Scaling Attention Beyond GPUs for LLM Inference

Weishu Deng
The University of Texas at Arlington
Arlington, USA
weishu.deng@mavs.uta.edu

Yujie Yang
The University of Texas at Arlington
Arlington, USA
yxy1115@mavs.uta.edu

Peiran Du
The University of Texas at Arlington
Arlington, USA
pxd2350@mavs.uta.edu

Lingfeng Xiang
University of Texas at Arlington
Arlington, USA
lingfeng.xiang@mavs.uta.edu

Zhen Lin
University of Texas at Arlington
Arlington, USA
zxl5722@mavs.uta.edu

Chen Zhong
University of texas at arlington
Arlington, USA
chen.zhong@mavs.uta.edu

Faraz Ahmed
Hewlett Packard Labs
Milpitas, USA
faraz.ahmed@hpe.com

Lianjie Cao
Hewlett Packard Enterprise
Milpitas, USA
lianjie.cao@hpe.com

Puneet Sharma
Hewlett Packard Labs
Milpitas, USA
puneet.sharma@hpe.com

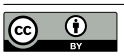
Song Jiang
University of Texas at Arlington
Arlington, USA
song.jiang@uta.edu

Hui Lu
The University of Texas at Arlington
Arlington, USA
hui.lu@uta.edu

Jia Rao
The University of Texas at Arlington
Arlington, USA
jia.rao@uta.edu

Abstract

Scaling inference for large language models is increasingly constrained by limited GPU memory, primarily due to the expanding intermediate states (KV caches) required for long-context generation and multi-user workloads. Once the KV cache exceeds the capacity of high-bandwidth memory, it must be offloaded to host memory and reloaded on demand, a workflow severely bottlenecked by the CPU–GPU interconnect, typically PCIe. Existing approaches exploiting offload KV caches to CPU memory and selectively reload partial segments for attention computation often underutilize CPU compute resources and suffer from accuracy degradation. We present Beyond, a drop-in runtime that integrates a smart offloading scheme to selectively identify and retain salient KV entries across continuous decoding sessions, together with a hybrid CPU–GPU attention mechanism for scalable inference. Beyond executes dense attention over recent KV entries stored in GPU memory while performing parallel, per-head sparse attention on salient contextual KV entries residing in CPU memory. The outputs are fused efficiently through a log-sum-exp scheme. During the bandwidth-constrained decoding phase, oversized KV caches are processed cooperatively by the aggregated CPU and GPU memory bandwidth, with only minimal PCIe data movement. Experiments across diverse models and workloads demonstrate that Beyond improves scalability, supports longer sequences and larger batch sizes, and outperforms existing sparse attention baselines in both efficiency and accuracy—all on commodity GPU hardware.



HPDC '26, Cleveland, OH, USA

ACM ISBN 979-8-4007-2640-8/26/07
https://doi.org/10.1145/3806645.3807596

ACM Reference Format:
Weishu Deng, Yujie Yang, Peiran Du, Lingfeng Xiang, Zhen Lin, Chen Zhong, Faraz Ahmed, Lianjie Cao, Puneet Sharma, Song Jiang, Hui Lu, and Jia Rao. 2026. Scaling Attention Beyond GPUs for LLM Inference. In *The 35th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '26), July 13–16, 2026, Cleveland, OH, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3806645.3807596

1 Introduction

Transformer-based large language models (LLMs) have shown remarkable capabilities in a wide range of applications, including chatbots [4], code assistance [15], text generation and summarization [14]. Recent studies [6, 20] show that providing step-by-step Chain-of-Thought (CoT) enhances LLM's ability to solve complex tasks involving reasoning and mathematical problem-solving. Such inference-time scaling techniques, including CoT prompting, sampling with majority voting [10], and tree-search-based sequence generation [32], demand substantial transient states, typically in the form of key-value (KV) caches, during LLM inference. The size of the KV cache scales with input and output sequence lengths, as well as batch size, and may exceed the size of the model weights, becoming the primary contributor to memory consumption in LLM serving. As researchers strive to make LLMs accessible to small institutions and individual users, efficiently serving LLM inferences on single or multiple commodity GPUs has become increasingly important.

While many pre-trained, open-source LLMs can fit their model weights within the memory capacity of commodity GPUs such as the NVIDIA A6000, the KV cache can grow without bound and eventually exhaust available GPU memory. To enable long-sequence generation and support concurrent inference requests, modern LLM serving systems [8] incorporate mechanisms for offloading KV caches to CPU memory, while other approaches leverage data compression [19, 22]. Unlike model weights, which must be fully

loaded and applied to transform input representations throughout inference, the intermediate results from an LLM's attention layers (keys and values) can either be stored in the KV cache for reuse or recomputed for each newly generated token. The key challenge in KV cache management is efficiently reusing stored key-value entries once the cache exceeds available GPU memory.

Although a significant portion of the KV cache can be offloaded to CPU memory, it is generally believed that attention computation on the KV cache should be performed on the GPU to fully utilize its computational power. Full attention-based KV management [16] requires the entire KV cache stored in CPU memory to be transferred to the GPU for attention computation. However, loading the KV cache over the PCIe bus – limited to up to 32 GB/s on commodity GPUs with 16 PCIe 4.0 lanes – introduces a performance bottleneck and disrupts attention computation on the GPU. In contrast, sparse attention [31] attends a subset of KV cache entries with the highest attention scores and only keeps these entries in GPU memory. While sparse attention reduces the need to frequently load KV cache from CPU memory, it depends on accurately identifying important KV entries. Excluding critical KV entries due to a reduced attention scope can degrade the accuracy of LLM inference. As LLMs scale and content generation becomes more complex, the demand for intermediate data storage increasingly outpaces available GPU memory, resulting in reduced inference throughput and potential accuracy degradation.

In this paper, we seek to develop an attention mechanism for LLM inference that simultaneously achieves good throughput, high accuracy, and (almost) unlimited [1] scalability to long sequence outputs and large batch sizes. This paper seeks to answer a fundamental question – *Given an LLM model, what is the highest performance that can be achieved on a single consumer-class GPU without undermining model accuracy or requiring algorithmic changes to the attention mechanism?* Our work is motivated by two important observations. *First*, unlike LLM training, LLM inference often exhibits low operational density [26] – defined as the number of operations per byte of memory traffic – offering opportunities for offloading a portion of attention computation to the CPU without incurring drastic performance degradations. The performance discrepancies between the computational power (i.e., TFLOPS) of CPUs and consumer-class GPUs are still significant, while the gap in memory bandwidth is much narrower. For example, the gap in TFLOPS for FP16 in the Intel Xeon Gold 6430 processor and NVIDIA A6000, a widely available and adopted GPU for LLM serving, is at least an order of magnitude (1.229 TFLOPS vs. 38.7 TFLOPS). In contrast, the CPUs can achieve up to 500 GB/s memory bandwidth with all 32 DDR slots fully populated, whereas the NVIDIA RTX A6000 GPU's GDDR6 memory can only deliver up to 768 GB/s. *Second*, CPUs have advanced control logic that efficiently handles complex sparsification algorithms with intricate control flow, helping close the performance gap in attention computation.

Figure 1 shows a roofline model of attention operators in LLM inference [26]. The *decode* and majority *append* stages are memory-bound, where the CPU can potentially keep up with the GPU during attention computation. Ideally, optimal attention performance is

[1]The scale of LLM inference due to KV caching is not limited by the capacity of GPU memory.

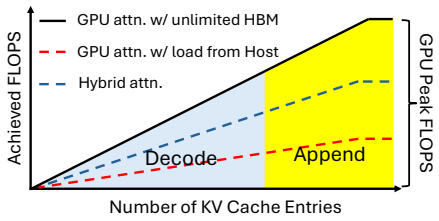


Figure 1: Roofline model of attention stages in LLM serving.

achieved via *GPU-only attention* with unlimited GPU memory, always underneath the peak GPU bandwidth ceiling. *GPU attention with CPU offloading* (represented by the red dotted line) allows the KV cache size to exceed GPU memory capacity but suffers from performance degradations due to the transfer of the KV cache between the CPU and GPU over PCIe, leaving the GPU idle during the transfer. Alternatively, this work explores *hybrid attention* to leverage the combined bandwidth of CPU and GPU memory, achieving the aggregated computational power of both devices.

We propose Beyond, a hybrid CPU-GPU attention mechanism that partitions computation across devices with distinct runtime key-value management. To streamline attention computation on the two heterogeneous devices, Beyond performs full and sparse attention on the GPU and CPU, respectively. Full attention operates on KV entries stored in GPU memory, corresponding to recently generated tokens, whereas sparse attention selectively focuses on high-scoring KV entries residing in CPU memory. The attention output on the CPU is then transferred to the GPU, where the two partial attention outputs are aggregated using log-sum-exp fusion. Furthermore, Beyond leverages multiple CPU threads to perform per-head sparse attention over only the most critical KV entries, effectively preserving contextual information during inference and maintaining high accuracy.

Beyond's hybrid attention offers two advantages. *First*, CPU attention not only contributes computationally to the overall attention computation but also eliminates the cross-device data transfer bottleneck – the partial attention results are orders of magnitude smaller than raw KV caches and can be efficiently transferred to GPU memory via zero-copy transfer. *Second*, Beyond's head-granular sparse attention more accurately identifies critical KV entries compared to the widely used Top-K sparse attention. Experimental results with three representative LLMs show that 1) Beyond scales to much larger LLMs, longer sequence outputs, larger batch sizes on a single or multiple commodity GPUs; 2) Beyond achieves accuracy close-to or better than full attention; 3) Beyond is implemented as a drop-in replacement for existing attention layers and can be seamlessly integrated into any LLM with minimal engineering effort.

2 Background and Motivation

The attention mechanism is central to LLM, allowing tokens to attend to previously computed key–value (KV) representations. During autoregressive inference, KV states produced at each attention layer are cached and reused to avoid redundant computation.

LLM inference can be divided into three stages—*prefill*, *append*, and *decode*—based on the query-to-KV ratio. In the prefill stage,

each query attends only to newly generated KV states, yielding a 1:1 ratio and high arithmetic intensity, which makes the stage compute-bound. In contrast, the append and decode stages involve few queries attending to a large KV cache, resulting in low KV reuse and increased memory traffic, and are therefore primarily memory-bandwidth-bound.

2.1 Standard Full Attention

Standard full attention incorporates all KV entries when computing attention, ensuring that no relevant context is omitted. The decode and append processes follow the standard formulation described in [26]. During inference, incoming hidden states are projected into query, key, and value tensors and reshaped to $\mathbb{R}^{h\times N\times d}$, where h is the number of attention heads, N is the length of the incoming sequence, and d is the per-head hidden dimension.

The newly generated keys and values are concatenated with the cached tensors, producing an expanded KV cache of length $N' = N + N_{\text{cache}}$:

$$K = \text{concat}(K_{\text{cache}}, K), \quad V = \text{concat}(V_{\text{cache}}, V) \in \mathbb{R}^{h\times N'\times d}.$$

Attention scores are computed independently per head, followed by a weighted aggregation of values:

$$S = QK^T, \quad A = \text{softmax}(S) \in \mathbb{R}^{h\times N\times N'}, \quad O = AV \in \mathbb{R}^{h\times N\times d}.$$

After computation, queries are discarded, while keys and values are retained for reuse. In the decode and append stages, only N output tokens are produced, yet all N' KV pairs must be repeatedly accessed for attention computation. Since typically $N \ll N'$, the workload becomes memory-bandwidth-bound, leaving compute resources underutilized. As N' exceeds on-device memory capacity, KV offloading to host memory is required; however, the substantially lower PCIe bandwidth compared to GPU HBM further exacerbates the attention bottleneck.

2.2 Sparse Attention

In practice, many KV entries receive negligible softmax weights and can be safely omitted with minimal impact on model accuracy. By attending only to salient KV entries, sparse attention reduces computation, memory footprint, and data movement overhead. Existing approaches broadly fall into two categories: *static* and *dynamic* sparse attention.

Static sparse attention selects KV entries using fixed heuristics derived from empirical attention patterns and retains only these entries in GPU memory. Prior studies show that KV importance is strongly correlated with token position: Longformer [2] emphasizes recent tokens via a sliding window, while StreamLLM [23] identifies early tokens ("attention sinks") as globally important. In practice, static methods combine a recent window with a small global window, enabling sparsification without runtime overhead.

Dynamic sparse attention identifies salient KV entries at runtime, enabling context-aware adaptation and reducing the risk of discarding important tokens. H2O [31] exploits temporal locality using an LRU-based policy, Quest [17] applies block-wise scanning to balance latency and accuracy, and Infinigen [9] predicts future KV accesses asynchronously based on prior queries. While more flexible, dynamic approaches introduce additional computation and control overhead that can offset sparsification gains. Most methods also fix the number of selected entries (top-k). Twilight [11] relaxes this constraint by dynamically selecting top-P entries per layer, but fine-grained, per-head KV tracking remains impractical on GPUs due to irregular control flow and uncoalesced memory accesses, especially at large scale.

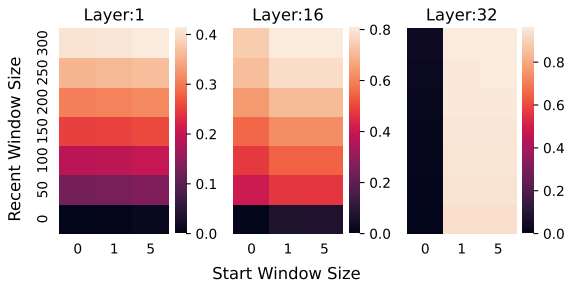


Figure 2: The heatmap of cumulative attention weights, aggregated over various starting and recent window, where values closer to 1 indicate high influence on attention output.

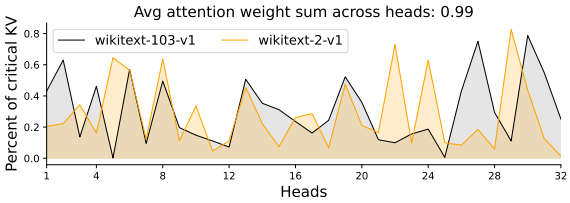


Figure 3: The percentage of KV entries required to achieve 99% of the cumulative attention score per head in layer 16 of OPT-6.7B, for two contexts at the same decoding step.

2.3 Attention Analysis

Next, we analyze attention computation in OPT-6.7B on WikiText; similar trends hold across models and datasets at scale. We examine KV attention distributions across layers and heads, KV locality, and the implications for KV-cache offloading.

Figure 2 shows cumulative attention heatmaps for entry, middle, and exit layers over start and recent windows. Attention is relatively uniform in early layers but becomes highly skewed in deeper layers—for example, the first token dominates by layer 32. This indicates that sparse attention should be layer-adaptive, as static or top-k schemes may miss important KV entries.

Because attention weights within each head sum to one, cumulative attention directly measures sparse attention coverage. Figure 3 reports the fraction of KV entries required to reach 0.99 cumulative attention across 32 heads in layer16 using two WikiText samples. The fraction varies widely, from about 10% in head12 to nearly 80% in head 30, underscoring the need for per-head sparsification. Uniform layer-wise selection can discard critical KVs in some heads while retaining redundant ones in others.

O-1: Attention becomes increasingly skewed in deeper layers and varies substantially across heads.

Figure 4 shows attention score distributions for a single request at two decoding steps: token 256 and token 512. Orange points

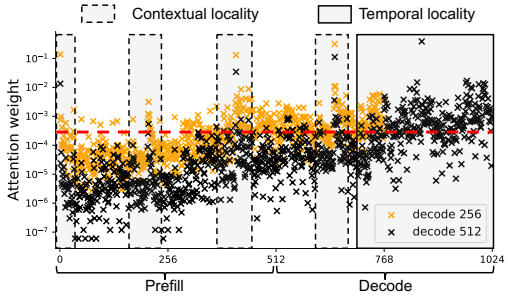


Figure 4: Attention weights at Head 6 of Layer 16 with respect to cached key-value entries during decoding of the 256th and 512th tokens, given a prefill length of 512 tokens.

denote KV entries available at token 256, while black points extend the cache to token 512. Two observations follow.

First, attention exhibits strong spatial locality: high scores concentrate on recent KV entries and on early prefill KVs, decaying with distance from the current token. Second, scores at fixed positions decrease as decoding progresses, as tokens move farther from the decoding frontier. Meanwhile, a small set of KV entries demonstrates strong contextual locality, remaining influential throughout decoding, as highlighted in the dotted region.

The red horizontal line indicates the KV entries needed to reach 0.9 cumulative attention at the latest step, consisting mainly of spatially local KVs with a few contextually important earlier entries.

O-2: Attention exhibits both spatial and contextual locality within a given context.

Figure 5 compares CPU and GPU attention when KV entries are fetched from host memory, emulating KV caches that exceed GPU capacity. Despite higher compute throughput and bandwidth, GPUs do not consistently outperform CPUs in this regime.

For single-token decoding (query size1), GPU attention is slower due to PCIe transfer overhead. At typical append sizes (query size32), GPU and CPU performance converge as computation amortizes data movement. With larger batches, GPU attention scales better through parallelism, but PCIe transfer costs grow proportionally.

Overall, PCIe KV transfers dominate GPU attention latency, making data movement the primary bottleneck.

O-3: With PCIe KV transfers, CPU-based attention can match GPU-based attention performance.

Summary. Our analysis reveals a clear trade-off between CPU and GPU attention when the KV cache exceeds GPU memory capacity, and shows that fine-grained, per-head sparse attention can preserve high accuracy. These insights motivate the design of a hybrid CPU–GPU attention mechanism.

2.4 More Related Work

Sparse attention: Sparse attention reduces long-context inference cost by retaining only salient KV entries. Prior work focuses on heuristic, GPU-resident sparsification [2, 7, 29] and recent hardware-aware designs scale these techniques to long sequences [12, 24, 28]. In contrast, Beyond uniquely performs sparse attention on the CPU with head-granular selection, allowing DRAM bandwidth and

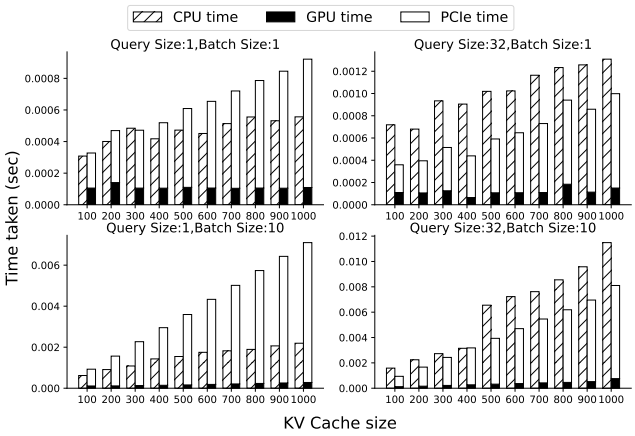


Figure 5: A breakdown of CPU-GPU attention time.

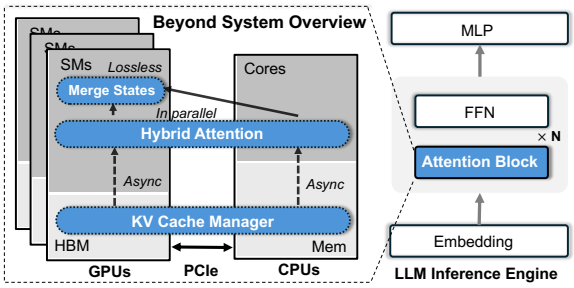


Figure 6: System overview of Beyond.

fast context switching to directly participate in memory-bound decoding.

Hybrid computation: Prior hybrid systems primarily use CPUs as auxiliary resources, enabling out-of-core execution [16] or staging KV caches that are later reloaded onto GPUs for attention [13, 25, 33]. In contrast, Beyond executes attention on both CPUs and GPUs, treating the CPU as a first-class compute device rather than a control or staging layer.

3 Beyond

In this section, we introduce Beyond, a novel CPU-GPU hybrid attention mechanism designed to deliver high accuracy and efficiency in LLM inference, especially for long-sequence contexts and large batch sizes. We begin with a high-level system overview of Beyond and outline its core challenges and our solutions (§3.1). We then detail the key components of Beyond, including locality-aware KV cache management (§3.2) along with a hybrid attention mechanism (§3.3).

3.1 System Overview

Figure 6 illustrates the architecture of Beyond, which operates within the attention blocks of LLM inference. Beyond orchestrates memory and computation across heterogeneous hardware to address the growing KV-cache footprint in LLM serving. It offloads portions of the KV cache to CPU memory to overcome GPU capacity limits for long sequences. Unlike prior offloading-only approaches (e.g., InfiniGen [9]), which use CPUs solely for KV storage, Beyond

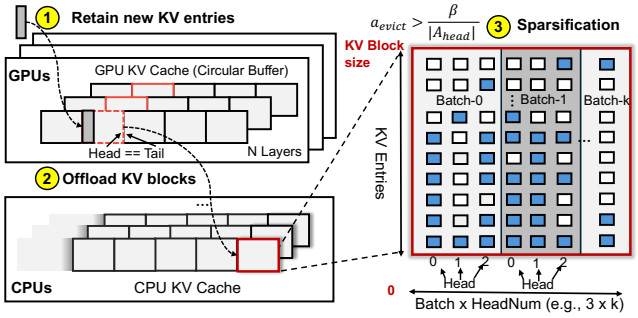


Figure 7: BEYOND: The KV cache manager.

further leverages CPU compute to cooperatively execute attention alongside the GPU, transforming GPU-only inference into collaborative CPU–GPU execution without sacrificing performance or accuracy.

Efficient joint attention execution faces two key challenges. (1) *Compute disparity*: CPUs offer much lower throughput than GPUs for matrix-heavy workloads, so naively offloading dense attention computation would bottleneck performance. Computation must be carefully partitioned to avoid slowing GPU progress. (2) *Limited interconnect bandwidth*: PCIe introduces bandwidth and latency constraints; transferring large volumes of KV data can easily dominate runtime. Communication must therefore be minimized.

To address these challenges, BEYOND introduces two techniques: *locality-aware KV cache management* (§3.2) and *bandwidth-efficient hybrid attention* (§3.3).

The KV cache manager distributes KV entries across GPU and CPU memory based on the observation that recent tokens dominate attention for subsequent decoding steps (§2). It keeps recent entries on the GPU while offloading older, less influential entries to CPU memory when GPU capacity is constrained. This recency-aware placement preserves fast access to critical context while exploiting the sparsity of attention over distant tokens. BEYOND processes these offloaded entries using fine-grained per-head sparse attention on CPUs, leveraging their control efficiency and multicore parallelism to significantly reduce computation without accuracy loss.

Building on tiled attention [5], BEYOND splits attention into GPU-resident and CPU-resident blocks and merges their results through a *lossless* aggregation. The GPU performs full attention over recent KV entries, while the CPU computes sparse attention over offloaded entries in parallel. Each produces partial outputs and normalization statistics, which are transferred to the GPU for final merging. Since only compact intermediate results are communicated, PCIe traffic is orders of magnitude smaller than moving full KV caches.

In summary, BEYOND repurposes CPUs as large-memory attention co-processors rather than passive KV storage. By combining locality-aware KV placement with bandwidth-efficient hybrid attention, BEYOND relieves GPU memory pressure, minimizes data movement, and enables scalable high-throughput inference for long contexts and large batches on single or multi-GPU systems.

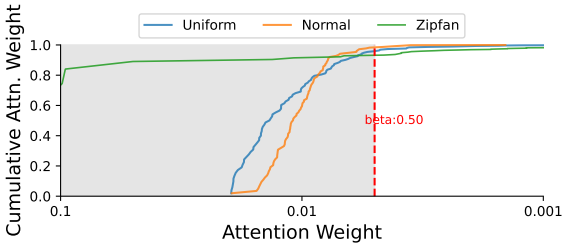


Figure 8: Cumulative attention weights with $\beta = 0.5$.

3.2 Locality-Aware KV Cache Management

BEYOND's KV cache manager focuses on two main tasks: (1) managing GPU and CPU memory for KV cache retention and offloading (❶ and ❷ in Figure 7) and (2) tracking the importance of KV cache for CPU-side sparsification (❸).

3.2.1 GPU KV cache management. BEYOND leverages *temporal locality* in GPU KV cache management and applies a simple FIFO eviction policy to offload KV entries to CPU memory. As discussed in § 2, the most recent KV entries have the greatest impact on decoding accuracy, and different layers exhibit varying degrees of skew in their attention distributions. Therefore, BEYOND manages the GPU-resident KV caches on a per-layer basis to better identify critical entries. As shown in Figure 7 ❶, BEYOND pre-allocates a contiguous GPU memory buffer for KV cache storage and partitions it into sub-regions, each dedicated to an attention layer. Each sub-region is organized as a circular buffer of fixed-size KV blocks. New KV entries are appended to the tail, while the oldest block at the head is evicted and offloaded to CPU memory if the buffer reaches its capacity. To reduce transfer overhead, BEYOND performs KV offloads at the block granularity rather than per token, effectively amortizing the GPU–CPU data transfer cost. The block size is configurable, representing a trade-off between GPU memory internal fragmentation (i.e., memory slack) and efficient PCIe bandwidth utilization.

While GPU-side KV cache management is straightforward, following a recency-based full-attention design with a simple FIFO eviction policy, it supplies the CPU-side sparse attention with valuable information about the attention weight of each KV entry. During the decoding process, BEYOND calculates a moving average of attention weight (MAW) for each KV entry and associates this score of importance with the KV entry when it is offloaded to CPU memory, where more sophisticated sparse attention is performed.

3.2.2 CPU KV cache management. Compared to GPUs, CPUs are well-suited for tasks with complex control logic, frequent branching, and irregular memory access patterns, making them ideal for implementing more sophisticated sparse attention algorithms. The existing GPU-side sparse attention algorithms either adopt a fixed attention budget [31] or compute it dynamically at runtime [11]. However, the limited control logic and inefficiency of divergent memory accesses on GPUs prevent these algorithms from performing sparse attention at the per-attention-head level. As discussed in § 2, the attention distribution across heads within the same layer is highly non-uniform. Consequently, applying coarse-grained, uniform sparse attention across all heads either excludes important KV entries or admits an excessive number of insignificant ones.

Head-granular sparsification: To enable fine-grained sparse attention on the CPU, Beyond maintains two KV cache regions in CPU memory: a large region that stores all offloaded KV blocks and a smaller region containing only those blocks with high attention weights. Beyond employs multiple CPU threads to perform per-head, sparse attention only on these significant KV entries. As discussed in § 2, KV cache entries with high attention weights include both those corresponding to recent tokens (retained on the GPU) and those far from the current decoding window that nonetheless carry important contextual information. Beyond's CPU attention focuses on the latter with contextual locality. Upon receiving offloaded KV blocks from the GPU, Beyond identifies the highest-scoring KV entries and replicates them in the small, sparse-attention region. The objective of this sparsification process is to retain entries that collectively account for more than 90% of the attention weight within each head. As shown in Figure 7 (❸), Beyond organizes each layer's sparse attention region into sub-regions for different batches to exploit concurrency across batches during sparse attention.

Beyond uses an adaptive thresholding algorithm to select important KV cache entries for each attention head. Let a_{evict} denote the moving average of the attention weight of a KV cache entry, which is calculated on the GPU during the decoding process, and $|A_{head}|$ be the total number of KV cache entries from the same attention head. Beyond introduces a tunable parameter β to determine the significance of a KV cache entry – a KV entry is retained in sparse attention if and only if $a_{evict} > \frac{\beta}{|A_{head}|}$. As all KV attention weights within a head sum up to 1, the average KV attention weight is $a_{avg} = \frac{1}{|A_{head}|}$, if attention weights were evenly distributed. However, attention weights within a head are highly skewed. Our empirical analysis shows that only 1–30% of the total KV entries contribute the majority of the attention weight across diverse datasets, while the remaining entries (at least 70%) can be safely disregarded during attention computation.

Figure 8 shows the cumulative attention weights of the selected KV entries with $\beta = 0.5$ for three representative, synthetic attention weight distributions. The figure shows that the selected KV entries account for more than 90% of the total attention weights. Based on the attention distribution, β can be adjusted to vary the selection criteria. Furthermore, our experiments show that the same β remains effective across a wide range of data sets. After selection, the attention weights of the retained KV entries are re-normalized such that their attention weights sum to 1 within each head, thereby preserving a valid attention distribution for downstream computations.

Re-evaluation: Once KV entries are sparsified on the CPU, they typically do not require *re-evaluation* until the current decoding session concludes, as the contextual locality has already been established. However, when a new prompt is issued – commonly referred to as an append operation – the newly appended tokens may alter the contextual relevance of previously offloaded entries. Tokens that were previously considered insignificant may become salient as the context changes, while previously important tokens may diminish in relevance. For example, Figure 3 demonstrates that the set of salient KV entries can vary significantly across different contextual conditions. To accommodate such changes, an asynchronous evaluation thread is launched on the CPU following each append operation. This thread re-evaluates the contextual relevance of all KV entries within an attention stored on the CPU. KV entries that no longer meet the significance threshold $\frac{\beta}{|A_{head}|}$ are removed from the sparse attention region, while those exceeding the threshold are added to the sparse attention. Although maintaining a sparse-attention region introduces additional space overhead in CPU memory, our empirical studies across diverse datasets show that its memory requirement averages less than 10% of the total KV cache size in CPU memory.

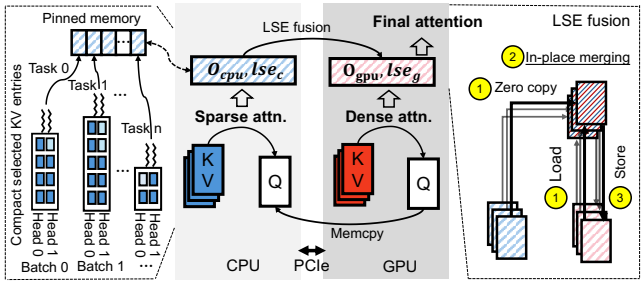


Figure 9: Beyond: hybrid attention computation.

3.3 Hybrid Attention

A key feature of Beyond 's design is its hybrid attention mechanism (Figure 9), which partitions attention computation between the GPU and CPU. The GPU performs full attention over a sliding window of recent KV entries, while the CPU applies sparse attention to older entries that have been offloaded from the GPU. The final attention result is then produced by merging the partial results from both devices. This co-design seeks to achieve two goals: (1) *Reducing compute disparity.* Although there is a significant disparity in attention computation capability between the CPU and GPU, Beyond seeks to bridge this gap and minimize GPU idle time. It does so by focusing CPU attention on a small fraction (~10% of the total) of the offloaded KV cache and leveraging multicore parallelism to accelerate sparse attention. (2) *Preserving decoding accuracy.* Beyond 's hybrid attention is designed to deliver decoding accuracy comparable to performing full attention over the entire KV cache on the GPU, as if all entries resided in GPU memory. To achieve this, Beyond selects the most important KV entries for sparse attention, aligns and normalizes the partial attention outputs from the CPU and GPU to ensure compatibility, and synchronizes the two computations in lockstep for correct integration.

GPU full attention. Beyond's KV cache management (§ 3.2) enables sliding-window full attention on the GPU: it maintains only the most recent KV blocks in GPU memory, evicting older blocks to CPU memory as new tokens are generated. The latest W tokens in the sliding window are processed using the standard dense attention.

CPU sparse attention. Recall that Beyond employs a per-head sparsification mechanism to select the most salient KV entries for sparse attention (§ 3.2.2). To enable efficient computation, the

selected entries are reorganized by attention head and stored contiguously in CPU memory.[2] This layout produces compact, head-specific KV subsets that facilitate efficient parallel processing. Beyond then exploits multicore CPUs by distributing these subsets across multiple threads, with each thread handling the attention computation for a single head. Because the subsets are disjoint, the final multi-head attention output is obtained by simply concatenating the per-thread results, which are stored in pinned memory as shown in Figure 9. The pinned memory allows the GPU to use zero-copy memory access to merge attention outputs on the CPU with those on the GPU.

Increasing the batch size presents a practical challenge: the number of required threads scales with the product of batch size and the number of attention heads, potentially causing CPU oversubscription. To address this, Beyond merges computations from multiple adjacent heads into a single sparse-attention task, thereby reducing the overall thread count (Figure 9)[3]. Merging adjacent heads requires a relaxed per-head sparsification strategy, as heads may select different numbers of KV entries. To align the selected entries across merged heads, Beyond allows padding with additional KV entries even if they do not meet the sparsification threshold. The CPU's flexible control logic makes it well-suited for managing such heterogeneous tasks, in contrast to GPUs, which favor rigid, uniform execution patterns and perform best on workloads of equal size.

Merging states. After computing attention over the GPU sliding window and the selected CPU KV entries, Beyond merges the two partial results to obtain the final attention output. The objective is to combine these computations as if they were derived from a single softmax over the union of tokens, while preserving numerical stability and computational efficiency. Inspired by FlashAttention [5], Beyond employs an *in-place aggregation and log-sum-exp (LSE) fusion* strategy. After the GPU computes its partial attention output O_{gpu}, and the CPU computes its partial output O_{cpu}, both outputs are locally *normalized* within their respective domains and must be appropriately re-scaled to obtain a unified result. Each partial result is associated with a log-sum-exp term of the form:

$$\text{lse}_I = \log\left(\sum_{j \in I} e^{s_j}\right).$$

where $s_j = q \cdot k_j$ denotes the attention score corresponding to the j^{th} key. The index set I identifies the subset of tokens residing in a particular compute domain (CPU or GPU). A coordinated normalization step is then carried out to compute a unified scaling factor that spans both domains:

$$z = e^{\text{lse}_c} + e^{\text{lse}_g}.$$

Using this shared factor z – which is derived by merging two scalars between the CPU and GPU (the maximum score and the sum of exponentials) – the partial outputs are scaled and combined in-place:

$$\mathbf{O} = \frac{1}{z}\left(e^{\text{lse}_c} \cdot O_{\text{cpu}} + e^{\text{lse}_g} \cdot O_{\text{gpu}}\right).$$

This yields an output equivalent to that produced by a single softmax operation over the union of GPU and CPU tokens. The merge is performed in-place by accumulating O_{cpu} directly into the GPU's output buffer and applying the shared normalization factor. The use of the LSE trick guarantees numerical stability when fusing attention scores from heterogeneous sources. The communication overhead is minimal, as only the vector O_{cpu} and scalar lse_{cpu} are transmitted from the CPU to the GPU, avoiding the need to transfer full KV tensors. In practice, Beyond facilitates this exchange via *zero-copy* memory access, enabling the GPU to directly access CPU-resident data. This zero-copy strategy bypasses the GPU memory hierarchy and is especially effective for small or irregular data structures, making it well-suited for the dynamic behavior of Beyond's hybrid attention mechanism. The merging process occurs at the end of the attention layer, ensuring that the combined output can be directly consumed by the subsequent feed-forward network.

4 Evaluation

We implemented Beyond with ~1.5K lines of Python, built on PyTorch v2.4+cu124 with a customized flashinfer v0.1.6. KV cache management employs pre-allocated GPU buffers with asynchronous CUDA streams for updates and offloading, while CPU-side caches expand dynamically with a context cache that retains selected entries per head. Dense attention is executed on the GPU, and sparse attention on the CPU, optimized via `torch.jit` and multithreading. Intermediate CPU states are stored in reused pinned memory for zero-copy fusion [27], minimizing overhead and enabling scalable CPU–GPU integration. In this section, we present an extensive evaluation of Beyond across multiple models, datasets, and hardware platforms, benchmarking against representative baselines to demonstrate its advantages in efficiency, scalability, and accuracy.

Platforms. Table 1 summarizes the hardware platforms. Platform A was equipped with 16x32 GB DDR5 modules, whereas Platforms B and C were configured with 8x or 16x 64 GB DDR5 modules. Notably, Platform B used only half of the available memory channels compared to Platforms A and C, reducing its effective memory bandwidth by half.

Platforms	CPU	DRAM	GPU	PCIe
A	2× Intel 6430	16×32GB DDR5	8× A6000	5.0
B	2× Intel 6430	8×64GB DDR5	A100, 4090	5.0
C	2× Intel 8470	16×64GB DDR5	2× H100	5.0

Table 1: Hardware configurations of evaluated platforms.

Models. We evaluated three representative open-source language models with distinct architectural designs: Llama [18], GPT-NeoX [3], and OPT [30]. Each model was further scaled to multiple sizes to assess the scalability of all methods.

Baselines. We integrated Beyond into two inference frameworks – FlexGen [16] and Hugging Face (HF)[21] – and evaluated its performance against representative baselines.

FlexGen targets memory-constrained, single-GPU environments by leveraging a tiered memory hierarchy that swaps model components (e.g., weights and KV caches) between device and host memory, trading latency for extended capacity to maximize throughput. We compared Beyond to FlexGen-based variants that incorporate sparse attention and KV offloading, including Infinigen[9],

[2]This reorganization is performed during sparsification, immediately after KV offloading to CPU, and thus does not add to the critical path of attention computation.

[3]In practice, the number of heads merged per task is chosen to approximate $\frac{batch\_size \times head\_num}{core\_cnt}$.

which performs rehearsal-based KV prefetching across layers, and H2O [31], which evicts KV entries according to accumulated attention scores. The HF framework provides broad support for diverse model configurations and features automatic device mapping when GPU memory is insufficient. We integrated Beyond by managing KV cache placement in coordination with HF's device assignment logic, and evaluated performance under two settings: (1) a GPU-constrained configuration with KV caches offloaded, and (2) a GPU-rich configuration with all KV caches retained in high-bandwidth device memory.

Metrics. We used four metrics: (1) *end-to-end task completion time* as a function of output sequence length and batch size; (2) *peak GPU memory consumption* attributable to KV cache storage; (3) *inference throughput* under diverse model configurations; and (4) *per-token generation latency* during decoding. Accuracy was measured in terms of perplexity and per-task scores on the LongBench benchmark.

4.1 Micro-benchmark

To evaluate the performance of the attention mechanism enabled by Beyond, we first conducted experiments by focusing on a single attention layer under varying batch sizes and offloaded KV sizes across three OPT configurations and three hardware platforms (A, B, and C). As shown in Figure 10, the comparison considers two approaches: (i) Beyond's hybrid attention with result merging, and (ii) reloading followed by GPU computation. It shows the breakdown of attention time under these two approaches.

The results show that reloading introduces substantial latency, which increases linearly with KV size due to PCIe transfer overhead (e.g., from 512 to 4096). By contrast, Beyond reduces this cost by partitioning the KV cache between CPU and GPU and exchanging only essential data. The advantage of hybrid attention grows with larger CPU-resident caches and higher batch sizes. For instance, on OPT-66B with 4,096 entries and batch size 10, Beyond's hybrid attention runs about 11× faster than the reload–compute approach. For small offloaded caches (e.g., 512 entries), as in OPT-13B with batch size 1, reload–compute remains more efficient, since less data must be transferred over PCIe. Note that, for fairness, CPU execution is restricted to head-granular attention without sparsification; enabling sparsification would further reduce KV size and deliver additional performance gains, as demonstrated later.

4.2 End-to-end Performance

We further conducted the end-to-end performance comparisons with two inference frameworks, FlexGen and HF.

In the **FlexGen** framework, experiments were conducted on Platforms A and C, both with fully populated memory channels, using three OPT model sizes. We measured the *total generation time* for 128 output tokens with a prefill length of 1,920 tokens. FlexGen supports on-demand model weight swapping, allowing the execution of larger models. The ratio of model weights persisted on GPU during swapping was set to maximize throughput. We evaluated three OPT models while scaling batch size up to the maximum supported by each method: FlexGen, H2O, Infinigen, and Beyond.

For KV management, 100% of entries were initially placed in CPU memory and loaded into GPU memory on demand, following the setup in Infinigen [9]. FlexGen executed standard full attention, whereas the other methods applied sparse attention. Infinigen and H2O were configured to perform top-k attention over 20% of KVs, as specified in their respective works. For Beyond, 5% of KVs remained resident on GPU, with the remainder stored and computed on the CPU.

Figure 11 shows that Beyond consistently outperforms FlexGen and H2O across all batch sizes, achieving up to 5.6x and 2.1x speedups (e.g., on OPT-66B with H100). It is because those methods incur overhead from KV loading and LRU-based eviction. Infinigen achieves performance comparable to Beyond. However, it incurs significantly higher memory usage due to rehearsal-based attention, which requires auxiliary memory buffers to predict scores. This overhead leads to out-of-memory (OOM) failures across all models, particularly pronounced with large model and batch sizes, e.g., on OPT-66B with A6000, Infinigen encounters OOM beyond batch size 15, while Beyond continues to serve larger sizes.

In the **HF** framework, model parallelism and hardware mapping are automatically determined based on the number of available GPUs. Standard attention executes full attention within the provisioned device HBM. In contrast, Beyond restricts GPU attention to a predefined buffer capacity and delegates overflow computation to the CPU. We compared Beyond against full attention across varying GPU counts and different predefined GPU buffer ratios for Beyond.

Figure 12 reports the aggregate *token generation rate* (tokens per second) for batch size 15, generating 4,096 tokens on three models on Platform A. Beyond successfully serves all generation tasks with fewer GPUs by offloading KV caches and portions of the attention workload to CPU. Although CPU TFLOPs lags behind GPU, performance degradation of hybrid attention remains minimal at larger batch sizes and longer contexts compared to full GPU attention (ratio 1) due to Beyond leveraging the combined CPU–GPU memory bandwidth to sustain bandwidth-bound decoding tasks.

4.3 Accuracy

We also compared the model generation quality of Beyond's hybrid attention against full attention using two representative benchmarks, Perplexity and LongBench. Note that, Beyond is designed to improve system-wide efficiency *without* compromising model accuracy, albeit it does not aim to enhance intrinsic generation quality. Accordingly, accuracy evaluations serve only to confirm that end-to-end generation quality remains consistent with full attention.

Perplexity, a standard NLP metric, measures how closely a model's predicted probability distribution aligns with ground-truth outputs given prior context, with lower values indicating stronger predictive performance. Evaluating perplexity over long sequences quantifies the model's ability to capture long-range dependencies. We compared Beyond against a full-attention baseline over the entire KV cache.

Beyond applies sparse attention on CPU by exploiting contextual KV locality. We varied the filtering threshold β and the fraction of KV attention executed on the GPU. As shown in Table 2, Beyond

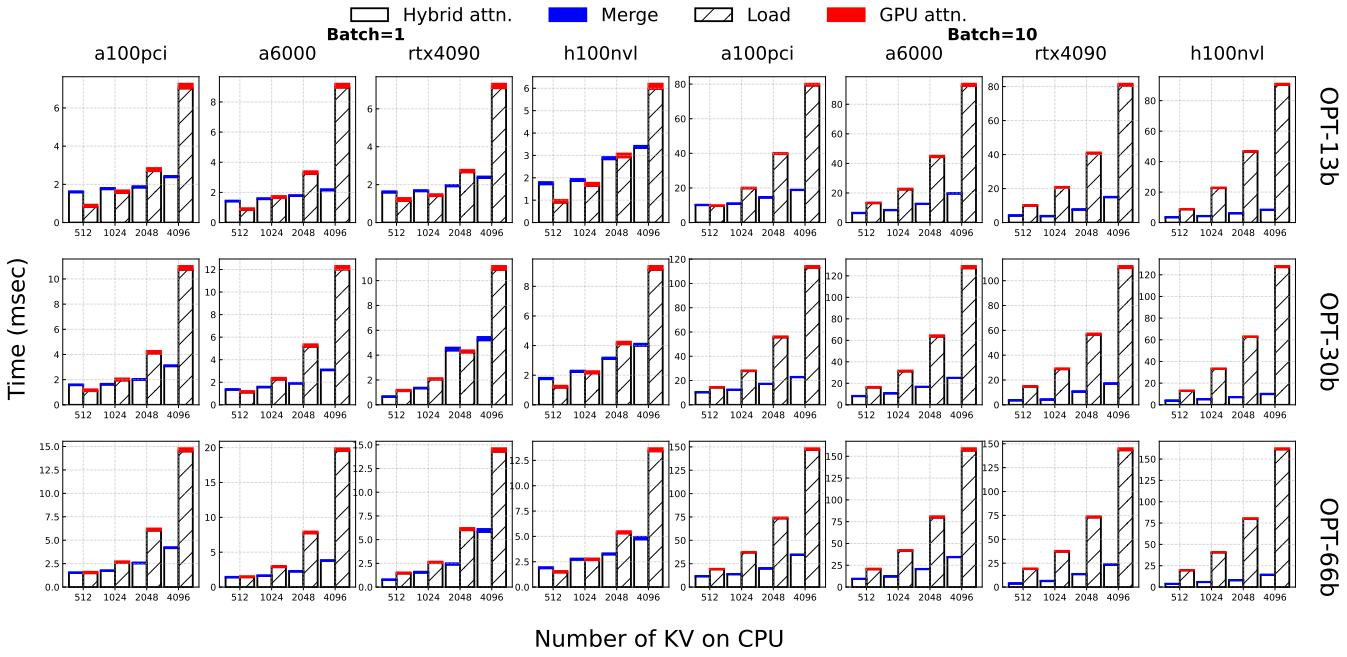


Figure 10: A breakdown of attention time.

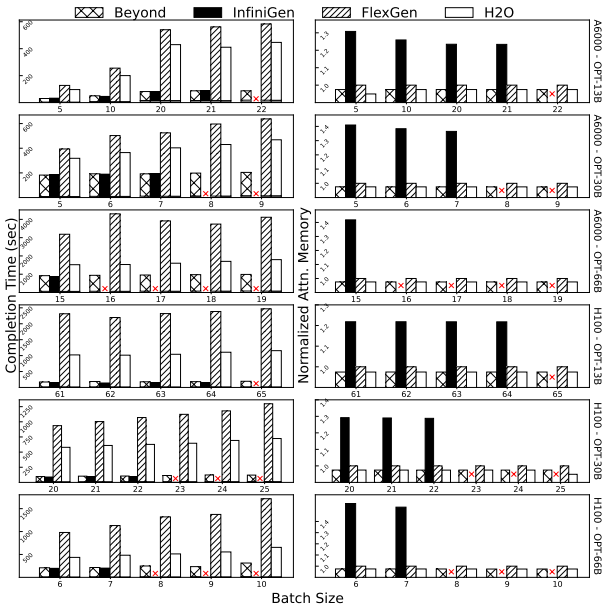


Figure 11: Generation performance of OPT models using FlexGen under varying batch sizes and model configurations.

achieves accuracy comparable to, and in some cases exceeding, that of full attention. Notably, the GPU KV ratio showed no consistent impact on accuracy, highlighting the robustness of sparse attention. Strongest performance typically occurred at larger β values with more selective KV filtering, indicating more aggressive sparsification for improved efficiency in resource-constrained settings.

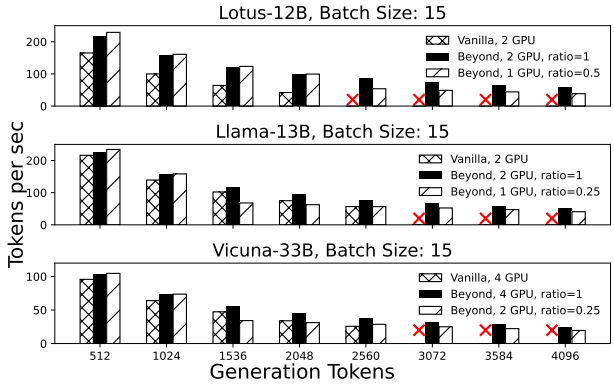


Figure 12: The generation of 4096 tokens.

LongBench: We further assessed end-to-end generation quality using the LongBench benchmark [1], which covers five representative datasets and three Vicuna model sizes, as illustrated in Figure 13. As a control baseline, we used StreamingLLM (with static sparse attention), which retains the same subset of recent KV entries in HBM as Beyond but adds 64 attention sink tokens at the sequence prefix while evicting all others. In contrast, Beyond dynamically identifies salient KV entries on the CPU side. Beyond shows only minor degradation on a few tasks while consistently preserving generation quality across most cases, matching full attention and even outperforming it on the 13B and 30B models. Further, Beyond significantly outperforms StreamingLLM in almost all cases on all three models. These results underscore the importance of dynamic runtime mechanisms for selecting salient KV entries in sparse attention.

Table 2: The comparison of perplexity between full attention and BEYOND's hybrid attention. Text lengths are 2048, 4096, and 8192 for OPT, GPT-Neox, and LLaMA-based models, respectively. Reference perplexity and BEYOND's better-than-reference performance are highlighted in bold.

Model **Baseline PPL**	GPU Ratio	$\beta = 0.25$	$\beta = 0.5$	$\beta = 0.75$	$\beta = 1.0$
OPT-6.7B **19.12**	0.25	19.42	19.17	**19.02**	19.12
	0.5	19.17	19.12	19.09	19.09
	0.75	19.12	19.17	19.17	19.20
OPT-13B **17.69**	0.25	17.97	18.05	18.08	17.94
	0.5	17.77	17.80	17.80	17.83
	0.75	**17.69**	17.69	17.69	17.72
OPT-30B **16.78**	0.25	19.97	19.92	19.78	19.70
	0.5	16.91	16.78	16.78	**16.75**
	0.75	18.33	27.83	19.42	18.47
LLaMA-2-7B **124.4**	0.25	112.1	115.9	106.3	105.4
	0.5	105.3	100.8	**93.28**	99.81
	0.75	124.6	124.9	125.6	125.5
LLaMA-2-13B **227.2**	0.25	288.7	308.9	310.2	307.2
	0.5	293.8	308.8	330.9	341.1
	0.75	226.2	225.7	225.6	**225.1**
Vicuna-33B-v1.3 **889.1**	0.25	993.5	1013	941.5	983.5
	0.5	896.2	985.4	894.9	893.8
	0.75	1013	1033	1130	1013
GPT-NeoX-12B **66.88**	0.25	51.38	51.19	51.19	**51.09**
	0.5	54.59	52.81	51.91	51.59
	0.75	65.88	65.38	64.81	64.56

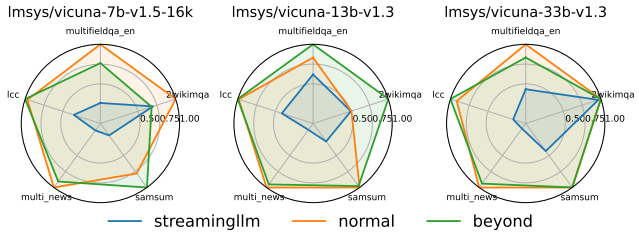


Figure 13: LongBench.

4.4 Ablation Test

BEYOND's hybrid attention enables configurable CPU-side sparse attention. We conducted the following ablation tests on Platform A (64 physical cores and 2 threads per core).

(i) **Head-level attention**: Consecutive attention heads are grouped into a single attention task, each launched with a configurable number of threads. Figure 14 reports normalized throughput as a function of the number of tasks and threads per task under varying batch sizes. The total number of threads is calculated by *Threads Num* × *Total Tasks Num*. Throughput is normalized to GPU-only attention with sufficient HBM capacity. We observe that the optimal configuration uses two threads per task with 40–80 active threads

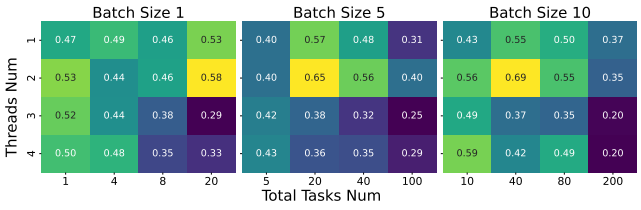


Figure 14: Thread number vs. head number per task.

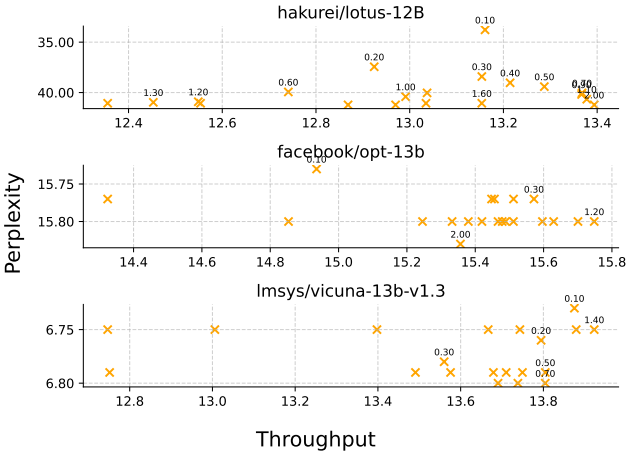


Figure 15: Efficiency and accuracy tradeoff.

(about 50% of total available CPU cores); larger thread counts oversubscribe the CPU and degrade performance.

(ii) **Threshold-based filtering**: A threshold parameter β determines the number of KV entries retained. Larger β values retain fewer entries, improving sparse attention efficiency at the potential expense of higher perplexity (i.e., lower accuracy). Figure 15 presents perplexity and throughput as β varies from 0.1 to 2.0 in increments of 0.1 across three model types. In each subfigure, the best balance between accuracy (low perplexity) and performance (high throughput) appears at the rightmost point for OPT-13B, and slightly left of the rightmost point for Lotus-12B and Vicuna-13B, where accuracy improves with little loss of throughput.

Since the ground-truth attention sparsity differs across heads, jointly tuning β and the number of heads per task more accurately captures these patterns and yields tangible wall-clock speedups under suitable threading configurations, aided by the CPU's efficient context switching.

4.5 Long Context Inference

Lastly, we stress-tested BEYOND with an extended decoding session of 16,384 tokens in a single request (batch size 1), in Figure 16. The GPU KV cache was capped at 4,096 entries with $\beta = 1$, and hybrid attention was activated once the sequence length exceeded this limit. Figure 16 reports both tokens-per-second throughput and time-between-tokens (TBT). BEYOND successfully scaled to long sequences without OOM errors, sustaining 3–4 tokens per second (180–250 tokens per minute) near the end of generation – comparable to human reading speed. Meanwhile, we observed certain TBT variance and outliers, primarily caused by inefficient multi-thread scheduling and load imbalance in CPU attention. These findings

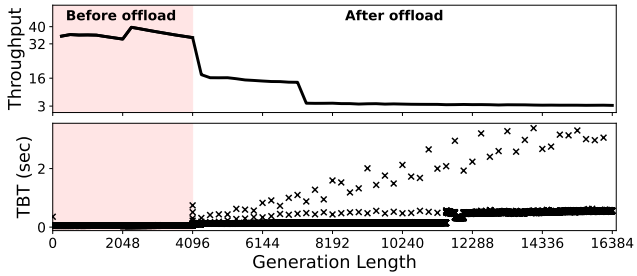


Figure 16: Long context inference with GPT-Neox-12B.

highlight both the feasibility and the need for further optimization for more efficient CPU-size sparse attention approaches.

4.6 Hybrid Attention with NVlink C2C

Faster CPU–GPU interconnects are becoming practical, powered by technologies such as NVLink-C2C. While the performance gains in our current design arise primarily from alleviating the PCIe bottleneck, next-generation interconnects open new opportunities for Beyond. With high-bandwidth links like NVLink-C2C, Beyond can continue to offload GPU-unfriendly operations (e.g., asynchronous sparsification and head-granular sparse attention) to the CPU, while delegating a larger share of the attention computation to the GPU.To explore this regime, we extend our attention experiments to the DGX Spark platform (Figure 17), in which CPU and GPU are connected via NVLink-C2C and share physical memory with up to 273 GB/s of bandwidth. We compare vanilla attention executed solely on the CPU or solely on the GPU against our hybrid attention across a range of GPU/CPU KV-cache size configurations. Surprisingly, hybrid attention outperforms GPU-only attention in several cases: when both partitions are large enough to saturate the CPU and GPU memory subsystems simultaneously, the aggregated bandwidth more than compensates for the compute gap between the two devices.

5 Conclusions

We present Beyond, a novel hybrid CPU-GPU attention mechanism that enables scalable LLM inference by combining GPU-based dense attention with CPU-optimized sparse attention. Beyond preserves model accuracy while s reducing GPU memory pressure. Through efficient fusion and fine-grained, per-head sparsification, Beyond delivers strong performance gains and achieves superior scalability, enabling support for longer contexts and larger batch sizes on commodity GPU hardware. These benefits primarily stem from reducing the PCIe bottleneck. Looking ahead, even with faster interconnects such as NVLink-C2C, Beyond remains effective by offloading GPU-unfriendly operations (e.g., asynchronous sparsification) to the CPU while shifting more attention computation to the GPU.

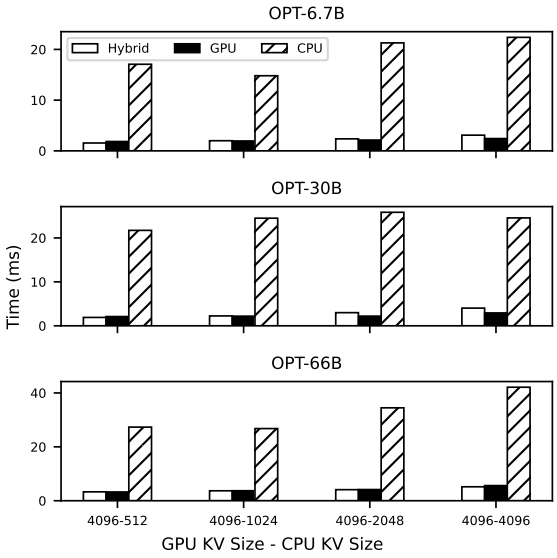


Figure 17: Attention Comparison on DGX Spark.

References

[1] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508* (2023).

[2] Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150* (2020).

[3] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, et al. 2022. Gpt-neox-20b: An open-source autoregressive language model. *arXiv preprint arXiv:2204.06745* (2022).

[4] Sumit Kumar Dam, Choong Seon Hong, Yu Qiao, and Chaoning Zhang. 2024. A complete survey on llm-based ai chatbots. *arXiv preprint arXiv:2406.16937* (2024).

[5] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems* 35 (2022), 16344–16359.

[6] Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. 2024. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Advances in Neural Information Processing Systems* 36 (2024).

[7] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451* (2020).

[8] 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*. 611–626.

[9] 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 (OSDI 24)*. 155–172.

[10] Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*. PMLR, 19274–19286.

[11] Chaofan Lin, Jiaming Tang, Shuo Yang, Hanshuo Wang, Tian Tang, Boyu Tian, Ion Stoica, Song Han, and Mingyu Gao. 2025. Twilight: Adaptive Attention Sparsity with Hierarchical Top-p Pruning. *arXiv preprint arXiv:2502.02770* (2025).

[12] Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, et al. 2025. MoBA: Mixture of Block Attention for Long-Context LLMs. *arXiv preprint arXiv:2502.13189* (2025).

[13] Xiurui Pan, Endian Li, Qiao Li, Shengwen Liang, Yizhou Shan, Ke Zhou, Yingwei Luo, Xiaolin Wang, and Jie Zhang. 2024. Instinfer: In-storage attention offloading for cost-effective long-context llm inference. *arXiv preprint arXiv:2409.04992* (2024).

[14] Jonathan Pilault, Raymond Li, Sandeep Subramanian, and Christopher Pal. 2020. On extractive and abstractive neural document summarization with transformer language models. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*. 9308–9319.

[15] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).

[16] 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 *International Conference on Machine Learning*. PMLR, 31094–31116.

[17] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. Quest: Query-Aware Sparsity for Efficient Long-Context LLM Inference. *arXiv preprint arXiv:2406.10774* (2024).

[18] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[19] Shihao Wang, Xiangyu Zou, Wen Xia, and Hao Hu. 2026. An Ultra-fast and Lossless Floating-Point Compressor for LLM Inference Systems. In *Proceedings of the 63rd ACM/IEEE Design Automation Conference (DAC '26)*. ACM, 1–7.

[20] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[21] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771* (2019).

[22] Mingze Xia, Sheng Di, Franck Cappello, Pu Jiao, Kai Zhao, Jinyang Liu, Xuan Wu, Xin Liang, and Hanqi Guo. 2024. Preserving topological feature with sign-of-determinant predicates in lossy compression: A case study of vector field critical points. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 4979–4992.

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

[24] Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. 2025. XAttention: Block Sparse Attention with Antidiagonal Scoring. *arXiv preprint arXiv:2503.16428* (2025).

[25] Wenhua Ye, Xu Zhou, Joey Zhou, Cen Chen, and Kenli Li. 2023. Accelerating attention mechanism on fpgas based on efficient reconfigurable systolic array. *ACM Transactions on Embedded Computing Systems* 22, 6 (2023), 1–22.

[26] Zihao Ye, Lequn Chen, Ruihang Lai, Yilong Zhao, Size Zheng, Junru Shao, Bohan Hou, Hongyi Jin, Yifei Zuo, Liangsheng Yin, et al. 2024. Accelerating self-attentions for llm serving with flashinfer.

[27] Zihao Ye, Ruihang Lai, Bo-Ru Lu, Chien-Yu Lin, Size Zheng, Lequn Chen, Tianqi Chen, and Luis Ceze. 2024. Cascade inference: Memory bandwidth efficient shared prefix batch decoding.

[28] Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, YX Wei, Lean Wang, Zhiping Xiao, et al. 2025. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *arXiv preprint arXiv:2502.11089* (2025).

[29] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *Advances in neural information processing systems* 33 (2020), 17283–17297.

[30] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068* (2022).

[31] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. 2024. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems* 36 (2024).

[32] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Jeff Huang, Chuyue Sun, Cody_Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. 2023. Efficiently Programming Large Language Models using SGLang. (2023).

[33] Shuzhang Zhong, Yanfan Sun, Ling Liang, Runsheng Wang, Ru Huang, and Meng Li. 2025. HybriMoE: Hybrid CPU-GPU Scheduling and Cache Management for Efficient MoE Inference. *arXiv preprint arXiv:2504.05897* (2025).