http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html

[99] Bingyang Wu, Shengyu Liu, Yinmin Zhong, Peng Sun, Xuanzhe Liu, and Xin Jin. 2024. LoongServe: Efficiently Serving Long-Context Large Language Models with Elastic Sequence Parallelism. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*. ACM, Austin, TX, USA, 640–654. https://doi.org/10.1145/3694715.3695948

[100] Wei Wu, Zhuoshi Pan, Chao Wang, Liyi Chen, Yunchu Bai, Kun Fu, Zheng Wang, and Hui Xiong. 2024. TokenSelect: Efficient Long-Context Inference and Length Extrapolation for LLMs via Dynamic Token-Level KV Cache Selection. *CoRR* abs/2411.02886 (2024). https://doi.org/10.48550/ARXIV.2411.02886

[101] Wenhao Wu, Yizhong Wang, Guangxuan Xiao, Hao Peng, and Yao Fu. 2025. Retrieval Head Mechanistically Explains Long-Context Factuality. In *The Thirteenth International Conference on Learning Representations*. OpenReview.net, Singapore. https://openreview.net/forum?id=EytBpUGB1Z

[102] Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. 2024. InfLLM: Training-Free Long-Context Extrapolation for LLMs with an Efficient Context Memory. In *The Thirty-Eighth Annual Conference on Neural Information Processing Systems*. Vancouver, BC, Canada. http://papers.nips.cc/paper_files/paper/2024/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html

[103] Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. 2025. DuoAttention: Efficient Long-Context LLM Inference with Retrieval and Streaming Heads. In *The Thirteenth International Conference on Learning Representations*. OpenReview.net, Singapore. https://openreview.net/forum?id=cFu7ze7xUm

[104] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient Streaming Language Models with Attention Sinks. In *The Twelfth International Conference on Learning Representations*. OpenReview.net, Vienna, Austria. https://openreview.net/forum?id=NG7sS51zVF

[105] Fangyuan Xu, Tanya Goyal, and Eunsol Choi. 2024. Recycled Attention: Efficient inference for long-context language models. *CoRR* abs/2411.05787 (2024). https://doi.org/10.48550/ARXIV.2411.05787

[106] Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. 2025. XAttention: Block Sparse Attention with Antidiagonal Scoring. In *Forty-second International Conference on Machine Learning*. OpenReview.net, Vancouver, BC, Canada. https://openreview.net/forum?id=KG6aBfGi6e

[107] Zhenliang Xue, Yixin Song, Zeyu Mi, Le Chen, Yubin Xia, and Haibo Chen. 2024. PowerInfer-2: Fast Large Language Model Inference on a Smartphone. *CoRR* abs/2406.06282 (2024). https://doi.org/10.48550/ARXIV.2406.06282

[108] Shang Yang, Junxian Guo, Haotian Tang, Qinghao Hu, Guangxuan Xiao, Jiaming Tang, Yujun Lin, Zhijian Liu, Yao Lu, and Song Han. 2025. LServe: Efficient Long-sequence LLM Serving with Unified Sparse Attention. *CoRR* abs/2502.14866 (2025). https://doi.org/10.48550/ARXIV.2502.14866

[109] Shu Yang, Junchao Wu, Xin Chen, Yunze Xiao, Xinyi Yang, Derek F. Wong, and Di Wang. 2025. Understanding Aha Moments: from External Observations to Internal Mechanisms. *CoRR* abs/2504.02956 (2025). https://doi.org/10.48550/ARXIV.2504.02956

[110] Wen Yang, Tao Li, Gai Fang, and Hong Wei. 2020. PASE: PostgreSQL Ultra-High-Dimensional Approximate Nearest Neighbor Search Extension. In *Proceedings of the 2020 International Conference on Management of Data*. ACM, online conference, Portland, OR, USA, 2241–2253. https://doi.org/10.1145/3318464.3386131

[111] Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. 2025. FlashInfer: Efficient and Customizable Attention Engine for LLM Inference Serving. *CoRR* abs/2501.01005 (2025). https://doi.org/10.48550/ARXIV.2501.01005

[112] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A Distributed Serving System for Transformer-Based

Generative Models. In *16th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, Carlsbad, CA, USA, 521–538. https://www.usenix.org/conference/osdi22/presentation/yu

[113] Lingfan Yu, Jinkun Lin, and Jinyang Li. 2025. Stateful Large Language Model Serving with Pensieve. In *Proceedings of the Twentieth European Conference on Computer Systems*. ACM, Rotterdam, The Netherlands, 144–158. https://doi.org/10.1145/3689031.3696086

[114] 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. 2025. Native Sparse Attention: Hardware-Aligned and Natively Trainable Sparse Attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Vienna, Austria, 23078–23097. https://aclanthology.org/2025.acl-long.1126/

[115] Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. 2025. PQCache: Product Quantization-based KVCache for Long Context LLM Inference. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 201:1–201:30. https://doi.org/10.1145/3725338

[116] Tianyi Zhang, Jonah Yi, Zhaozhuo Xu, and Anshumali Shrivastava. 2024. KV Cache is 1 Bit Per Channel: Efficient Large Language Model Inference with Coupled Quantization. In *The Thirty-Eighth Annual Conference on Neural Information Processing Systems*. Vancouver, BC, Canada. http://papers.nips.cc/paper_files/paper/2024/hash/05d6b5b6901fb57d2c287e1d3ce6d63c-Abstract-Conference.html

[117] Yanqi Zhang, Yuwei Hu, Runyuan Zhao, John C. S. Lui, and Haibo Chen. 2024. Unifying KV Cache Compression for Large Language Models with LeanKV. *CoRR* abs/2412.03131 (2024). https://doi.org/10.48550/ARXIV.2412.03131

[118] 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. 2023. H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. In *The Thirty-Seventh Annual Conference on Neural Information Processing Systems*. New Orleans, LA, USA. http://papers.nips.cc/paper_files/paper/2023/hash/6ceefa7b15572587b78ecfcebb2827f8-Abstract-Conference.html

[119] Xi Zhao, Yao Tian, Kai Huang, Bolong Zheng, and Xiaofang Zhou. 2023. Towards Efficient Index Construction and Approximate Nearest Neighbor Search in High-Dimensional Spaces. *Proc. VLDB Endow.* 16, 8 (2023), 1979–1991. https://doi.org/10.14778/3594512.3594527

[120] Xinyang Zhao, Xuanhe Zhou, and Guoliang Li. 2024. Chat2Data: An Interactive Data Analysis System with RAG, Vector Databases and LLMs. *Proc. VLDB Endow.* 17, 12 (2024), 4481–4484. https://doi.org/10.14778/3685800.3685905

[121] Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. 2024. Atom: Low-Bit Quantization for Efficient and Accurate LLM Serving. In *Proceedings of the Seventh Annual Conference on Machine Learning and Systems*. mlsys.org, Santa Clara, CA, USA. https://proceedings.mlsys.org/paper_files/paper/2024/hash/5edb57c05c81d04beb716ef1d542fe9e-Abstract-Conference.html

[122] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. 2022. Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning. In *16th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, Carlsbad, CA, USA, 559–578. https://www.usenix.org/conference/osdi22/presentation/zheng-lianmin

[123] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. 2024. SGLang: Efficient Execution of Structured Language Model Programs. In *The Thirty-Eighth Annual Conference on Neural Information Processing Systems*. Vancouver, BC, Canada. http://papers.nips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html

[124] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. In *18th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, Santa Clara, CA, USA, 193–210. https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin

[125] Kan Zhu, Tian Tang, Qinyu Xu, Yile Gu, Zhichen Zeng, Rohan Kadekodi, Liangyu Zhao, Ang Li, Arvind Krishnamurthy, and Baris Kasikci. 2025. Tactic: Adaptive Sparse Attention with Clustering and Distribution Fitting for Long-Context LLMs. *CoRR* abs/2502.12216 (2025). https://doi.org/10.48550/ARXIV.2502.12216

[126] Tobias Ziegler, Carsten Binnig, and Viktor Leis. 2022. ScaleStore: A Fast and Cost-Efficient Storage Engine using DRAM, NVMe, and RDMA. In *Proceedings of the 2022 International Conference on Management of Data*. ACM, Philadelphia, PA, USA, 685–699. https://doi.org/10.1145/3514221.3526187