arXiv:2607.22389v1 [cs.AR] 24 Jul 2026

HiKV: Hierarchical Importance-Aware KV Cache with Hardware Acceleration for LLM Decoding

Chao Fang, *Member, IEEE,* Jun Yin, Man Shi, and Marian Verhelst, *Fellow, IEEE*

***Abstract*—With the rapid adoption of long-context large language models (LLMs), the continuously growing KV cache during decoding has become the critical memory bottleneck. To tackle this challenge, we propose HiKV, a novel algorithm-hardware co-design that exploits KV cache redundancy through hierarchical importance awareness. Algorithmically, HiKV compresses the KV cache at two granularities: Stage I evicts unimportant tokens within a fixed budget, and Stage II further loads only the significant elements of each retained token, reaching compression ratios unattainable at a single granularity. Architecturally, we develop a dedicated accelerator centered on a reconfigurable importance sorter that switches between the distinct sorting datapaths each stage requires, unifying the two-stage acceleration in one circuit with minimal overhead. Evaluated on representative LLMs, HiKV achieves up to 7.95× speedup and 90% energy reduction in the attention computation over the vanilla KV cache baseline within negligible 1% accuracy loss. Under iso-accuracy constraints, HiKV outperforms state-of-the-art importance-based methods by achieving an additional 1.82∼4.87× reduction in external memory accesses. These benefits are enabled by specialized hardware components that add only 8% to the system area.**

***Index Terms*—algorithm-hardware co-design, LLM decoding, KV cache compression, hardware acceleration, efficient AI**

I. Introduction

LARGE language models (LLMs) [1]–[4] have achieved remarkable success across diverse natural language processing tasks [5]–[7], from question answering and summarization to code generation and mathematical reasoning. The deployment of LLMs in real-world applications, however, faces significant efficiency challenges, particularly during the decoding phase of autoregressive text generation. In this phase, LLMs generate output tokens sequentially, with each token depending on all previously generated content. This sequential dependency necessitates a key-value (KV) cache mechanism that stores attention states from prior tokens to avoid costly recomputation. While eliminating redundant calculations, the KV cache introduces a critical memory bottleneck that becomes increasingly severe during the decoding phase of LLM inference.

This bottleneck, as shown in Fig. 1, stems from two compounding factors that simultaneously drive memory footprint growth. First, the cache size grows linearly with sequence length [8]–[10]. As LLMs generate longer outputs or process extended contexts for applications like document analysis and multi-turn conversations, each decoding step must access progressively larger cache volumes from external memory. Second, modern serving systems employ batch processing to improve throughput by handling multiple user requests concurrently [11]–[13]. Unlike model weights that can be shared across all requests in a batch, each request must maintain its own independent KV cache across all transformer layers. This dual scaling creates a memory-bound bottleneck where computational units remain underutilized while idling for KV cache transfers, fundamentally limiting inference efficiency.

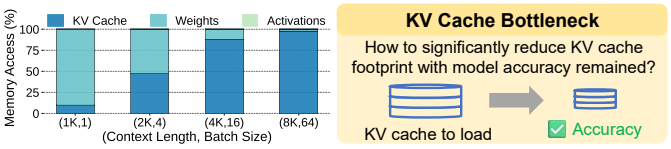

Fig. 1. The dominated memory access of KV cache becomes the bottleneck of LLM decoding, motivating HiKV to shrink its footprint with high accuracy.

To address this bottleneck, recent research has explored importance-based strategies to compress the KV cache, which can be broadly categorized into two classes. Heuristic methods [14]–[19] assign token importance statically through predefined rules, such as retaining only the most recent tokens, but often sacrifice accuracy when critical information resides beyond these fixed boundaries. Instead, importance-aware approaches [20]–[27] compute token importance dynamically through attention score accumulation, enabling data-driven eviction that achieves superior accuracy-compression tradeoffs. Despite this progress, their notion of importance remains defined exclusively at the token level: once a token is deemed important, its entire key and value vectors are required to load from external memory. As analyzed in Sec. III, however, KV cache redundancy exists at both token and element levels. Even within retained tokens, only a subset of vector elements significantly influences attention computation.

Moreover, dynamically maintaining importance across KV cache tokens at each decoding step introduces substantial sorting overhead, necessitating dedicated hardware support for efficient execution. While hardware-accelerated solutions [25]–[31] have been proposed to mitigate this sorting overhead, the memory bandwidth bottleneck imposed by loading complete key and value vectors for all selected tokens remains unresolved. This points to the need for extending importance-awareness beyond the token level to the elements within each

This work was supported in part by the European Research Council (ERC) under grant agreement No. 101088865, the European Union's Horizon 2020 program under grant agreement No. 101070374, the Flanders AI Research Program, and long-term structural Methusalem funding by the Flemish Government. *(Corresponding author: Chao Fang.)*

C. Fang, J. Yin, and M. Verhelst are with ESAT-MICAS, KU Leuven, Leuven, Belgium (e-mail: chao.fang@kuleuven.be; jun.yin@kuleuven.be; marian.verhelst@kuleuven.be).

M. Shi was with ESAT-MICAS, KU Leuven, Leuven, Belgium, and is now with UC Berkley, CA, USA (e-mail: manshi@berkeley.edu).

© 2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

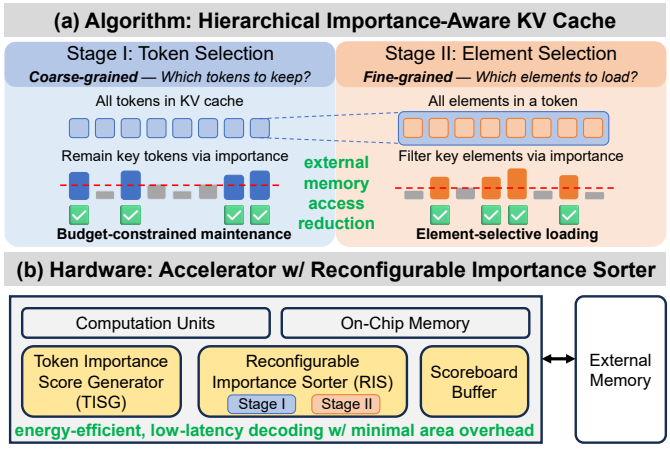

Fig. 2. Overview of HiKV, co-designing (a) an importance-aware algorithm that compresses the KV cache at both the token and element levels and (b) an accelerator featuring a reconfigurable importance sorter (RIS), achieving up to 7.95× speedup and 90% energy reduction with only 8% area overhead.

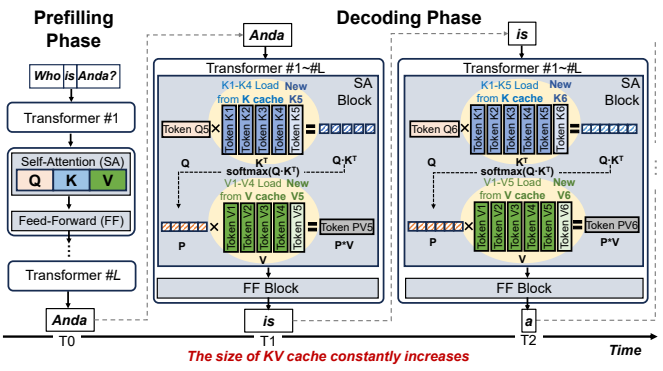

Fig. 3. During LLM autoregressive decoding, the KV cache grows linearly with each generation step, continuously amplifying external memory access volume and imposing an escalating memory bandwidth bottleneck.

token, motivating a hierarchical approach that systematically eliminates redundancy at both granularities through dedicated hardware support.

In this paper, we propose HiKV, as presented in Fig. 2, an algorithm-hardware co-design that addresses the KV cache bottleneck through hierarchical importance-aware optimization. Algorithmically, HiKV operates through two complementary stages: Stage I identifies and retains only important tokens within a fixed cache budget, while Stage II selectively loads only significant vector elements from retained tokens, together achieving compression ratios unattainable by any single-granularity approach. Architecturally, HiKV introduces a custom accelerator featuring a reconfigurable sorter that efficiently supports both stages: heap-based importance maintenance in Stage I and parallel chunk-based sorting in Stage II, delivering substantial performance and energy improvements with minimal area overhead.

In summary, the main contributions of this paper are:

- A hierarchical importance-aware approach that exploits KV cache redundancy at both token-level and element-level granularities, achieving superior compression efficiency over token-level state-of-the-art (SotA) methods within 1% accuracy loss across representative LLMs on long-context benchmarks. (Sec. III)
- A customized hardware accelerator featuring a reconfigurable sorter that efficiently supports both stages through heap-based importance maintenance for token-level optimization and chunk-based parallel sorting for element-level selection, achieving low overhead with only 8% system-level area increase. (Sec. IV)
- Comprehensive evaluation results demonstrating up to 7.95× speedup and 90% energy reduction in the attention computation during LLM decoding in contrast with the vanilla baseline, outperforming SotA KV cache optimizations under iso-accuracy constraints. (Sec. V)

The rest of the paper is organized as follows. Sec. II presents the background and motivation for KV cache optimization. Sec. III details the proposed two-stage importance-aware KV cache optimization approach. Sec. IV elaborates on the HiKV hardware architecture that enables efficient hierarchical importance maintenance. Sec. V evaluates our approach in terms of algorithm-, hardware-, and system-level performance. Finally, Sec. VI concludes the paper.

II. Background and Motivation

A. LLM Generation Fundamentals

As shown in Fig. 3, LLMs execute generation tasks through a prefill phase followed by a decoding phase [32]–[35]. During the prefill phase at timestamp T0, the model processes all input token vectors of the prompt simultaneously, computing query (Q), key (K) and value (V) matrices via self-attention mechanisms. These K and V matrices are stored in a structure, namely the KV cache [36]–[38], to avoid costly recomputation in subsequent processing. In the decoding phase, new tokens are generated step by step at timestamps T1, T2, ..., until the maximum sequence length is reached or an end-of-sequence token appears [39]. At each timestamp, the model performs two critical attention computations in general matrix-vector (GeMV) operations: QK^T determines token relationships and PV generates new context representations, where $P = \text{softmax}(QK^T)$. As text decoding proceeds, each new key-value token is continuously appended to the KV cache, causing linear expansion in memory usage. Unlike the prefill phase, where computation can be parallelized, the sequential nature of decoding, combined with the growing KV cache access demands, introduces severe computational challenges in LLM decoding.

B. KV Cache Bottleneck

In LLM decoding, memory access patterns significantly affect both performance and energy efficiency, particularly in the memory-bound decoding phase [40]–[42]. LLM services typically leverage batch processing [11]–[13] to handle multiple user requests concurrently and improve computational efficiency. Yet, this batching strategy's impact on memory access patterns needs careful examination.

Using the LLM-Viewer [42] tool, we visualize the memory transfer breakdown of four popular LLMs [1]–[4] when

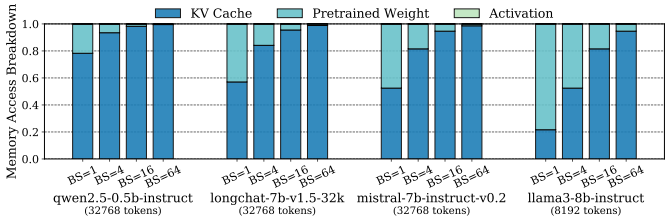

Fig. 4. Model memory transfer breakdown with different batch sizes when decoding tokens at long-context scenarios. KV cache increasingly dominates as batch size grows, while activation remains negligible.

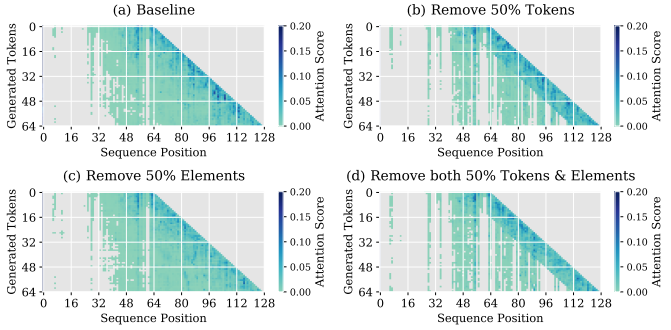

Fig. 5. Visualization of the average attention scores of qwen2.5-0.5b [2] for step-by-step generated tokens with zero-attention regions in gray: (a) baseline; (b) removing 50% unimportant tokens; (c) removing 50% unimportant elements per token; (d) removing both 50% unimportant tokens and elements.

generating sequences in long-context scenarios under various batch size requests. As shown in Fig. 4, the KV cache dominates memory accesses compared to other components, i.e., pre-trained weights and activations, especially as batch size increases. For instance, in llama3-8b-instruct model [3], KV cache consumes around 20% of memory access within a single batch but escalates sharply to over 90% at a batch size of 64. This dramatic increase occurs because each request in a batch requires its own KV cache in every attention layer while model weights can be shared across requests. These dominant off-chip memory accesses not only incur high latency but also consume significant energy, making KV cache optimization crucial for efficient LLM decoding.

C. KV Cache Compression Techniques

Given the memory bottleneck discussed above, recent research has explored various importance-based strategies to reduce KV cache footprint, which can be broadly divided into two types. Heuristic methods [14]–[19] employ predefined eviction policies without dynamic importance computation. StreamingLLM [14] retains only recent tokens and a fixed number of initial attention sink tokens, avoiding expensive scoring operations. DuoAttention [15] separates attention heads into retrieval and streaming types. SepLLM [17] segments sequences into compressed representations, and Longformer [19] introduces local-global attention patterns. However, these rigid windowing or pattern-based strategies often sacrifice accuracy in long-context scenarios where important information may reside beyond predefined boundaries.

To solve the issues, importance-aware methods [20]–[27], [43] dynamically track token significance through attention score accumulation. H2O [20] maintains a global importance score for each token by accumulating attention weights across all decoding steps, evicting tokens with the lowest scores when cache capacity is exceeded. Scissorhands [21] exploits the persistence of importance across layers, KVO-LLM [22] optimizes importance tracking for batched inference, Keyformer [23] selects key tokens for efficient generation, and ALISA [24] accelerates inference through sparsity-aware caching. While these approaches demonstrate superior accuracy-compression trade-offs compared to heuristic methods, they introduce substantial sorting overhead: maintaining importance scores across potentially thousands of tokens at each decoding step becomes a performance bottleneck.

D. Hardware-Accelerated KV Cache Processing

Recognizing the sorting overhead introduced by importance-aware methods, prior arts [25]–[31], [44] have explored dedicated hardware solutions to accelerate KV cache management. Token-Picker [27] proposes a dedicated hardware accelerator that prunes redundant KV token accesses through probability estimation, incorporating specialized sorting units to reduce token selection latency. However, Token-Picker still operates exclusively at the token-level granularity: once a token is selected, its entire vector representation must be loaded from external memory. Consequently, despite eliminating over half of the V cache accesses, it requires loading a substantial portion of the K cache, limiting overall memory bandwidth reduction. MATA [25] designs memory-efficient attention through look-back KV cache pruning. UniCAIM [26] proposes a unified architecture with static-dynamic cache pruning. DIAS [28] exploits distance-based attention sparsity with tree-like processing-in-memory. VEDA [29] enables efficient generation through voting-based eviction and dataflow-flexible acceleration. Moreover, Titanus [44] enables KV cache pruning and quantization on-the-fly.

While these works demonstrate the value of specialized hardware for KV cache optimization, none systematically combine hierarchical importance-aware optimization across both token and element granularities with unified hardware support. This motivates our hierarchical approach that eliminates redundancy at multiple granularities through a two-stage importance-aware mechanism with dedicated hardware acceleration.

III. Hierarchical Importance-Aware KV Cache

This section elaborates on the proposed HiKV technique, a hierarchical importance-aware KV cache. We first analyze the distribution patterns of attention scores to reveal optimization opportunities, then introduce the algorithm designs at both coarse-grained and fine-grained levels, and finally demonstrate how they integrate into the LLM decoding pipeline.

A. Importance Analysis of KV Cache

Recent studies [14], [20], [45] have demonstrated that KV cache tokens exhibit distinct importance to token generation

accuracy. Motivated by this observation, we conduct a comprehensive analysis to further explore potential optimization opportunities. Attention scores P, where $P = \text{softmax}(QK^T)$, naturally reflect the contribution of each token during the LLM decoding process [46]. Using this metric, we first analyze the attention score distribution patterns of qwen2.5-0.5b [2] model in the NarrativeQA task [7] with a simple case. It is with 8 input sequences, each consisting of 64 prefilled tokens and 64 generated tokens. As shown in Fig. 5, our analysis uncovers optimization opportunities not only between tokens but also within individual token elements.

Fig. 5(a) shows attention scores are non-uniform, with certain tokens receiving higher attention weights in dark blue regions while others have minimal impact. Furthermore, as depicted in Fig. 5(b), when removing 50% of input K cache tokens with the lowest attention scores, the primary attention patterns in the dark blue region remain preserved, demonstrating that a substantial portion of tokens contribute minimally to attention computation and can be safely excluded from the cache. While prior importance-aware methods [14], [20], [25] exploit only this between-token redundancy and stop at the token level, we further observe substantial redundancy within the elements of each token. As shown in Fig. 5(c), preserving only the top 50% elements of the input Q token, measured by their magnitude, maintains the overall attention distribution pattern, indicating that many elements within each token have negligible impact on attention scores. Most importantly, these two levels of optimization can be orthogonally combined for higher compression ratios. Fig. 5(d) demonstrates that applying both 50% compression at token and element levels, which achieves 4× total compression ratio, still preserves key attention patterns. These observations reveal that we can only load and compute important portions of KV cached tokens to avoid redundant memory accesses and computations, motivating our proposed hierarchical two-stage approach to KV cache management.

B. Stage I: Coarse-Grained Token–Level Management

To exploit redundancy between tokens, Stage I implements a token-level management strategy for KV cache. It retains only important tokens within a fixed cache budget, thereby reducing the number of tokens accessed from external memory during decoding.

Given a target compression ratio r_I of the KV cache and prefilling sequence length L_p, we establish a fixed KV cache budget as $B_I = L_p/r_I$. The attention score vector P quantifies the importance of prior tokens in generating the current token. The key challenge is to track token importance accurately while keeping the overhead low. Existing token-level methods [14], [20], [25] achieve only one. Static rules [14] are cheap but drop later-important tokens, while dynamic methods like H2O [20] stay accurate but re-sort all tokens at every step.

Instead, we propose a localized accumulation strategy that exploits token IS more efficiently. The budget B_I is partitioned equally into a recent token bank $\mathcal{R}$ and an important token bank $\mathcal{I}$. Notably, $\mathcal{R}$ stores the latest consecutive tokens of the generation sequence, while $\mathcal{I}$ stores historically significant

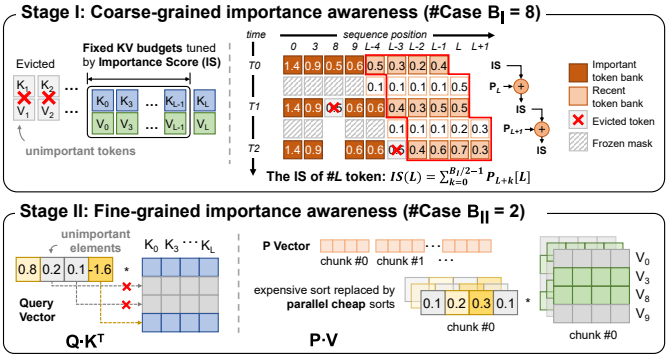

Fig. 6. HiKV applies orthogonal compression at two independent granularities at each decoding step. Stage I maintains a dual-bank structure to evict unimportant tokens, reducing the number of tokens loaded. Stage II selects significant elements via Q and P vector sorting, reducing the volume loaded per token. The two stages act on different dimensions of the KV cache and are jointly formalized in Algorithm 1.

tokens excluding those in $\mathcal{R}$. Rather than global updating IS in H2O [20], we only accumulate P vectors within $\mathcal{R}$. As shown in the upper part of Fig. 6, for a token at position ℓ in the recent bank $\mathcal{R}$, we compute its IS by accumulating attention weights across the $B_I/2$ decoding steps from its generation step onward: $\text{IS}(\ell) = \sum_{k=0}^{B_I/2-1} P_{\ell+k}[\ell]$, where $P_{\ell+k}[\ell]$ is the attention weight received by token ℓ at step $\ell + k$.

The upper part of Fig. 6 further presents the accumulation process across three consecutive timestamps T0, T1, and T2, showing how IS evolves for tokens in the recent bank. When a new token is generated and fed into the recent token bank, the oldest token in this bank is ejected and evaluated. If its IS exceeds the minimum IS in the important bank, the token is promoted to the important token bank, evicting the token with the lowest IS there. Otherwise, it is also evicted from the important token bank. Notably, all the IS in the important token bank are frozen to enable efficient eviction maintenance with the use of a min-heap data structure [47], greatly reducing token eviction complexity from $O(B_I)$ to $O(logB_I)$. This freezing keeps the maintenance efficient with minor effect on adaptivity. Within a single-turn generation, the persistence of importance [21] makes a token losing relevance after being important uncommon, while across multi-turn conversations HiKV recomputes the IS of all prior tokens at each turn boundary to demote stale tokens and re-admit revived ones.

C. Stage II: Fine-Grained Element-Level Management

Building on the observation that many elements within tokens have negligible impact on attention scores, Stage II further reduces memory transfer volume through element-level selection. Unlike prior token-level methods [14], [20], [25] that treat a token as an indivisible unit and must fetch its complete K/V vector once retained, Stage II selects only the significant elements within each retained token. The key insight stems from the asymmetric scale of memory access in attention computation: during decoding, Q and P are single vectors whose memory footprint is minimal, while K and V remain as matrices stored in external memory. Even with batch processing, K and V caches are not shared across requests, making

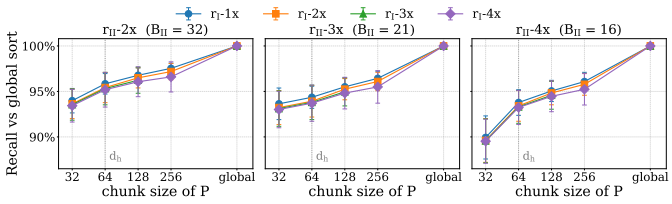

Fig. 7. Chunk-based sorting recall on Qwen2.5-0.5B-Instruct, evaluated on 8 samples from the 2wikimqa task in LongBench. Each point is the mean over all query heads, layers, and samples. Error bars show the standard deviation.

matrix access the dominant memory bottleneck. Hence, we leverage the element magnitudes of Q and P to identify which corresponding positions in K and V matrices are significant, enabling selective loading that substantially reduces memory transfer volume.

We define a selected element budget $B_{II} = d_h/r_{II}$, where d_h denotes the attention head dimension and r_{II} is the target element-level compression ratio. This determines the number of elements to retain per Q token for QK^T computation.

Yet, it is important to notice that the Q token and P vector require different sorting strategies to determine their most significant elements, stemming from their distinct characteristics as shown in the lower part of Fig. 6. Formally, let $\mathrm{Top}_{B_{II}}(\cdot)$ return the indices of the B_{II} largest-magnitude entries of a vector. Since each Q token dimension equals the model's head dimension d_h, which is typically limited to 64 or 128, global sorting is feasible in an efficient way. We thus select a single index set $\mathcal{E}_Q = \mathrm{Top}_{B_{II}}(Q)$ over the whole Q token, which guides the loading of the corresponding K cache for QK^T. P vectors, however, pose a greater challenge as their length can reach thousands of elements even after Stage I optimization, making global sorting computationally prohibitive. To address this challenge, as shown in the bottom of Fig. 6, we introduce a chunk-based sorting strategy that turns the expensive global sorting into multiple independent local sortings. Specifically, let N be the length of the P vector after Stage I. We partition P into $C = \lceil N/d_h \rceil$ chunks of size d_h matching the Q token dimension, and select within each chunk independently, $\mathcal{E}_P^{(c)} = \mathrm{Top}_{B_{II}}\left(P^{(c)}\right)$ for $c = 1, \ldots, C$, where $P^{(c)}$ is the c-th chunk. The selected indices $\{\mathcal{E}_P^{(c)}\}_{c=1}^{C}$ then guide the loading of the corresponding V cache for PV. On the V side, since each position in P corresponds to a specific token, the selected elements are chunk-local token rows rather than sub-token dimensions. This chunk-based design reduces both memory access and computation. Since only $B_{II} = d_h/r_{II}$ of the d_h elements (i.e., token rows) per token are loaded and computed, both the V cache memory access and the PV computation drop by a factor of r_{II}, and the selection cost of the P vector is lowered from $O(N \log N)$ for a global sort to $O(N \log d_h)$ across C parallelizable chunks.

To verify that this approximation is near-lossless, we compare the elements selected by chunk-based sorting against those selected by a global sort over the full P vector on 8 samples from Qwen2.5-0.5B-Instruct, varying r_I and r_{II} from $1\times$ to $4\times$. As shown in Fig. 7, for example, at r_I=4×, r_{II}=2× with chunk size equal to d_h, the chunk-based selection achieves over 95% recall, i.e., it retains over 95% of the elements that a global sort would select. The overlap remains stable across all tested r_I values, indicating that the chunking error is independent of the Stage I compression ratio.

Algorithm 1 HiKV Attention Computation at Decoding Step t

Input: Query Q_t, KV cache ($\mathcal{R}$: Recent bank, $\mathcal{I}$: Important bank)
Output: Attention output O_t, KV cache with updated $\mathcal{R}$ and $\mathcal{I}$

1: // QK^T Computation with hierarchical selection
2: tokens $\leftarrow \mathcal{R} \cup \mathcal{I}$ ▷ Stage I
3: elements$_Q$ $\leftarrow$ Select top-B_{II} indices of Q_t ▷ Stage II
4: Load K[tokens, elements$_Q$] from external memory, where each element in elements$_Q$ is a head dimension
5: $P_t \leftarrow \mathrm{Softmax}(Q_t \cdot K^T)$
6: // Maintenance for $\mathcal{R}$ and $\mathcal{I}$
7: IS$[\ell] \leftarrow$ IS$[\ell] + P_t[\ell]$, $\forall \ell \in \mathcal{R}$ ▷ Stage I
8: evi $\leftarrow$ oldest token in $\mathcal{R}$ ▷ Stage I
9: Promote evi to $\mathcal{I}$ if IS[evi] $> \min\{\mathrm{IS}[j] : j \in \mathcal{I}\}$ ▷ Stage I
10: // PV Computation with hierarchical selection
11: Partition P_t into C chunks
12: **for all** chunk $c \in \{1 \ldots C\}$ **do**
13: elements$_c$ $\leftarrow$ Select top-B_{II} indices from chunk c ▷ Stage II
14: **end for**
15: Load V[tokens, $\{\text{elements}_c\}_{c=1}^{C}$] from external memory, where each element in elements$_c$ is a token row.
16: $\mathrm{O}_t \leftarrow P_t \cdot V$
17: Append (K_t, V_t) of newly generated token to $\mathcal{R}$
18: **return** O_t, Updated $\mathcal{R}$ and $\mathcal{I}$

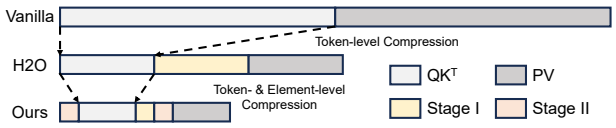

Fig. 8. Operation flow of one decoding step for vanilla, H2O [20], and HiKV.

D. Hierarchical Decoding Integration

Having established the individual mechanisms of token-level and element-level optimization, we elaborate how these two stages collaborate throughout the LLM decoding pipeline to systematically reduce KV cache memory access overhead.

Before the decoding phase starts, HiKV leverages the attention scores generated during the prefilling phase to initialize the dual-bank structure. The last $B_I/2$ consecutive tokens from the prefill sequence are directly placed into the recent bank $\mathcal{R}$, while from the remaining $L_p - B_I/2$ tokens, the top $B_I/2$ tokens, with the highest ISs following the IS definition in Sec. III-B, populate the important bank $\mathcal{I}$. Since prefill processes the prompt in parallel, each prefill token's IS is computed in one pass by summing its attention column over the $B_I/2$-step window. This one-time initialization incurs minimal overhead, as dual-bank construction requires only an additional $O(L \log L)$ sorting operation, negligible compared to prefill's inherent $O(L^2)$ attention computation [33], and prefill itself constitutes a small fraction of total inference time in long-context scenarios.

During the decoding phase, Algorithm 1 presents the hierarchical optimization workflow for attention computation executed at each token generation step. The process begins with QK^T computation. Stage I determines the token set $\mathcal{R} \cup \mathcal{I}$ (Line 2) and Stage II identifies the top-B_{II} element indices of Q_t by magnitude, producing elements$_Q$ (Line 3). The system

selectively loads $K[\text{tokens}, \text{elements}_Q]$ from external memory (Line 4) and computes P_t via softmax (Line 5). The dual-bank maintenance then updates ISs in $\mathcal{R}$ using P_t (Line 7) and evicts or promotes the oldest token against $\min(\mathcal{I})$ via heap comparison (Line 8–9). For PV computation, Stage II partitions P_t into C chunks and selects the top-B_{II} elements (i.e., token rows) from each (Line 11–13), guiding selective V-cache loading (Line 15). The output O_t is computed (Line 16) and (K_t, V_t) is appended to $\mathcal{R}$ (Line 17). Fig. 8 complements Algorithm 1 with an operation-flow view contrasting HiKV against vanilla and H2O [20] decoding, where bar width reflects the relative number of operations and memory accesses. The QK^T and PV bars shrink progressively from vanilla to H2O to HiKV, reflecting token-level compression in H2O and the additional element-level compression in HiKV. Two Stage II sort bars appear in the HiKV row for K-side and V-side element selection, and the Stage I sort bar captures the token eviction overhead, which the min-heap structure reduces to $O(\log B_I)$, making it noticeably narrower than the $O(B_I)$ Stage I bar in H2O. Stage II therefore appears twice as the two halves of a single element-level selection, with the K-side half running before the Stage I eviction. This ordering follows from data dependency rather than the stage indices, as the K-side selection needs only Q_t available at the step start, whereas the Stage I eviction and the V-side selection both consume P_t produced by QK^T. To fully realize these algorithmic benefits, Sec. IV elaborates a custom accelerator design featuring specialized hardware support for various sorting operations in Stage I and Stage II.

IV. HiKV Accelerator Architecture

This section presents the hardware architecture of the HiKV accelerator. The overall architecture centers around the reconfigurable importance sorter (RIS), which flexibly switches between heap-based operations for Stage I and chunk-based parallel sorting for Stage II, jointly supporting the two-stage execution flow of hierarchical KV cache optimization. We detail the system architecture overview, the RIS circuit design, and the RIS operations in Stage I and Stage II, and finally the computing datapath and external memory layout that complete the execution pipeline.

A. System Architecture Overview

The overall architecture of the HiKV accelerator is depicted in Fig. 9, which comprises five major functional blocks: the token importance score generator (TISG), the reconfigurable importance sorter (RIS), the scoreboard buffer, the decoding compute engine (DCE), and the KV cache selection controller. These modules are coordinated by a central computing controller and an external memory manager to maintain dataflow during the autoregressive decoding phase. Specifically, TISG is exclusively activated during Stage I importance maintenance. Upon completion of QK^T computation, it reads P_t directly from the on-chip operand memory and updates the IS of each token in the recent bank via 512 parallel FP16 adders. The resulting ISs are retained in Scoreboard. The DCE comprises 512 FP16 MAC units in CU and 64 FP32 floating-point

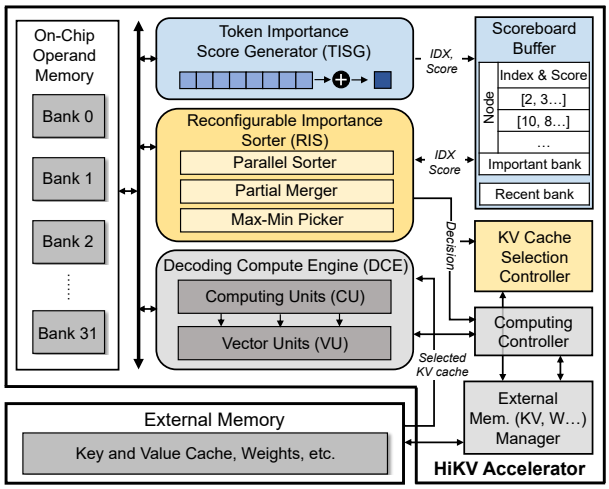

Fig. 9. HiKV accelerator architecture, where the highlighted components, the RIS, scoreboard buffer, and TISG, constitute the HiKV-specific overhead that enables hierarchical two-stage KV cache optimization.

units in VU, responsible for GeMV and non-linear operations, respectively. The RIS serves as the core innovation of HiKV design and is elaborated in the following subsections.

The on-chip operand memory is organized as 32 independent banks, providing unified on-chip storage for intermediate data throughout the decoding process, including loaded KV cache tokens, query vectors, attention score vectors. The scoreboard buffer maintains two logical regions. The recent token bank is implemented as a FIFO structure that tracks the index and accumulated IS of the most recent $B_I/2$ tokens. The important token bank is a min-heap-based register array that retains the index and frozen IS of the $B_I/2$ tokens with historically significant ISs. By preserving only token indices and their corresponding accumulated or frozen IS, the scoreboard acts as a lightweight metadata structure to guide selective cache access, while the full KV tensors reside exclusively in external memory.

During attention computation, the RIS is invoked three times to drive the two-stage importance selection, operating sequentially in Stage II, Stage I, and Stage II modes within a single decoding step, as described in Algorithm 1. For the QK^T computation, the RIS first performs top-B_{II} sorting on Q_t in Stage II mode, and the resulting index mask is forwarded to the KV cache selection controller. Concurrently, the token indices maintained in the scoreboard guide the external memory manager to selectively load the corresponding K cache entries. After the VU produces P_t via softmax, the TISG updates the IS registers accordingly, and when the recent bank FIFO ejects its oldest token, the RIS switches to Stage I mode to perform heap maintenance. For the PV computation, the RIS switches back to Stage II mode, performing parallel top-B_{II} sorting across all chunks of P_t, whose output masks guide the selective loading of V cache entries. For computations outside the attention mechanism, the RIS and TISG are bypassed.

B. Reconfigurable Importance Sorter (RIS)

The RIS is the core innovation of the HiKV accelerator, comprising three core functional modules as illustrated in

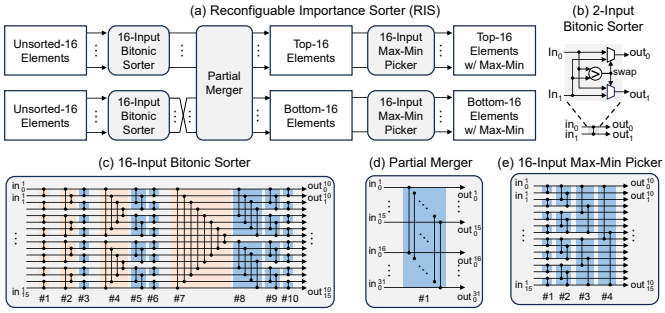

Fig. 10. (a) RIS architecture comprises three reconfigurably combined modules, all built upon (b) a unified 2-input bitonic sorter: the parallel sorter centered on (c) a 16-input bitonic sorter, (d) partial merger, and (e) max-min picker.

Fig. 10(a): the parallel sorter, the partial merger, and the max-min picker. These modules share the 2-input bitonic sorter as a common basic building block, as shown in Fig. 10(b), which contains a comparator and a swap unit that routes the larger and smaller inputs to designated outputs based on the comparison result. Built upon this basic unit, the aforementioned modules are combined in different topological configurations to form the complete RIS circuit, with a critical path of 15 parallel comparison stages when fully activated. Owing to the unified 2-input bitonic sorter building block, the input scale of the RIS can be flexibly extended to accommodate different application scenarios.

A 16-input bitonic sorter, as shown in Fig. 10(c), sorts 16 unordered inputs into an ordered sequence and is organized as 80 dual-input bitonic sorters following the topology of a bitonic sorting network [48], corresponding to 10 parallel comparison stages each containing 8 parallel comparators. The parallel sorter comprises two such 16-input bitonic sorters operating in parallel, enabling simultaneous processing of two independent 16-element input vectors. A direction switcher is attached to the output of the parallel sorter. It selects, via a control signal, whether to reverse the output ordering, converting an ascending sequence into a descending one.

The partial merger, as shown in Fig. 10(d), accepts two 16-element sorted sequences as input: one in ascending order and one in descending order. Through a single parallel comparison stage, it performs element-wise comparison and redistribution between the two sequences, routing the global top-16 elements to the upper output and the bottom-16 elements to the lower output. Notably, elements within each output group remain unordered, a deliberate design choice that avoids the additional overhead of complete intra-group sorting.

The max-min picker processes each group of 16 unordered elements produced by the partial merger, as shown in Fig. 10(e). Built upon the 2-input bitonic sorter, it identifies the minimum and maximum values within each group through 4 parallel comparison stages, placing them at the first and last positions respectively, while remaining elements stay unordered. Retaining intra-group disorder is deliberate, as subsequent heap and selection operations require only the boundary values to be precisely located, rendering complete intra-group ordering unnecessary and thus eliminating the

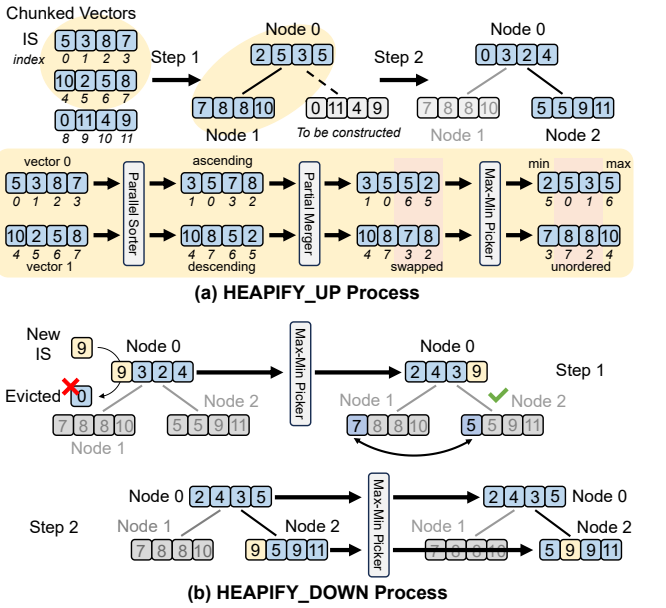

Fig. 11. Stage I token importance tracking via RIS-enabled heap operations: (a) HEAPIFY_UP constructs the vectorized min-heap node by node; (b) HEAPIFY_DOWN maintains the heap upon token eviction, traversing only a single root-to-leaf path rather than scanning all tokens.

associated circuit overhead.

Through differential control signal configurations across the modules, the RIS efficiently supports four operational modes: HEAPIFY_UP for heap construction during the prefill phase, HEAPIFY_DOWN for heap maintenance, GLOBAL_SORT for global sorting of the Q vector, and CHUNK_SORT for chunk-based sorting of the P vector. Among these, HEAPIFY_UP and HEAPIFY_DOWN serve the token-level importance maintenance of Stage I, while GLOBAL_SORT and CHUNK_SORT serve the element-level selection of Stage II, as elaborated in Sec. IV-C and Sec. IV-D, respectively.

C. Stage I: Heap-Based Token Importance Tracking

Stage I invokes two operational modes of the RIS to dynamically maintain the important token bank: HEAPIFY_UP for initial heap construction at the end of the prefill phase, and HEAPIFY_DOWN for ongoing heap maintenance. These operations, together with the per-step IS accumulation, are lightweight. Each decoding step adds only a single vector addition for the IS accumulation and a one-path HEAPIFY_DOWN update, while HEAPIFY_UP is a one-time prefill cost amortized over decoding.

The RIS employs a parallel comparison network that enables multiple element pairs within a single parent-child node comparison to be processed in parallel, thereby increasing the parallelism and execution efficiency of heap operations. To illustrate the detailed execution flow of both modes, the following examples adopt a 4-way parallel configuration for clarity. Each node in the heap stores a vector of 4 IS along with their corresponding token indices, denoted as $[\cdot]_{(\cdot)}$ where the subscript represents the token indices. For node i in the important bank, its left and right children are indexed as $2i+1$

and $2i+2$, respectively, and its parent is indexed as $\lfloor (i-1)/2 \rfloor$. Note that this vectorized heap requires that the IS_{max} of any parent node does not exceed the IS_{min} of its children, with IS_{min} and IS_{max} placed at the first and last positions of each node vector, respectively, while the remaining elements stay unordered.

During the HEAPIFY_UP phase, the RIS fully activates all modules to construct the parent-child hierarchy of the heap in a node-by-node manner. As illustrated in Fig. 11(a), taking the initial comparison of two candidate nodes $[5,3,8,7]_{(0,1,2,3)}$ and $[10,2,5,8]_{(4,5,6,7)}$ as an example: both nodes are fed into the parallel sorter, where the first is configured via the direction switcher to produce an ascending output $[3,5,7,8]_{(1,0,3,2)}$, and the second a descending output $[10,8,5,2]_{(4,7,6,5)}$. The two sorted outputs are then forwarded to the partial merger, which performs element-wise comparison and redistribution in a single parallel comparison stage, yielding a smaller-element group $[3,5,5,2]_{(1,0,6,5)}$ and a larger-element group $[10,8,7,8]_{(4,7,3,2)}$. Both outputs are subsequently processed by the max-min picker, which identifies the minimum and maximum values within each group, producing Node 0 $[2,5,3,5]_{(5,0,1,6)}$ with IS_{min}=2 and IS_{max}=5, and Node 1 $[7,8,8,10]_{(3,7,2,4)}$ with IS_{min}=7 and IS_{max}=10. As each new node is inserted into the heap, the above comparison process is repeated bottom-up against its parent node, until the IS_{min} of the current node is no less than the IS_{max} of its parent, completing the hierarchical heap construction.

During the HEAPIFY_DOWN phase, whenever the recent bank FIFO ejects its oldest token, it must be determined whether IS_{new} of that token should replace an existing element in the important bank heap. Since the positions of IS_{min} and IS_{max} within the root node are already known, a state machine rapidly branches into one of two cases. When $\text{IS}_{\text{new}} < \text{IS}_{\text{min}}$, the RIS is not invoked, and the token is discarded directly. When $\text{IS}_{\text{new}} > \text{IS}_{\text{min}}$, HEAPIFY_DOWN is triggered, as illustrated in Fig. 11(b). Taking $\text{IS}_{\text{new}}=9_{(12)}$ as an example. It is compared against Node 0 $[0,3,2,4]_{(8,1,5,10)}$, satisfying $\text{IS}_{\text{new}} > \text{IS}_{\text{min}}$. $\text{IS}_{\text{new}}=9_{(12)}$ first replaces IS_{min} in Node 0, yielding $[9,3,2,4]_{(12,1,5,10)}$. Only the max-min picker is then activated to relocate the boundary elements, with the parallel sorter and partial merger both bypassed, updating Node 0 to $[2,3,4,9]_{(5,1,10,12)}$. The heap property is then checked by comparing Node 0's IS_{max} against the IS_{min} values of its children: since Node 1 has IS_{min}=7 and Node 2 has IS_{min}=5, the right child (Node 2) is selected as the propagation direction. As Node 0's IS_{max}=9 > Node 2's IS_{min}=5, a heap violation is detected. Node 2's IS_{min} therefore replaces Node 0's IS_{max}, updating Node 0 to $[2,3,4,5]_{(5,1,10,0)}$, while $\text{IS}_{\text{new}}=9_{(12)}$ is placed at Node 2's former IS_{min} position, updating Node 2 to $[9,5,9,11]_{(12,5,11,9)}$; the parallel sorter and partial merger are again bypassed, and only the max-min picker is activated to process Node 2. This process iterates down the heap until the IS_{max} of the current node does not exceed the IS_{min} of either of its children, at which point heap maintenance is complete.

D. Stage II: Chunk-Based Element Importance Selection

Stage II performs global sorting over the Q vector within the attention head dimension, and chunk-wise sorting over the P vector using the head dimension as the chunk size, extracting the top-B_{II} most important elements from each chunk to drive selective loading from external memory. To this end, the RIS supports two operational modes: GLOBAL_SORT for globally sorting the Q vector prior to QK^T computation, and CHUNK_SORT for independently sorting each chunk of the P vector prior to PV computation. Since both modes identically invoke the RIS, the following description applies to both. In this configuration, the RIS activates the parallel sorter and the partial merger, while the max-min picker is bypassed. For input P vectors that do not fill a complete chunk, the missing elements are padded with negative infinity to ensure they are excluded from the top-B_{II} selection.

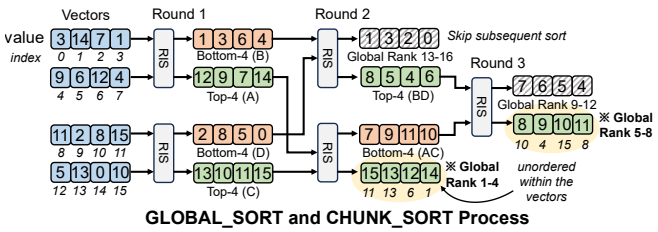

Fig. 12. Stage II element importance selection via RIS-enabled chunk-based sorting: GLOBAL_SORT and CHUNK_SORT proceed over multiple rounds of RIS invocations, skipping intervals confirmed outside the top-k range early.

As illustrated in Fig. 12, and following the same convention as Sec. IV-C, the following example adopts a 4-way parallel configuration for clarity. Taking the extraction of the top-8 elements from 16 inputs as an example, the process completes in three rounds comprising five RIS invocations in total. In Round 1, the 16 elements are partitioned into four groups of 4, which are paired and processed in two parallel RIS invocations, yielding Top-4(A) and Bottom-4(B) from the first pair, and Top-4(C) and Bottom-4(D) from the second. In Round 2, Top-4(A) and Top-4(C) are fed into the RIS, producing the global Top-4 and Mid-4(AC); Bottom-4(B) and Bottom-4(D) are similarly processed, yielding Mid-4(BD) and the global Bottom-4. In Round 3, Mid-4(AC) and Mid-4(BD) are fed into the RIS, producing the elements ranked 5th–8th and those ranked 9th–12th. At this point, all 16 elements are partitioned into four ranked intervals, and the global top-8 is obtained by merging the first two intervals. The corresponding top-B_{II} mask is generated by the KV cache selection controller according to the configured B_{II}. In practice, both the input vector dimension and B_{II} are configurable, and the number of RIS invocations adjusts accordingly. A larger d_h can be handled by increasing the number of invocation rounds, while the RIS needs to be enlarged when the added sorting latency exceeds the external memory access latency and becomes a bottleneck.

Stage II requires only that elements be correctly partitioned into their respective ranked intervals. Once an interval is confirmed to fall entirely outside the top-B_{II} range, it requires no further RIS invocations and is skipped directly. Since boundary element identification within each group is unnecessary, the max-min picker is fully bypassed, reducing unnecessary logic switching and thereby lowering dynamic power consumption.

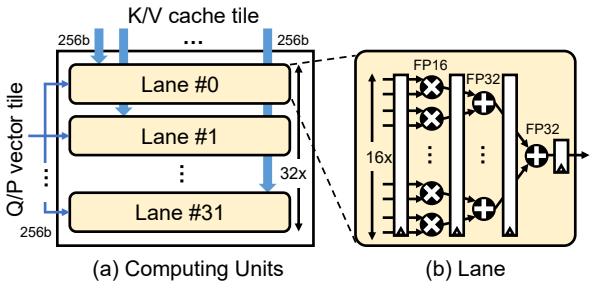

Fig. 13. The CU comprises (a) 32 parallel lanes, each of which (b) processes a partial dot-product via 16 FP16 multipliers and an adder tree.

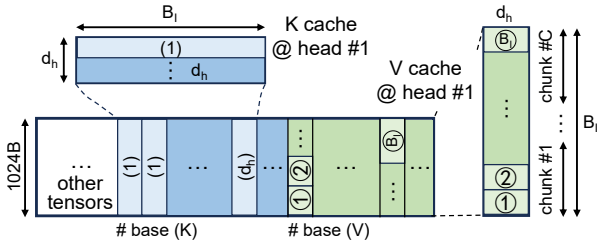

Fig. 14. Off-chip KV cache layout for one attention head.

Together with the heap maintenance operations of Stage I, the same RIS circuit serves the hierarchical KV cache optimization flow through differential control signal configurations, each realizing distinct sorting semantics.

E. Computing Datapath and External Memory Layout

Beyond the RIS, the HiKV accelerator also relies on the computing datapath and the external memory layout to complete the execution pipeline.

The sparse index masks produced by the RIS guide the CU in the DCE to execute compressed GeMV over only the selected KV cache entries, reducing both DRAM transfers and multiply-accumulate operations proportionally. As shown in Fig. 13, the CU comprises 32 parallel lanes, each with 16 FP16 multipliers and an adder tree that reduces to a FP32 partial dot-product. One Q or P vector tile of 16 elements is broadcast to all lanes simultaneously, while each lane receives a distinct K or V cache tile of 16 elements as its unicast input. For QK^T, the 32 lanes each compute the inner product between Q_t and one K token along the d_h dimension, producing 32 attention logits in parallel. The resulting logits are then passed to the VU, which executes softmax to produce P_t. For PV, the 32 lanes each accumulate one output element along the B_I token dimension, producing 32 output elements in parallel.

To serve the resulting hierarchical access patterns of both stages efficiently, the K and V caches are organized in different DRAM layouts, as shown in Fig. 14. The K cache uses an element-first layout, where all B_I token values for a given element position are stored contiguously. Stage II's selection of B_{II} element positions therefore translates directly into B_{II} fully contiguous burst transfers. The V cache uses a chunk-indexed layout, where within each of the C chunks, token entries are stored contiguously by token position. Stage II reads only the B_{II} selected token entries per chunk, and in the worst case these entries span different DRAM rows, causing a row miss that cannot amortize the row-activate latency. Since both stages operate at token-block granularity and Stage II element-level accesses remain within individual token blocks, the two-stage access pattern is naturally compatible with block-wise memory management schemes such as PagedAttention [49].

V. Experimental Results

This section evaluates the HiKV co-design across algorithmic, hardware, and system-level dimensions. Following the description of the experimental setup, we first analyze the algorithmic accuracy-compression trade-off. We then quantify the hardware-level area and power efficiency of the implementation. To demonstrate the system-level benefits, we measure the reductions in external memory traffic and improvements in end-to-end inference performance. Finally, we provide a comparative analysis against SotA KV cache accelerators to highlight the superior efficiency of the HiKV co-design.

A. Experimental Setup

Algorithmic Evaluation. We evaluate HiKV's algorithmic performance across four representative LLMs: mistral-7b-instruct-v0.2 [1], llama3-8b-instruct [3], longchat-7b-v1.5-32k [4], and qwen2.5-0.5b-instruct [2]. The evaluation utilizes a selected subset of ten tasks from the LongBench benchmark suite to represent diverse scenarios in question answering, summarization, and reasoning. We fix the prefill prompt length at 4096 tokens and employ single-request inference with a batch size of one to focus on the decoding phase. To comprehensively characterize the accuracy-compression trade-off, we sweep the Stage I token-level compression ratio r_I and Stage II element-level compression ratio r_{II} from $1\times$ to $4\times$. The total compression ratio is defined as $r_{total} = r_I \times r_{II}$. The primary baseline for comparison is the vanilla KV cache implementation, which maintains all KV tensors in FP16 precision without any eviction or selection. We define a performance degradation of less than 1% compared to this vanilla baseline as the requirement for maintaining algorithmic integrity. We also include SotA importance-based methods such as StreamingLLM (S-LLM) [14] and H2O [20] to benchmark our hierarchical strategy.

Hardware Implementation. The HiKV accelerator is implemented in Verilog and synthesized using the Cadence Genus tool. We target the TSMC 16nm technology node with an operating frequency of 300 MHz and a supply voltage of 0.8V. Functional verification is performed using Cadence Xcelium to ensure the correctness of the two-stage sorting and eviction logic across various importance distributions. The reported area and power are obtained from the post-layout design in Cadence Innovus, with power analysis using activity-annotated post-layout netlists from representative workloads. This allows for a precise breakdown of the power consumed by the HiKV-specific components, including the Scoreboard, the RIS, and the TISG. We quantify the area and power overheads of these modules by comparing them against the on-chip computation and memory resources in our HiKV accelerator.

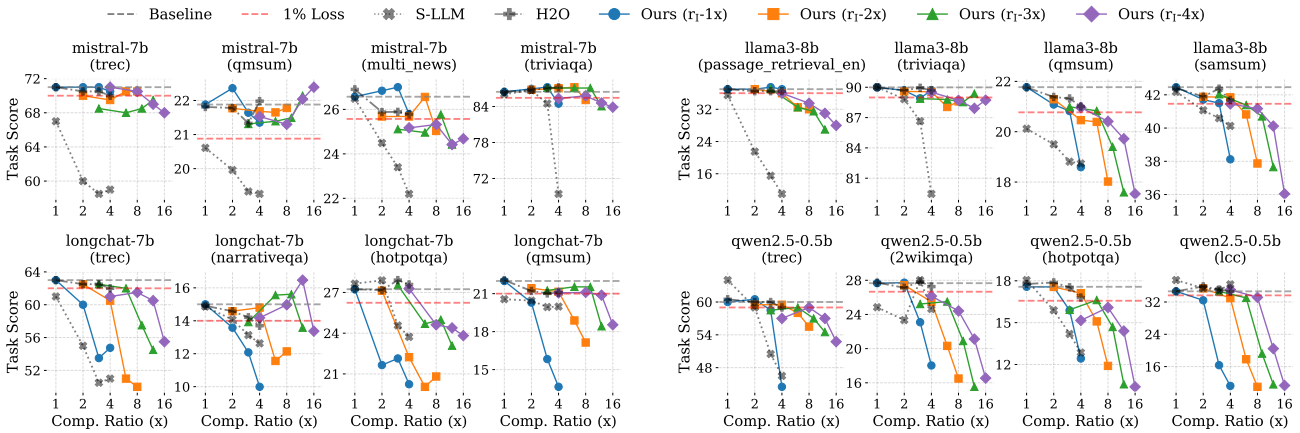

Fig. 15. Accuracy comparison between our HiKV, S-LLM [14], and H2O [20] under various compression settings for various tasks in LongBench [7].

System Simulation. We use HBM2 as the external memory interface with a 1024-bit bus width, an average access energy of 3.9 pJ/bit, and a 256 GB/s peak bandwidth following [25], [31], [44], [50], [51]. The DRAM access is modeled with DRAMSim3 [52]. We leverage the LLM-Viewer performance analyzer to model the end-to-end execution of the decoding phase by incorporating the synthesized area and power parameters of the HiKV-specific modules. To ensure a fair comparison, we configure a unified hardware platform with 512 FP16 MAC units and 2 MB of on-chip SRAM for all evaluated methods. We implement S-LLM [14], H2O [20], and Token-Picker (Token-P) [27] on this identical platform to eliminate variations arising from disparate hardware assumptions. For all evaluated methods, each is configured at the most aggressive compression setting that keeps the average score over selected tasks within 1% of the vanilla baseline. Otherwise, it falls back to the uncompressed baseline. The systemic energy analysis accounts for both the on-chip computation power and the off-chip memory traffic energy under the previously defined 1% accuracy constraint. By measuring external memory access reduction, inference speedup, and energy efficiency, we quantify the end-to-end performance gains of our hierarchical pruning strategy across diverse LLM architectures.

B. Algorithm-Level Evaluation

The algorithmic effectiveness of HiKV is validated through an extensive accuracy-compression trade-off analysis across representative LLM architectures. As shown in Fig. 15, HiKV consistently pushes the Pareto frontier of task performance relative to the total KV cache compression ratio r_{total}, where $r_{total} = r_I \times r_{II}$. In contrast, the SotA importance-aware methods H2O [20] and S-LLM [14] only facilitate compression at the token-level granularity, where r_{total} is equivalent to r_I. This multi-dimensional approach enables HiKV to maintain comparable accuracy with the FP16 baseline even under aggressive total compression settings. For instance, with the mistral-7b on trec task, while H2O sustains the 1% negligible accuracy loss threshold at a $4\times$ compression ratio ($r_I = 4\times$), HiKV leverages its orthogonal Stage II optimization to double the efficiency, achieving an $8\times$ total compression ratio ($r_I = 4\times, r_{II} = 2\times$) under the same accuracy constraint. This 1% threshold is applied per task, and the attainable ratio thus varies with each task's compression sensitivity, ranging from $2\times$ on sensitive multi-hop reasoning tasks to $16\times$ on compression-tolerant ones. Notably, S-LLM fails to meet this 1% requirement even at minimal compression levels across several benchmarks.

The superior robustness of HiKV across diverse tasks stems from the synergistic effect of its two-stage optimization. Unlike the static window-based eviction of S-LLM or the single-granularity dynamic pruning of H2O, HiKV employs a localized accumulation strategy in Stage I to maintain essential tokens, and further eliminates element-level redundancy in Stage II by exploiting the magnitude distribution within vectors. As evidenced by the r_I curves in Fig. 15, prioritizing token-level significance before element-wise selection yields a more robust performance profile than pushing a single granularity to its extreme. The consistent accuracy gains observed across various LLMs also demonstrate that the hierarchical importance assessment is architecture-agnostic. These results show that HiKV preserves the semantic integrity required for complex reasoning while significantly outperforming existing importance-aware compression techniques [14], [20]. We further note that this trade-off can be non-monotonic in a few cases. On LongChat-7B/HotPotQA and Qwen2.5-0.5B/TREC, for example, $(r_I{=}4, r_{II}{=}4)$ exceeds $(r_I{=}1, r_{II}{=}4)$ despite a larger r_{total}. This reflects an interaction between the two stages, as tightening Stage I under aggressive Stage II concentrates the kept tokens on high-IS entries that still dominate the softmax under the element-level approximation. On most other panels, accuracy decreases monotonically with r_{total}.

In our evaluation, r_I and r_{II} are determined by an offline per-task calibration, while both ratios remain runtime-configurable in the HiKV accelerator. This renders HiKV complementary to dynamic budget allocation methods like Ada-KV [53] and DynamicKV [54], whose adaptively allocated token budget is orthogonal to the two compression dimensions of HiKV and can be applied at runtime.

C. Hardware-Level Evaluation

Table I summarizes the implementation specifications, area distribution, and power consumption of the HiKV architecture.

TABLE I
COMPONENT SPECIFICATIONS, AREA, AND POWER BREAKDOWN

Component	Specification	Area (mm^2)	Power (mW)
CU	512 FP16 MAC	0.141 (6.85%)	54.9 (40.5%)
VU	64 × FP32 FPU	0.028 (1.36%)	2.01 (1.48%)
TISG	512-dim adders	0.009 (0.44%)	0.14 (0.10%)
RIS	16-parallel sorter	0.031 (1.51%)	1.93 (1.43%)
Scoreboard	8 × 64b × 2K	0.138 (6.70%)	7.42 (5.48%)
Control	System controls	0.001 (0.05%)	0.02 (0.01%)
Operand Mem.	32 × 128b × 4K	1.710 (83.1%)	69.0 (51.0%)
Others	Others	0.001 (0.05%)	0.01 (0.01%)
Total		**2.059 (100%)**	**135.4 (100%)**

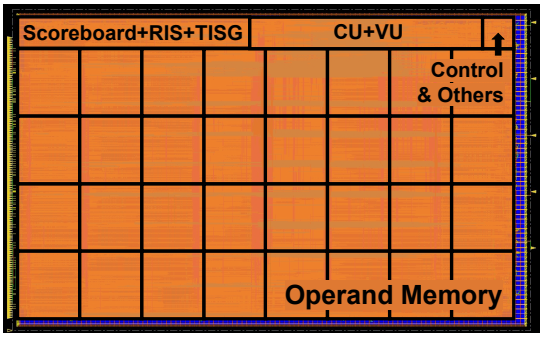

Fig. 16. Post-layout view of the HiKV accelerator in TSMC 16nm.

Implemented in a 16nm CMOS technology node at 300 MHz, the complete HiKV accelerator occupies a total silicon area of 2.059 mm^2 with a power consumption of 135.4 mW. The on-chip operand memory dominates the design, taking 83.1% of the area and 51.0% of the power, while the computation units occupy 6.85% of the area but 40.5% of the power.

Within this resource budget, the hardware overhead specifically introduced by the HiKV hierarchical mechanism is minimal. This mechanism consists of the Scoreboard, the RIS, and the TISG, which collectively account for only 8.64% of the total area and 7.01% of the power. This high area efficiency is achieved through the reconfigurable design of the RIS, which utilizes a unified architecture to support four distinct operational modes across both Stage I and Stage II. By enabling hardware-level execution of hierarchical pruning, the specialized HiKV architecture aims to minimize the dominant DRAM accesses that characterize conventional LLM decoding. The corresponding post-layout view is shown in Fig. 16. This dedicated hardware implementation provides the basis for the systemic evaluations in the following subsection.

D. System-Level Evaluation

External Memory Access Reduction. As shown in Fig. 17, HiKV consistently achieves the lowest memory traffic across all tested scenarios under a strict 1% iso-accuracy constraint. On average, HiKV reduces external memory accesses to 13.94% of the vanilla baseline, significantly outperforming H2O [20] (25.41%), Token-P [27] (38.91%), and S-LLM [14] (67.97%). For instance, on mistral-7b, HiKV compresses traffic to 9.52%, whereas H2O, Token-P, and S-LLM are limited to 25.0%, 38.91%, and 80.0%, respectively. The inefficiency of

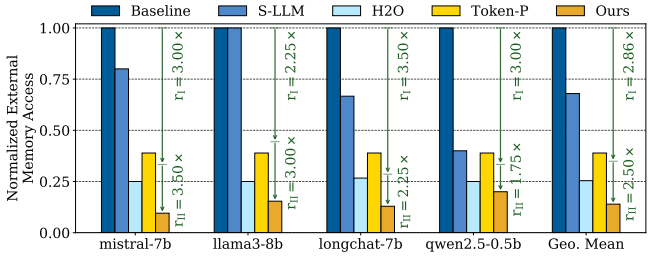

Fig. 17. External memory access normalized to the vanilla baseline. The HiKV reduction is decomposed into the Stage I (r_I) and Stage II (r_{II}) factors.

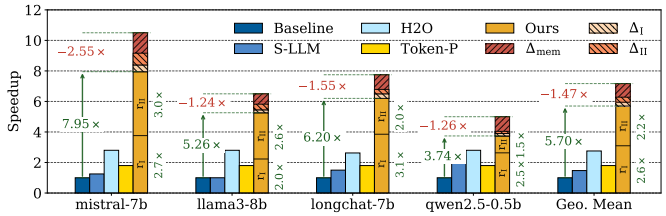

Fig. 18. Inference speedup normalized to the vanilla baseline. For HiKV, the green value is the achieved speedup, split into the Stage I (r_I) and Stage II (r_{II}) factors, and the red value is the overhead, which comes from the memory penalty Δ_{mem} and the sorting cost Δ_I, Δ_{II}.

S-LLM is attributed to its sliding window policy. On llama3-8b model, it fails to satisfy the accuracy constraint at any compression level, as shown in Fig. 15, and thus contributes no memory access reduction on that model. In contrast, HiKV breaks the single-granularity bottleneck of existing dynamic token-level pruning techniques, like H2O and Token-P, by exploiting element-level redundancy in Stage II. By selectively transferring only significant elements rather than complete vectors, HiKV effectively resolves the memory bandwidth bottleneck to enable the substantial speedup observed in the following evaluation. This reduction can be attributed to the two orthogonal stages, as annotated on the HiKV bar in Fig. 17. The token-level Stage I contributes a reduction of r_I = 2.86× and the element-level Stage II contributes r_{II} = 2.50× on geometric average, which jointly bring the overall reduction to 7.17×. Each stage contributes a substantial portion on its own, and the two prunings are largely complementary.

Inference Speedup. As depicted in Fig. 18, HiKV delivers an average speedup of approximately 5.70× across the four representative LLMs, consistently outperforming H2O [20] (2.76×), Token-P [27] (1.80×), and S-LLM [14] (1.47×). For instance, on mistral-7b, HiKV achieves a peak speedup of 7.95×, whereas H2O, Token-P, and S-LLM are limited to 2.80×, 1.80×, and 1.25×, respectively. The performance gains of H2O are partially constrained by the high computational overhead of maintaining global importance scores through frequent sorting operations. Take qwen2.5-0.5b as an example, H2O reduces external memory accesses by 37.5% relative to S-LLM, yet its speedup improvement is only 12%, indicating that the sorting overhead substantially offsets the benefit of its memory access reduction. In contrast, HiKV achieves a 1.34× speedup over H2O, despite only a 1.25× reduction in memory accesses. This outsized gain highlights the ability of

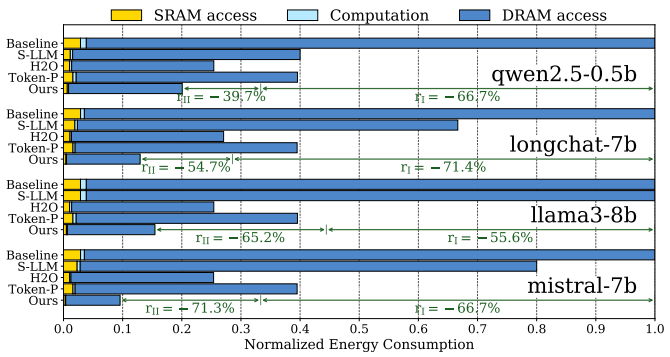

Fig. 19. Energy consumption normalized to the vanilla baseline, split into DRAM, computation, and SRAM. The HiKV reduction is decomposed into the Stage I (r_I) and Stage II (r_{II}) factors.

the RIS hardware to more effectively convert memory savings into inference acceleration. By effectively balancing the trade-off between importance maintenance and memory bandwidth, HiKV realizes substantial inference speedup compared to the SotAs [14], [20], [27]. As annotated on the HiKV bar in Fig. 18, this speedup also separates into a Stage I factor of 2.57× and a Stage II factor of 2.21× on geometric average, reaching 5.70× in total. The speedup lands slightly below the 7.17× access reduction because of the on-chip sorting and memory access overheads of the accelerator. The dominant overhead is the memory penalty from the fine-grained element-level access of Stage II, while the RIS sorting cost is only about 8% and is largely masked by this fine-grained memory access overhead on the critical path.

Energy Consumption. The systemic energy consumption is analyzed by breaking down the operational costs into DRAM access, SRAM access, and computation components. As shown in Fig. 19, HiKV achieves an energy reduction of 80% to 90% relative to the vanilla baseline across all evaluated models. This superior efficiency is primarily driven by the drastic mitigation of off-chip DRAM movement, which accounts for over 96% of the energy in the vanilla baseline. For instance, on the longchat-7b model, HiKV eliminates approximately 87% of the total energy consumption, whereas H2O [20] and Token-P [27] reduce energy by approximately 73% and 60%, respectively. These gains are mainly enabled by Stage II element-level selection. On the mistral-7b model, HiKV reduces the DRAM energy component by 75.5% compared to Token-P. This systemic leap is achieved by performing specialized on-chip importance sorting to prune redundant data before transmission, investing a minimal 7.01% hardware power overhead. By leveraging efficient on-chip processing to minimize energy-expensive external memory transfers, HiKV provides a promising KV cache co-design method for energy-efficient LLM decoding. Looking at the two stages, this energy saving is delivered by the two prunings jointly cutting the dominant DRAM energy. On mistral-7b, Stage I alone reduces the DRAM energy by 66.7% and Stage II alone by 71.4%, and as the two act multiplicatively rather than additively, together they cut 90.5%. Since the on-chip computation and SRAM that include the RIS account for only about 4% of the total energy, this algorithmic external memory access saving is carried over almost directly into the end-to-end energy reduction.

E. Comparison with State-of-the-Arts

Table II provides a systematic comparison of HiKV against SotA KV cache accelerators across algorithmic granularity and hardware implementation. For a fair cross-node comparison, the reported area, power, and energy efficiency of all prior arts are normalized to the 16 nm node of HiKV using DeepScaleTool [55], shown as the $^{[\cdot]}$ superscripts. Since external memory access in the LLM decoding is dominated by KV cache traffic, its reduction serves as the primary efficiency metric for KV cache accelerators, and all comparisons reported below are obtained under 1% accuracy loss constraint. Token-level methods, including Token-Picker [27], VEDA [29], KV-MMU [31], and MATA [25], achieve reductions in the range of 2.57∼5×, reflecting an inherent ceiling of token-granularity compression. These approaches reduce memory traffic by evicting low-importance tokens, yet each retained token needs to be fully loaded with all its elements, leaving intra-token element-level redundancy unaddressed. Although Titanus [44] incorporates element-level operations, the absence of coarse-grained token-level filtering limits its external memory access reduction to only 2.4×, which falls below several token-only designs, showing that single-granularity compression strategies face fundamental limitations regardless of the targeted granularity. HiKV addresses this bottleneck through the reconfigurable design of the RIS, which time-multiplexes two semantically distinct importance sorting requirements within a unified circuit, serving Stage I token-level ranking and Stage II element-level ranking in an alternating fashion. This hardware capability enables the compression gains from both stages to accumulate cooperatively, yielding the highest external memory access reduction in Table II, substantially surpassing VEDA [29] and MATA [25]. The hardware overhead introduced by this hierarchical compression mechanism, comprising the Scoreboard, the RIS, and the TISG, amounts to only 8.64% of the total area, with the RIS alone occupying merely 1.51%, supporting the full hierarchical algorithm at minimal silicon cost. In summary, HiKV is the first design to realize hierarchical granularity compression, and it is precisely the algorithm-hardware co-design methodology that enables this hierarchical compression to be efficiently realized, thereby breaking through the single-granularity bottleneck and achieving the highest external memory access reduction of 7.17× among all prior arts.

VI. Conclusion

This paper presents HiKV, an algorithm-hardware co-design framework for LLM decoding that systematically alleviates the memory bottleneck imposed by the KV cache. Built on the insight that KV cache redundancy exists at both token and element granularities, HiKV employs two orthogonally complementary compression stages, supported by the reconfigurable importance sorter as the core hardware innovation that accommodates the semantically distinct sorting requirements of both stages within a unified circuit, incurring only 8% area

TABLE II
COMPARISON WITH STATE-OF-THE-ART KV CACHE ACCELERATORS

	Token-Picker [27]	Titanus [44]	UniCAIM [26]	KV-MMU [31]	MATA [25]	VEDA [29]	HiKV (Ours)
Venue / Year	DAC'24	GLSVLSI'25	DAC'25	JETCAS'25	TC'25	DAC'25	TCAS-I'26
Optimization Granularity	Token-level	Element-level	Token-level	Token-level	Token-level	Token-level	**Hierarchy-level**
Technology Node (nm)	65	14	45	65	28	28	**16**
Clock Frequency (MHz)	500	200	N/A	800	1000	1000	**300**
Implementation	Pre-layout	Pre-layout	N/A	Pre-layout	Pre-layout	Pre-layout	**Post-layout**
Area Consumption (mm^2) (5)	$8.59^{[0.45]}$	$83.32^{[112.3]}$	N/A	$\sim 49.80^{[2.63]}$ (1)	$4.601^{[1.42]}$	$1.058^{[0.33]}$	**2.059**
Power Consumption (mW) (5)	$3149^{[1136]}$	$31{,}950^{[36,016]}$	N/A	N/A	$461.4^{[297.2]}$	$375.3^{[241.8]}$	**135.4**
Arithmetic Precision	INT12	INT2–3 (mixed)	Analog+10b ADC	INT8	FP16	FP16	**FP16**
Energy Efficiency (TOPS/W) (2),(5)	$2.78^{[7.70]}$	N/A	N/A	$2.59^{[7.18]}$	N/A	$0.65^{[1.01]}$	**2.26**
External Memory Access Reduction (3)	2.57×	2.4×	N/A	2.6×	$\sim 4.09\times$ (4)	5×	**7.17×**

(1) Estimated by summing module-level area breakdowns reported in the original paper.
(2) Computed based on on-chip synthesized power, excluding off-chip DRAM access energy.
(3) Reduction factor of off-chip memory access volume relative to the vanilla full-KV cache baseline.
(4) Estimated as $1.9\times$ (SpAtten vs. full-KV baseline) $\times$ $2.15\times$ (MATA vs. SpAtten, geomean) $\approx 4.09\times$.
(5) Each entry is shown as "reported$^{[*]}$", where $^{[*]}$ denotes the value scaled to the 16 nm node using DeepScaleTool [55].

and power overhead in total. Under less than 1% accuracy loss, HiKV achieves superior compression efficiency over token-level SotA methods, and attains up to 7.95× inference speedup and 90% energy reduction at the system level. Under iso-accuracy constraints, HiKV delivers an additional 1.82∼4.87× external memory access reduction over state-of-the-art methods. As size of context windows continues to grow, HiKV offers a co-design methodology for energy-efficient LLM decoding under increasing KV cache memory pressure.

ACKNOWLEDGMENT

The authors sincerely thank the anonymous reviewers for their constructive feedback, and Xinfa Zheng, Sander Crols, and Robin Geens for their generous technical support. The authors acknowledge EuroHPC JU for awarding the project ID EHPC-DEV-2026D03-124 access to Leonardo BOOSTER hosted by CINECA, Italy, and NVIDIA Corporation for awarding GPU hardware through its Academic Grant Program.

REFERENCES

[1] A. Q. Jiang *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.
[2] A. Yang *et al.*, "Qwen2 technical report," *arXiv preprint arXiv:2407.10671*, 2024.
[3] AI@Meta, "Llama 3 model card," 2024. [Online]. Available: https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md
[4] D. Li *et al.*, "How long can context length of open-source llms truly promise?" in *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*, 2023.
[5] Z. Zhang *et al.*, "Language models are few-shot learners," *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
[6] K. Wang *et al.*, "Mathcoder: Seamless code integration in llms for enhanced mathematical reasoning," in *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.
[7] Y. Bai *et al.*, "Longbench: A bilingual, multitask benchmark for long context understanding," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (ACL)*, 2024, pp. 3119–3137.
[8] Y. Chen *et al.*, "P3-llm: An integrated npu-pim accelerator for llm inference using hybrid numerical formats," in *Proceedings of the 53rd Annual International Symposium on Computer Architecture (ISCA)*, 2026.
[9] L. Haoyang *et al.*, "A survey on large language model acceleration based on kv cache management," *Transactions on Machine Learning Research (TMLR)*, 2025.
[10] Z. Wang *et al.*, "Codec: Prefix-shared decoding kernel for llms," *Proceedings of the ACM on Management of Data (SIGMOD)*, vol. 4, no. 3, pp. 1–27, 2026.
[11] G.-I. Yu *et al.*, "Orca: A distributed serving system for transformer-based generative models," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2022, pp. 521–538.
[12] P. Patel *et al.*, "Splitwise: Efficient generative llm inference using phase splitting," in *ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 118–132.
[13] Z. Liu *et al.*, "Kivi: A tuning-free asymmetric 2bit quantization for kv cache," in *International Conference on Machine Learning (ICML)*, 2024.
[14] G. Xiao *et al.*, "Efficient streaming language models with attention sinks," in *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.
[15] G. Xiao *et al.*, "Duoattention: Efficient long-context llm inference with retrieval and streaming heads," in *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.
[16] Y. Li *et al.*, "Snapkv: Llm knows what you are looking for before generation," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 37, pp. 22 947–22 970, 2024.
[17] G. Chen *et al.*, "Sepllm: Accelerate large language models by compressing one segment into one separator," in *Forty-second International Conference on Machine Learning (ICML)*, 2025.
[18] C. Han *et al.*, "Lm-infinite: Zero-shot extreme length generalization for large language models," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2024, pp. 3991–4008.
[19] I. Beltagy *et al.*, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.
[20] Z. Zhang *et al.*, "H2o: Heavy-hitter oracle for efficient generative inference of large language models," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, pp. 34 661–34 710, 2023.
[21] Z. Liu *et al.*, "Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, pp. 52 342–52 364, 2023.
[22] Z. Li *et al.*, "Kvo-llm: Boosting long-context generation throughput for batched llm inference," in *ACM/IEEE Design Automation Conference (DAC)*, 2025, pp. 1–7.
[23] M. Adnan *et al.*, "Keyformer: Kv cache reduction through key tokens selection for efficient generative inference," *Proceedings of Machine Learning and Systems (MLSys)*, vol. 6, pp. 114–127, 2024.
[24] Y. Zhao *et al.*, "Alisa: Accelerating large language model inference via sparsity-aware kv caching," in *ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 1005–1017.
[25] Z. Zhu *et al.*, "Mata: A memory-efficient attention accelerator for llms exploiting look-back kv cache pruning," *IEEE Transactions on Computers (TC)*, 2025.
[26] W. Xu *et al.*, "Unicaim: A unified cam/cim architecture with static-dynamic kv cache pruning for efficient long-context llm inference," in *ACM/IEEE Design Automation Conference (DAC)*, 2025, pp. 1–7.
[27] J. Park *et al.*, "Token-picker: Accelerating attention in text generation with minimized memory transfer via probability estimation," in *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*, 2024.

[28] Z. Chen *et al.*, "Dias: Distance-based attention sparsity for ultra-long-sequence transformer with tree-like processing-in-memory architecture," in *ACM/IEEE Design Automation Conference (DAC)*, 2025, pp. 1–7.
[29] Z. Wang *et al.*, "Veda: Efficient llm generation through voting-based kv cache eviction and dataflow-flexible accelerator," in *ACM/IEEE Design Automation Conference (DAC)*, 2025, pp. 1–7.
[30] Y. Zhang *et al.*, "Kv-cache oriented query-aware sparse attention accelerator with cross-stage precision-configurable digital cim," *IEEE Transactions on Circuits and Systems II: Express Briefs (TCAS-II)*, 2025.
[31] A. Moradifirouzabadi *et al.*, "End-to-end acceleration of generative models with runtime regularized kv cache management," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS)*, 2025.
[32] M. Huang *et al.*, "Edgellm: A highly efficient cpu-fpga heterogeneous edge accelerator for large language models," *IEEE Transactions on Circuits and Systems I: Regular Papers (TCAS-I)*, 2025.
[33] Y. Zhong *et al.*, "Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024, pp. 193–210.
[34] S. Zeng *et al.*, "Flightllm: Efficient large language model inference with a complete mapping flow on fpgas," in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, 2024, pp. 223–234.
[35] G. Wang *et al.*, "Ofq-llm: Outlier-flexing quantization for efficient low-bit large language model acceleration," *IEEE Transactions on Circuits and Systems I: Regular Papers (TCAS-I)*, 2025.
[36] J. Yuan *et al.*, "Kv cache compression, but what must we give in return? a comprehensive benchmark of long context capable approaches," in *The 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.
[37] S. Ma *et al.*, "Apt-llm: Exploiting arbitrary-precision tensor core computing for llm acceleration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2025.
[38] Z. Zhou *et al.*, "A survey on efficient inference for large language models," *arXiv preprint arXiv:2404.14294*, 2024.
[39] L. Guo *et al.*, "When to stop? towards efficient code generation in llms with excess token prevention," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2024, pp. 1073–1085.
[40] H. Zhang *et al.*, "Llmcompass: Enabling efficient hardware design for large language model inference," in *ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 1080–1096.
[41] R. Geens *et al.*, "Energy cost modelling for optimizing large language model inference on hardware accelerators," in *IEEE 37th International System-on-Chip Conference (SOCC)*, 2024, pp. 1–6.
[42] Z. Yuan *et al.*, "Llm inference unveiled: Survey and roofline model insights," *arXiv preprint arXiv:2402.16363*, 2024.
[43] J. Tian *et al.*, "Skipkv: Selective skipping of kv generation and storage for efficient inference with large reasoning models," in *Proceedings of Machine Learning and Systems (MLSys)*, 2026.
[44] P. Chen *et al.*, "Titanus: Enabling kv cache pruning and quantization on-the-fly for llm acceleration," in *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, 2025, pp. 71–77.
[45] W. Lee *et al.*, "Infinigen: Efficient generative inference of large language models with dynamic kv cache management," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024, pp. 155–172.
[46] L. Ribar *et al.*, "Sparq attention: Bandwidth-efficient llm inference," in *International Conference on Machine Learning (ICML)*, 2024.
[47] J. Williams, "Algorithm 232: Heapsort," *Communications of the ACM*, vol. 7, no. 6, pp. 347–348, 1964.
[48] K. E. Batcher, "Sorting networks and their applications," in *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, 1968, pp. 307–314.
[49] W. Kwon *et al.*, "Efficient memory management for large language model serving with PagedAttention," in *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles (SOSP)*, 2023.
[50] C. Fang *et al.*, "Anda: Unlocking efficient llm inference with a variable-length grouped activation data format," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2025, pp. 1467–1481.
[51] N. P. Jouppi *et al.*, "Ten lessons from three generations shaped google's tpuv4i: Industrial product," in *ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 1–14.
[52] S. Li *et al.*, "Dramsim3: A cycle-accurate, thermal-capable dram simulator," *IEEE Computer Architecture Letters (CAL)*, vol. 19, no. 2, pp. 106–109, 2020.
[53] Y. Feng *et al.*, "Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.
[54] X. Zhou *et al.*, "Dynamickv: Task-aware adaptive kv cache compression for long context llms," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.
[55] S. Sarangi *et al.*, "DeepScaleTool: A tool for the accurate estimation of technology scaling in the deep-submicron era," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.