

Kelle: Co-design KV Caching and eDRAM for Efficient LLM Serving in Edge Computing

Tianhua Xia

Tandon School of Engineering
New York University
New York, NY, USA
tx856@nyu.edu

Sai Qian Zhang

Tandon School of Engineering
New York University
New York, NY, USA
sai.zhang@nyu.edu

Abstract

Running Large Language Models (LLMs) on edge devices is crucial for reducing latency, improving real-time processing, and enhancing privacy. By performing inference directly on the device, data does not need to be sent to the cloud, ensuring faster responses and reducing reliance on network connectivity. However, implementing LLMs on edge devices presents challenges, particularly with managing key-value (KV) caches, which plays a pivotal role in LLM serving. As the input text lengthens, the size of the KV cache increases linearly with the sequence length, leading to a significant memory footprint and data access costs. On the other hand, edge devices have limited memory and computational power, making it hard to store and efficiently access the large caches needed for LLM inference.

To mitigate the substantial overhead caused by KV cache, we propose using embedded DRAM (eDRAM) as the primary storage for LLM serving in edge device, which offers higher storage density compared to SRAM. However, to ensure data integrity, eDRAM needs periodic refresh operations, which are power-intensive. To reduce eDRAM costs and improve overall system performance, we propose *Kelle*, a software-hardware co-design solution optimized for deploying LLMs on eDRAM-based edge systems. Combined with our fine-grained memory eviction, recomputation, and refresh control algorithms, the *Kelle* accelerator delivers a 3.9 \times speedup and 4.5 \times energy savings compared to existing baseline solutions.

CCS Concepts

- Computer systems organization → Neural networks.

Keywords

Large Language Model, Embedded DRAM

ACM Reference Format:

Tianhua Xia and Sai Qian Zhang. 2025. Kelle: Co-design KV Caching and eDRAM for Efficient LLM Serving in Edge Computing. In *58th IEEE/ACM International Symposium on Microarchitecture (MICRO '25), October 18–22, 2025, Seoul, Republic of Korea*. ACM, New York, NY, USA, 16 pages. <https://doi.org/10.1145/3725843.3756071>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MICRO '25, October 18–22, 2025, Seoul, Republic of Korea

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM. ACM ISBN 979-8-4007-1573-0/2025/10
<https://doi.org/10.1145/3725843.3756071>

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities across a wide range of domains. While cloud-based deployments offer advantages like increased processing power, they also come with limitations, including high communication latency and security risks. As LLMs continue to evolve, it is increasingly important to bring their capabilities directly to edge devices [69]. The integration of LLMs into edge devices not only broadens their accessibility but also ensures robust, customized experiences tailored to individual and industrial needs. This trend is gaining traction not only in academia [12, 25, 90, 91, 94], but also in the industry, where leading companies like Intel [67], NVIDIA [2], Microsoft [77], and Qualcomm [70] are actively exploring similar solutions.

However, implementing LLMs on edge devices presents challenges, particularly with managing key-value (KV) caches [65], which play a critical role to enhance LLM token generation speed. This mechanism involves storing previously computed *Key and Value* vectors (KV vectors) during attention calculations and reusing them for generating subsequent tokens. By doing so, it avoids recalculating vectors for earlier tokens with each new token generation. On the other hand, the KV caching incurs a significant memory footprint that grows rapidly as both the model size and the length of generated text increase [32, 99]. For example, when LLaMA 2-7B processes a sequence with the length of 8192 in FP16, the KV cache consumes 4GB of memory, causing the total execution latency to be primarily limited by frequent memory access between on-chip SRAM and off-chip DRAM [59, 90]. This becomes particularly problematic in resource-constrained systems with limited on-chip SRAM capacity, such as edge devices [99]. For example, the Jetson Orin NX edge GPU has only 4MB L3 cache [2].

A simple way to address this issue is by increasing the on-chip SRAM size, which effectively reduces costly off-chip memory access and enhances overall system performance [15, 76]. However, edge devices have limited area and power budgets, and expanding SRAM reduces the resources available for other critical components, such as computational cores [14, 39, 79]. Alternatively, this study explores the use of embedded DRAM (eDRAM) as the primary on-chip storage medium for KV vectors during LLM execution. With fewer transistors per memory cell, such as 3T for eDRAM cells compared to 6T for SRAM cells, eDRAM provides a higher data storage density, resulting in more than twice the capacity [15, 28]. This increased storage density enables a greater on-chip storage capacity within the same chip area. Moreover, eDRAM also offers much lower leakage power than SRAM (around 3.5 \times according to prior work [17]). These benefits make eDRAM an attractive option for storing KV vectors in edge devices.

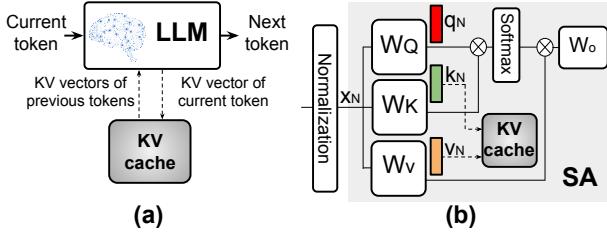


Figure 1: (a) LLM token generation. (b) KV cache for intermediate data storage, where N denotes the token index.

However, a key drawback of eDRAM is the need for periodic refreshes to prevent data loss from leakage. Specifically, refreshing eDRAM cells requires a read-write operation, increasing latency and power consumption, which can significantly impact the efficient deployment of LLMs. To address the challenges of integrating eDRAM, we co-design the KV caching algorithm with the eDRAM-based hardware system, enabling a highly efficient KV cache implementation without compromising accuracy. Our contributions are summarized:

- We propose *Kelle*, an algorithm-system co-design solution for in-device LLM serving on eDRAM-based edge systems. To optimize eDRAM integration cost and improve LLM execution efficiency, we introduce *attention-based eviction and recomputation policy* (AERP) and *two-dimensional adaptive refresh policy* (2DRP) for efficient KV cache implementation (Section 4.1 and 4.2).
- We design a *Kelle accelerator* that utilizes eDRAM as the primary on-chip storage, featuring a customized memory layout. To maximize efficiency, the accelerator integrates a dedicated eDRAM controller and a *systolic evictor* for efficient AERP and 2DRP implementation (Section 5).
- We also introduce the Kelle scheduler (Section 6), which adopts an efficient computation pattern to optimize eDRAM data lifetime and LLM serving latency, significantly reducing both eDRAM refresh energy and memory traffic.
- The evaluation results show that Kelle achieves a 3.9× speedup and 4.5× energy savings compared to other baseline hardware platforms, while maintaining a negligible impact on LLM accuracy (Section 7, 8).

2 Background and Related Work

2.1 LLM Workflow

Modern LLMs (e.g., Llama series [74, 75], GPT series [11, 65]) are constructed as a stack of transformer decoders, with each decoder comprising two fundamental components: a Self-Attention (SA) block and a feed forward network (FFN). During the LLM serving process, the input of the SA block is first multiplied with three weight matrices \$W_Q\$, \$W_K\$, and \$W_V\$, yielding the outputs termed query (\$q\$), key (\$k\$), and value (\$v\$), respectively. The resulting \$q\$ and \$k\$, in combination with \$v\$, will then undergo multiplication, softmax, and residual addition to generate the SA output. The output from the SA will then be passed to the FFN for further processing, which typically involves a standard MLP [63, 65] or gated MLP [45, 74, 75].

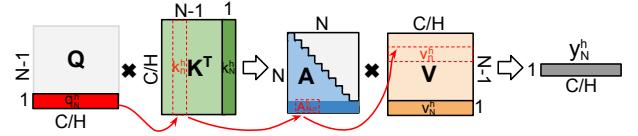


Figure 2: An example on KV vector computation.

The FFN consists of multiple fully connected (FC) layers along with an intermediate activation function, such as GeLU [31].

LLM serving involves two main stages: pre-filling and decoding. In the pre-filling stage, the model processes the context tokens in parallel. During the decoding stage, the model predicts the next token based on the current and previous tokens. This is done by combining the current input with information from previous tokens, expressed in terms of their Key and Value (KV) vectors. This process is repeated in an auto-regressive manner (Figure 1 (a)).

2.2 KV Caching

During the decoding stage, the KV vectors of each newly generated token are stored in a KV cache to enhance generation speed, as shown in Figure 1 (b). By doing so, it avoids recalculating vectors for earlier tokens with each new token generation. Specifically, to produce the output of an LLM block, the input vector of Nth token, with a length \$C\$, is multiplied by \$W_Q\$, \$W_K\$, and \$W_V\$ to generate three vectors: query \$q_N\$, key \$k_N\$, and value \$v_N\$, each with dimensions \$1 \times C\$, where \$C\$ denotes the channel size, as shown in Figure 1 (b). \$q_N\$ together with other KV vectors are then split along the channel dimension into multiple parts, each has a dimension of \$1 \times \frac{C}{H}\$, where \$H\$ denotes number of heads. The resulting vectors for head \$h\$ are denoted as \$q_N^h, k_N^h, v_N^h\$, respectively. The KV vectors from the previous \$N - 1\$ tokens are then loaded from memory. For each head \$h\$, the dot products are then performed between \$q_N^h\$ and each of the key vectors \$k_{n,1}^h, 1 \leq n \leq N\$, and the result is passed through a softmax function, yielding a vector of attention scores, denoted as \$A_{N,n}^h\$, with dimensions of \$1 \times N\$. Next, the attention score vector is used to compute a dot product with each of the value vectors \$v_n^h, 1 \leq n \leq N\$, producing a result vector \$y_N^h\$ of length \$\frac{C}{H}\$. \$y_N^h\$ will then be concatenated across multiple heads, the resulting vector \$y_N\$ which is further multiplied with \$W_O\$. The process is illustrated in Figure 2 and can be represented by the following equations:

$$A_N^h = \text{softmax}([q_N^{h\top} k_1^h, q_N^{h\top} k_2^h, \dots, q_N^{h\top} k_N^h]) \quad (1)$$

$$y_N^h = \sum_{1 \leq n \leq N} (A_{N,n}^h \cdot v_n^h) \quad (2)$$

where \$A_{N,n}^h\$ represents the \$n\$-th element of \$A_N^h\$. As per Equation 1 and Equation 2, we notice that the relative order of KV vector pairs \$[k_n^h, v_n^h]\$ does not affect the decoding computation. In other words, if we swap the values of two pairs of KV vectors (e.g., swap \$[k_1^h, v_1^h]\$ with \$[k_2^h, v_2^h]\$), the result \$y_N^h\$ produced using Equation 1 and 2 remains unchanged.

KV cache compression techniques can be broadly classified into two approaches: token dropping [27, 44, 47, 83, 88, 98] and KV cache quantization [32, 48, 82]. The token dropping strategy identifies

Table 1: Comparison of SRAM and eDRAM.

65nm 4MB	Area	Access Latency	Access Energy	Leakage Power	Refresh Energy	Retention Time
SRAM	7.3 mm^2	2.6ns	185.9 pJ/byte	415mW	NA	NA
eDRAM	3.2 mm^2	1.9ns	84.8 pJ/byte	154mW	1.14 mJ	45 μs [38]

and permanently discards unimportant tokens, making them inaccessible thereafter. StreamLLM [83] identifies *sink tokens*, which are tokens at the start of a sequence that are critical for LLM performance, and preserves recent tokens to maintain performance. H2O [98] identifies *heavy hitter tokens* that have high accumulated attention scores. KIVI [49] groups the KV vectors channel-wise to achieve 2-bit asymmetric quantization. QuaRot [6] utilizes zero-shot Hadamard transformation to reduce the outliers in the model and enables 4-bit quantization. Speculative decoding [33, 34, 42, 51] present another inference technique that accelerates LLMs by using a lightweight draft model to propose multiple tokens, which are then selectively verified by the full model. Kelle can be adopted in orthogonal with the speculative decoding techniques.

2.3 Embedded DRAM

Various eDRAM circuit designs [28, 89] have emerged as alternatives to SRAM, with some requiring only two transistors. Among these, 3T-eDRAM stands out, offering over twice the density and reducing static power dissipation by $3.5\times$ [13, 17] compared to SRAM. Table 1 presents a comparison between 3T-eDRAM and SRAM, with results simulated using Destiny [60] on a 65nm technology node. The advantages of eDRAM, including higher storage density, lower access latency and energy, make it an attractive choice for LLM implementation. Although eDRAM provides several advantages, a serious drawback is the need for periodic refreshes to prevent data corruption caused by charge leakage. As a result, eDRAM is best suited for storing **large amounts of transient data**, where frequent refreshes can be avoided.

Research has explored using eDRAM in accelerator systems to facilitate CNN computation [15, 54, 76, 97], with proposed methods to mitigate refresh power overhead. DaDianNao [15] partitions eDRAM into banks to mitigate refresh failures but does not address the challenges of refresh energy consumption or data retention. RANA [76] injects the bit retention errors during the training process of CNN to mitigate the accuracy drop caused by low refresh frequency. CAMEL [97] optimizes CNN model architecture to shorten the data lifetime during training. While prior research has shown the efficiency of eDRAM in Convolutional Neural Network (CNN) inference and training, its potential has yet to be explored for LLMs. In contrast, Kelle focuses on minimizing the off-chip memory access of KV cache in LLMs using eDRAM, an area previously unexplored.

2.4 Edge LLM Accelerator

To enable deployment of LLMs on edge devices, several studies have proposed methods to improve the accuracy of quantized transformers [23, 30, 40, 46, 50, 57, 85, 92, 100]. Tender [40] suggests a hardware-efficient LLM quantization method by making the scale factor a power of two. COMET [46] designs efficient mixed-precision GPU kernels for 4-bit LLM quantization. Other works, such as FlexGen [68], InfiniGen [41], InstInfer [56], LLM.npu [86],

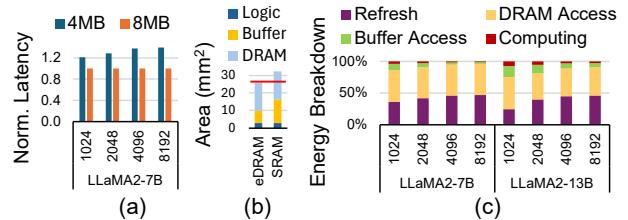


Figure 3: (a) Normalized latency of edge systems with 4MB vs. 8MB SRAM across models and sequence lengths. (b) Area breakdown of the edge systems with 8MB eDRAM and 8MB SRAM. Red line is the area budget. (c) Energy breakdown of the edge system integrating eDRAM. The decoding lengths are shown with a prefilling length of 512. An 8MB eDRAM is used to store KV cache for a subset of layers during decoding. The reported DRAM energy accounts for both model weight access and KV cache offloading from eDRAM.

explore the model offloading strategy between on-chip units and main storage for efficient LLM deployment in resource constraint devices. Cambricon-LLM [90] proposes a chiplet-based hybrid architecture with NPU and a dedicated NAND flash chip to enable efficient on-device inference.

3 Why Use eDRAM for LLMs on Edge Devices?

3.1 Benefits and Challenges of Expanding On-Chip Memory

As shown by earlier research [90, 93, 99], the speed of serving LLMs is significantly constrained by the bandwidth of off-chip memory. In particular, accessing the KV cache poses the most critical bottleneck during the decoding stage of LLMs [41, 98, 99]. A straightforward approach to minimize off-chip memory usage is to expand the on-chip SRAM size, which decreases expensive off-chip memory accesses and boosts system performance [15, 76]. To illustrate this, we evaluate the latencies of two edge computational systems with 4MB and 8MB of SRAM executing the LLaMA2-7B models across different sequence lengths. Tests are conducted on a simulated platform with a 32×32 systolic array for 8-bit MAC operations, and 16GB DRAM with 64GB/s bandwidth, reflective of an edge tensor processing unit (TPU) similar to the Google Coral edge device [72]. As shown in Figure 3 (a), doubling the SRAM size leads to an average of $1.27\times$ speedup. However, expanding the SRAM capacity from 4MB to 8MB in the evaluation platform increases the power consumption and chip area by 29% and 26%, respectively. Given the limited area and power budget for edge environment, increasing SRAM size reduces the resource available for other critical components, leading to a suboptimal system performance [14, 39, 79]. Therefore, we have the following observation:

Obs. 1: Larger on-chip memory alleviates the KV caching bottleneck in LLM, but brings area and power penalties in edge devices using SRAM as the on-chip storage.

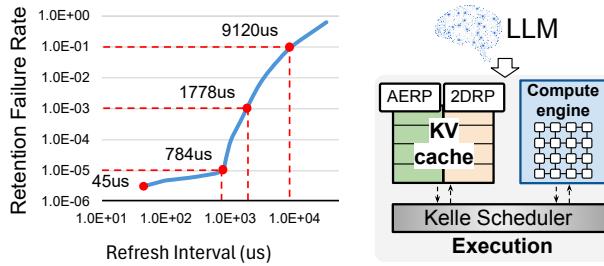


Figure 4: 65nm eDRAM retention failure distribution at 105°C [38]. **Figure 5:** Overview of the Kelle accelerator.

3.2 Pros and Cons of Integrating eDRAM

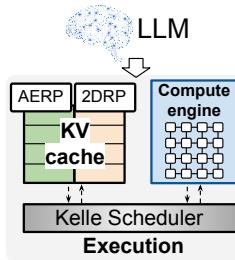
To increase the on-chip memory size without increasing area, one approach is to replace SRAM with eDRAM. eDRAM not only provides more than twice the capacity under the same area of SRAM, but also consumes lower access and leakage energy according to Table 1. As shown in Figure 3 (b), the evaluation system with 8MB eDRAM takes less area than the system with 8MB SRAM, leading to lower LLM serving latency within a smaller chip size. Extensive research [5, 16, 84] and commercial products [22, 24, 28, 37, 80] demonstrate the feasibility of integrating eDRAM as the primary on-chip storage medium. However, its potential to benefit LLM serving in edge devices has not been explored. Although eDRAM offers several advantages, prior studies [76, 97] have shown that refresh operations can become a significant bottleneck in overall system energy consumption. Moreover, when eDRAM is used to store data with longer lifespans, infrequent refreshes can elevate the risk of readout errors, as illustrated in Figure 4. The retention failure rates are represented as the percentage of bits with retention errors, as the refresh interval varies. To illustrate this issue, we use an 8MB eDRAM to replace the 4MB SRAM in the system described in Section 3.1. The eDRAM refresh interval is set to 45 μ s to ensure no data corruption. We evaluate the energy consumption of the eDRAM system across different models and sequence lengths. As shown in Figure 3 (c), without optimization, eDRAM refresh operations take up to 46% of the total energy consumption, leading to 1.7 \times more energy consumption on average.

Obs. 2: Under the same chip area, eDRAM can bring latency benefits over SRAM for LLM serving in edge devices. However, to fully leverage its power advantages, eDRAM refresh operations must be greatly minimized.

3.3 Kelle: Co-design KV Caching and eDRAM

To minimize eDRAM energy consumption, three effective strategies are: reducing data refresh frequency, decreasing stored data size, and decreasing data lifetime. To enable eDRAM for enhanced LLM serving performance in edge devices, we propose *Kelle*, a hardware and algorithm co-design solution for minimized eDRAM refresh energy and efficient KV cache management.

3.3.1 eDRAM Refresh Control. Reducing data refresh frequency may raise the risk of retention failures, causing data corruption. This leads to a key question: *To what extent can LLMs tolerate data*



corruption in the KV cache without compromising accuracy? Led by this question, we co-design the eDRAM memory layout and controller with *two-dimensional adaptive refresh policy* (2DRP), which sets fine-grained dynamic refresh intervals described in Section 4.2.

3.3.2 KV Cache Eviction. A smaller KV cache can significantly reduce the data storage demands on eDRAM, resulting in lower refresh energy consumption and improved system performance. Previous studies have observed that evicting unimportant tokens does not compromise the generation quality. However, to identify the unimportant tokens, previous works either require profiling of the sequences [27, 47], or extra computation [83, 98]. To manage the KV cache efficiently, we propose a novel *systolic evictor* architecture to accelerate the operation of *attention-based eviction and recomputation policy* (AERP), as described in Section 4.1.

3.3.3 KV Vector Recomputation. As the sequence length grows, the benefit of KV caching diminishes at a certain threshold since the time for accessing off-chip memory might outweigh that for recomputing partial KV tensors. Recomputation aligns well with the strength of eDRAM to store transient data, as shown in Section 4.1. However, the balance between recomputation and storage requires careful scheduling considering the hardware features. We propose the *Kelle Scheduler* to reduce the KV vector data lifetime via designing the computational pattern, which is depicted in Section 6.

4 Kelle Algorithm

In this section, we present the efficient algorithms used within the Kelle framework, with an overview provided in Figure 5. During execution, Kelle utilizes *attention-based eviction and recomputation policy* (AERP) and the *two-dimensional adaptive refresh policy* (2DRP) to manage eDRAM operation, as described in Section 4.1 and Section 4.2.

4.1 Attention-based Eviction and Recomputation Policy

We begin by discussing the eviction policy when the eDRAM capacity is reached during the decoding stage.

4.1.1 Eviction policy. For a KV cache with limited capacity, capable of holding up to N' tokens, during the decoding stage, the arrival of the $(N'+1)$ -th token requires the eviction of the KV vectors $[k_n^h, v_n^h]$ from one of the tokens n (where $1 \leq n \leq N'$). The KV vectors of the h -th head and n -th token to be evicted are selected based on their importance s_n^h , which is computed by summing the attention scores (Equation 1) with all other tokens in KV cache, shown as:

$$s_n^h = \sum_{1 \leq i \leq n} A_{n,i}^h \quad (3)$$

An example of the eviction process is illustrated in Figure 6. Assume the KV cache has a budget to store a total of $N' = 4$ vectors. We consider a case with three attention heads. For clarity, we only depict the computation for the first head and omit the head notation. When $[k_5, v_5]$ arrives, the importance scores are first computed using Equation 3, as shown in Figure 6 (a). The KV vectors of the token with the smallest importance score (third token) are then evicted, as depicted in Figure 6 (b). By leveraging the fact that the computation of y_N is unaffected by the relative order of the KV

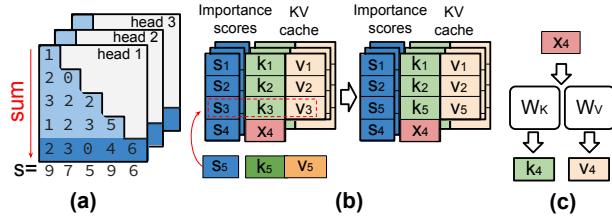


Figure 6: (a) Computation of the importance scores for each of three heads. (b) The KV vectors of the token with the lowest score is replaced with the new KV vectors. Input vector x_4 is stored because the fourth token is important among two out of three heads. Storing x_4 frees up an eDRAM entry, thereby reducing the eDRAM refresh cost. (c) Recompute KV vectors of the fourth token for saving eDRAM storage.

vectors, Equation 1 and Equation 2 can be computed by reading the KV vectors from the cache in sequence, without concern for their original token indices. It is important to note that the importance score s_n^h of the same token n might vary across different attention heads. As a result, the eviction pattern of KV vectors will differ across these heads h .

For the pre-filling stage with a context token length of N_{ctx} , all the context tokens are processed in parallel. For each head within each layer, importance scores of N th token are calculated as $s_N^h = \sum_{1 \leq n \leq N_{ctx}} A_{n,N}^h$. The tokens with top N' highest s_n^h will be retained in the KV cache for decoding operations.

In addition to the tokens with the highest s_n^h scores, the initial tokens and most recent tokens are also retained due to their proven impact on model performance, as demonstrated by prior work [83, 98] and supported by our experiments.

4.1.2 Recomputation policy. As discussed in Section 2.3, eDRAM is well-suited for storing transient data. Although the eviction policy reduces the number of KV vectors that need to be retained during model execution, storing KV vectors with long lifetime in eDRAM still incurs a serious cost due to the required refresh operations. To mitigate the refresh cost, we can further apply the recomputation technique. Specifically, for a subset of tokens N_{recomp} in the KV cache, their KV vectors will be recomputed using the corresponding input vector x_N , which serve as inputs to W_Q , W_K , and W_V , as depicted in Figure 1 (b). By leveraging recomputation, the storage requirement can be reduced from maintaining two vectors (K and V) to holding just a single vector (input x). This approach allows for K and V to be recomputed as needed, effectively mitigating the long data lifetime of KV vectors.

During decoding stage execution, the KV vectors k_N^h and v_N^h for all heads $h \in H$ are first recomputed by multiplying the input vector x_N with W_K and W_V (Figure 6 (c)), which are then used for the decoding process. To achieve savings in KV cache storage through recomputation, the storage cost of the input vector x_N , which is $1 \times C$, must be smaller than that of the recomputed KV vectors. To satisfy this, the KV vectors for a token N are recomputed using x_N if they would otherwise be retained in at least $\theta > 50\%$ of the heads without recomputation, where θ represents the **popularity** of the token. This approach is justified because the storage cost

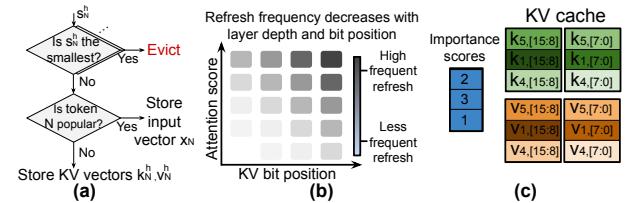


Figure 7: (a) Summary of AERP, only one head h is shown for simplicity. (b) 2D-adaptive refresh policy. (c) As an example on 2DRP. $k_5[15 : 8]$ denotes the eight to fifteenth bits of the key vector for the fifth token. Darker colors mean bits refreshed more frequently, resulting in a lower retention error rate.

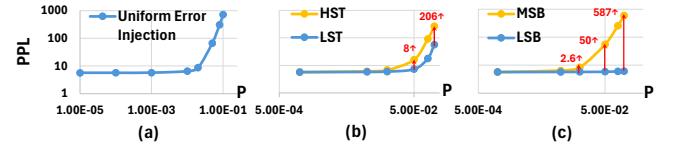


Figure 8: (a) PPLs with bit-flip error rates P . (b) LLM accuracy under varying bit-flip error rates when applying the bit flipping solely on (a) HST vs. LST and (b) MSB vs. LSB, where P denotes the error rate. A lower PPL reflects better performance, with the red numbers representing the gap between the PPL values.

associated with the KV vectors, calculated as $2 \times \frac{C}{H} \times \theta H$, would exceed the size of x_N (i.e., C). As shown in Figure 6 (b), the fourth token is deemed popular in two of three heads, so the input vector x_4 is retained, avoiding the storage of KV vectors.

In addition to storage savings, the recomputed KV vectors will be transient, as they are only used for a short duration to compute Equation 1, which further capitalizes on the advantages of eDRAM. Moreover, the additional cost of recomputation will be minimal due to the systolic array architecture for the computational engine, discussed in Section 5.2.

For the pre-filling stage, the importance scores s_n^h for each token n in head h are calculated first. Next, for each head h , KV vectors are evicted based on the importance scores of the corresponding tokens. Among tokens with high importance scores, those whose KV vectors are retained in at least 50% of the heads (i.e., popular tokens) have their input vectors x_n stored; otherwise, the KV vectors are stored. During decoding, each new token's storage format is dynamically determined by computing the popularity θ . Figure 7 (a) summarizes the overall AERP scheme. Although token popularity can vary during the decoding process, empirical evidence shows limited fluctuation, namely tokens important to over 50% of heads rarely decrease in importance. Therefore, in Kelle, once a token is stored with its input vectors, its storage format remains fixed throughout decoding unless it gets evicted.

4.2 Two-Dimensional Adaptive Refresh Policy

To explore the tolerance of LLMs to data corruption in the KV cache without compromising accuracy, we simulate retention failures by

introducing bit flip errors across the eDRAM memory cells. Specifically, we assess the impact on the perplexity (PPL) of LLaMA2-7B models using the WikiText-2 [52] dataset. Lower PPL indicates better performance. During execution, the bit flip errors are introduced in the KV cache with a uniform probability. The results, presented in Figure 8 (a), reveal that for error rates below 10^{-3} , the increase in PPL remains minimal, staying under 0.1. However, as the bit flip error continues to rise, PPL increases significantly. This suggests that LLMs can tolerate a certain degree of KV cache errors. A natural follow-up question arises: *Is it possible to develop a finer-grained refresh policy that could support even lower refresh frequencies while maintaining accuracy?*

In Section 4.1, tokens are evicted based on their importance scores, as defined in Equation 3. We hypothesize that a similar approach could be applied to the eDRAM refresh policy, assigning lower refresh frequencies to KV vectors or input vectors of less important tokens and higher frequency to those of more important tokens. To test this hypothesis, we implement an adaptive refresh policy and repeated the experiment. Tokens were divided into two groups based on their importance scores, referred to as the high score token (HST) group and the low score token (LST) group for simplicity. A probability p of bit retention failure (bit flip error) in the KV vectors was applied separately to the KV vectors of the corresponding tokens in HST and LST groups. The results in Figure 8 (b) show that LLM performance degrades more when bit retention failures affect the HST group than the LST group, indicating that tokens in the HST group require higher refresh frequency, thus supporting our hypothesis.

Additionally, it is reasonable to hypothesize that the less significant bits (LSBs) are less vulnerable to retention failure errors than the more significant bits (MSBs), as bit flip error on LSBs causes smaller changes in value. For each value in a KV vector, we introduce bit retention errors to either the MSBs (bits 15-8) or the LSBs (bits 7-0). The results, shown in Figure 8 (c), reveal that the MSBs are more sensitive to retention errors than the LSBs under the same bit flip error rate, further supporting our hypothesis.

Based on observations above, we propose an adaptive refresh control strategy called the *two-dimensional adaptive refresh policy* (2DRP), as shown in Figure 7 (b). This strategy adjusts the refresh frequency of each eDRAM cell based on both the bit position within each value in KV vectors or input vectors and the importance score of each token. An example of 2DRP is shown in Figure 7 (c), where the KV cache holds up to $N' = 3$ tokens. The refresh frequency increases with both the token importance and the significance of bit positions. During execution, the importance scores of KV and input vectors are dynamically calculated, and a refresh frequency is assigned accordingly based on these scores and bit positions.

5 Kelle Edge Accelerator

Figure 9 provides an overview of the Kelle accelerator. It incorporates a hybrid eDRAM-SRAM memory subsystem, a reconfigurable systolic array (RSA), and specialized function units (SFUs). The weights are quantized to 8 bits, and activations and KV vectors are maintained in 16 bits, with weights stored in SRAM, while activations and the KV vectors are held in eDRAM. During operation, systolic evictor accumulates attention scores and eDRAM

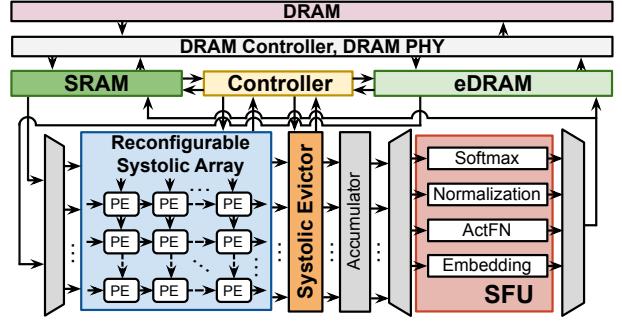


Figure 9: An overview of Kelle hardware accelerator.

controller handles KV vector eviction and recomputation while dynamically adjusting the refresh frequency, as discussed in Sections 4.1 and 4.2. Each processing element (PE) in RSA performs 8-bit multiply-accumulate (MAC) operations.

SFU handles non-linear operations, including activation functions, softmax, normalization, and positional embeddings. As prior research has shown [20, 62, 78, 81], the energy consumption of non-linear operations increases with input sequence length. Among these operations, softmax consumes significant resources. We employ online max calculation from Softmax [71] to minimize memory access. For other non-linear operations, we follow the computation flow and use lookup tables (LUTs) to perform the calculation.

5.1 Memory Subsystem

Figure 10 illustrates the memory subsystem of the Kelle accelerator. In this design, a 2MB SRAM stores the weights, while activations and KV vectors are held in a 256KB *activation eDRAM* and 4MB *KV cache eDRAM*, respectively. Kelle accelerator implements 2DRP by dividing KV vectors into four groups based on importance scores and bit positions and applies the refresh frequency accordingly. Specifically, the MSBs (bits 15-8) of the KV vectors for tokens in the HST group are refreshed at the highest, while LSBs (bits 7-0) of the KV vectors for tokens in the LST group are refreshed at the lowest frequency. To support the AERP, for certain tokens, input vectors are stored in the KV cache eDRAM instead of KV vectors. These input vectors are then divided into four groups and controlled in the same manner as KV vectors, organized by importance scores and bitwidth. For simplicity, we use KV vectors to describe the memory subsystem design, without referencing input vectors.

To execute 2DRP during LLM inference, each element of the KV vectors is split bitwise and stored across different eDRAM banks. Specifically, for KV vectors, the MSBs and LSBs are stored in separate KV cache eDRAM banks, referred to as *MSB banks* and *LSB banks*, highlighted by the darker and lighter colors in Figure 10. The importance score of each token is computed dynamically using Equation 3 under 4-bit precision and stored in a register file, with each entry corresponding to a KV vector spanning four banks. KV vectors corresponding to the same token share the same address across different eDRAM banks. The system features a single eviction controller that manages AERP across all four banks, along with two refresh controllers responsible for executing 2DRP separately over MSB and LSB banks, respectively.

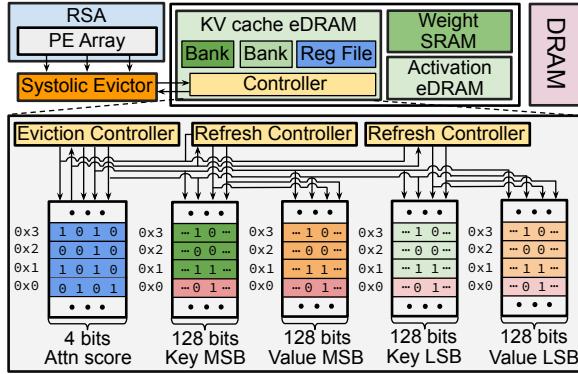


Figure 10: Kelle memory subsystem. Input vectors of certain tokens are stored in the KV cache, represented by red rows.

In each MSB and LSB bank, the tokens are further divided into two groups according to their attention scores, with a counter in the refresh controller monitoring each group’s refresh interval. The controller iterates through the eDRAM entries, identifying the group of each token by reading its attention score from the register file. When the refresh interval for a specific group expires, the controller triggers a *refresh* signal. The corresponding KV vector addresses for the tokens in that group are then computed, and the KV vectors are read out and written back as part of the refresh procedure. The refresh operation is triggered when the KV vectors are not used by the model, so the refresh latency can be hidden. When the KV cache reaches capacity and a new token arrives, the eviction controller receives the evict token index from the systolic evictor and replaces that token with the new token.

To fully utilize the 32×32 RSA by feeding data in parallel and avoiding bank conflicts, the Kelle KV cache is divided into 32 banks. Specifically, 8 banks are assigned to each of the Key MSB, Key LSB, Value MSB, and Value LSB groups. With this design and pipelined cache read, Kelle eDRAM provides sufficient bandwidth to make full use of the RSA without bank conflicts. Additionally, other eDRAM access operations such as token read and token eviction operate independently, this effectively mitigates bank conflicts.

During the LLM execution, the RSA I/O controller efficiently reconstructs the data from different banks for computation with minimal overhead. Additionally, the Kelle accelerator stores the KV vectors from a subset of LLM layers in eDRAM, with the number of layers determined by the specific LLM size and text length. eDRAM greatly minimizes off-chip memory access overhead.

5.2 Reconfigurable Systolic Array

The systolic array core consists of a 32×32 two-dimensional array that processes inputs in a staggered manner, sending the computed partial sums to the accumulator and SFUs. It utilizes a weight-stationary data flow, as shown in Figure 11 (a). We employ a reconfigurable strategy similar to that in FAST [96] to perform in-place transposed matrix multiplication.

Importantly, the recomputation in Section 4.1 introduces minimal overhead in the LLM decoding stage. Leveraging the strength of systolic arrays for matrix operations, the recomputed token vectors

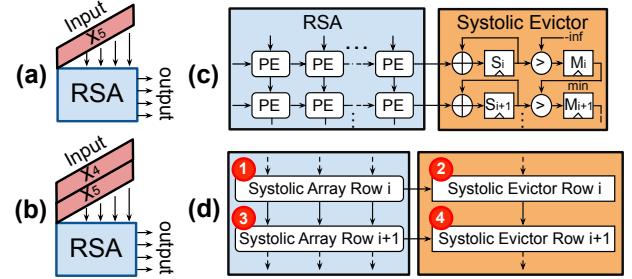


Figure 11: (a) and (b) show the impact of recomputation on RSA operation. (c) The integration of RSA and systolic evictor. (d) The execution order of systolic array and systolic evictor. Numbers in red circles denote the orders.

can be combined with the current token’s input vector to create an input matrix efficiently. Using the same notation as the example in Figure 6 (b), Figure 11 (a) shows the input vector x_5 of the current token being sent to RSA for KV vector computation. To recompute KV vectors for the fourth token, x_4 and x_5 can be combined into a matrix, causing minimal latency and energy growth in Figure 11(b).

5.3 Systolic Evictor

The token eviction process in the AERP algorithm includes computing the attention score as described in Equation 1, updating the importance score based on Equation 3, identifying the token with the lowest importance score, and performing the KV cache update.

To efficiently implement the eviction algorithm, we propose a systolic evictor (SE) which operates in a systolic style and is integrated with the RSA to search the minimum importance score on-the-fly. The importance score is calculated by summing the QK^T results in Equation 1 without passing through the softmax. This integration ensures the token with the minimum importance score is found as soon as the new token’s attention score is calculated from the RSA. After finding the index of the token with minimum importance score, the SE sends the index to the eviction controller in the eDRAM controller to evict the corresponding token. Figure 11 (c) illustrates the design of SE and its integration in the RSA. The SE comprises a column of registers, denoted as S in Figure 11 (c), to preload the importance scores of previous tokens, and a register chain, denoted as M , periodically propagates the minimum importance score (min) from top to bottom. Figure 11 (d) illustrates execution order of the RSA and SE. In one cycle, the attention score is calculated from the i -th row of RSA, and then the i -th row of SE updates the importance score and the minimum importance score index, marked as Step 1 and Step 2, respectively. In the next cycle, the same operations are executed in the next row of RSA and SE, marked as Step 3 and Step 4, respectively. The systolic evictor avoids the extra LLM execution latency from the minimum search.

6 Kelle Scheduler

To further minimize eDRAM refresh energy, we introduce a novel computation pattern that shortens data lifetime and accelerates LLM inference, all without compromising accuracy.

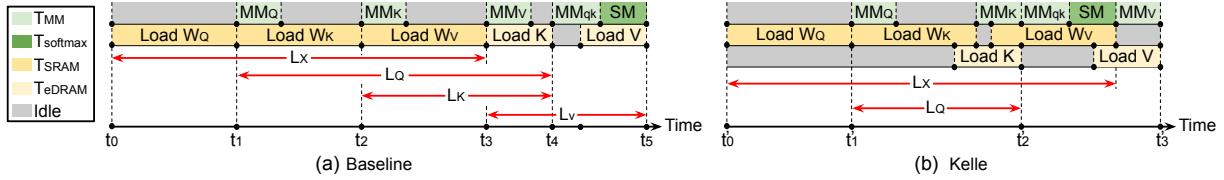


Figure 12: (a) and (b) show the computation patterns and eDRAM data lifetime of the SA block in baseline and Kelle scheduler. SM denotes softmax operation.

To begin, we perform a numerical analysis of the data lifetime associated with the self-attention (SA) architecture in the LLM decoding phase. As illustrated in Section 2.1, the computation in SA firstly involves matrix multiplication between the input X with weight matrices W_Q , W_K , and W_V producing the output Q , K and V . The processes are denoted as MM_Q , MM_K , and MM_V , respectively. Subsequently, Q and K undergo multiplication, followed by a softmax operation to compute the attention score A , which are referred to as MM_{qk} and SM , respectively. Ultimately, A is multiplied by the weight matrix W_O to produce the SA output, labeled as MM_O . The latency (T_{MM}) of matrix multiplications is estimated as follows:

$$T_{MM} = \frac{N_{MM}}{TOP_{RSA}} \quad (4)$$

where N_{MM} represents the number of MAC operations required by the matrix multiplication. TOP_{RSA} denotes the throughput of the RSA, as described in Section 5.2. The latency associated with eDRAM access operations for KV vectors, represented as T_{eDRAM} , is modeled as follows:

$$T_{eDRAM} = \frac{S_{KV}}{B_{eDRAM}} \quad (5)$$

where S_{KV} denotes the size of the KV vectors in bytes. B_{eDRAM} denotes the bandwidth of the eDRAM. Similarly, the latency associated with the weight SRAM access operations, denoted as T_{SRAM} , is modeled as follows:

$$T_{SRAM} = \frac{S_W}{B_{SRAM}} \quad (6)$$

where S_W denotes the size of weights in bytes. B_{SRAM} denotes the bandwidth of SRAM. Figures 12 (a) show a baseline computation pattern, where the matrix multiplication operations MM_Q , MM_K , MM_V , and MM_{qk} are conducted one after another, which prolongs the data lifetime for the inputs X , Q , K , and V . Data lifetime is defined as the interval from when data is computed to when it is utilized by a subsequent operation. For instance, in Figures 12 (a), the computation of vector Q starts at t_1 after the weight matrix W_Q is accessed from SRAM, and Q is consumed at t_4 when the multiplication of Q and K starts. Between t_1 and t_4 , W_K and W_V are loaded from SRAM and K is accessed from eDRAM KV cache. The latencies of accessing W_K and W_V are both T_{SRAM} and the latency of accessing K is T_{eDRAM} . So the data lifetime of Q is $2 \times T_{SRAM} + T_{eDRAM}$. The total data lifetime of all the activations is the sum of the data lifetime of each activation because they are all stored in eDRAM and require refreshing. We omit the computation time T_{MM} from Equation 4 due to its negligible magnitude relative to T_{SRAM} and T_{eDRAM} . This extended data lifetime leads to a higher refresh cost for eDRAM. The total data lifetime L_{bl} of the transient data in

baseline schedule is modeled as follows:

$$\begin{aligned} L_X &= 3 \times T_{SRAM}, L_Q = 2 \times T_{SRAM} + T_{eDRAM} \\ L_K &= T_{SRAM} + T_{eDRAM}, L_V = 2T_{eDRAM} \\ L_{bl} &= L_X + L_Q + L_K + L_V = 6T_{SRAM} + 4T_{eDRAM} \end{aligned} \quad (7)$$

where L_X , L_Q , L_K and L_V denotes the data lifetime of X , Q , K and V , respectively. T_{SRAM} and T_{eDRAM} are defined in Equation 6 and Equation 5. In contrast, the computation pattern used by the Kelle is illustrated in Figures 12 (b). Thanks to the integration of separate on-chip memory, the memory access for weights and KV vectors is parallelized. This arrangement reduces the data lifetime of activations, which can be estimated as follows:

$$\begin{aligned} L_X &= 3 \times T_{SRAM}, L_Q = T_{SRAM} + T_{eDRAM} \\ L_{Kelle} &= L_X + L_Q = 4T_{SRAM} + T_{eDRAM} \end{aligned} \quad (8)$$

The key and value vectors are used immediately for their respective computations, eliminating the need for long-term storage; therefore, their data lifetimes can be considered negligible. Compared to the baseline scheme, the Kelle scheduler significantly reduces the data lifetime of transient data in eDRAM, resulting in decreased refresh energy consumption and enhanced system performance.

7 Accuracy Evaluation

7.1 Main Accuracy Result

Kelle is evaluated on various LLMs, including Llama2 [75], Llama3 [21], Llama3.2 [21], Mistral [35], QWEN [87], and OPT [95] with varying model sizes. We evaluate Kelle on the language generation tasks by perplexity of WikiText-2 (WK2) [52], and PG19 [66]. WK2 sequences range from hundreds to thousands of tokens. PG19 sequences range from tens of thousands to millions. We evaluate the PG19 text generation task using the Cold Compress framework [4, 61] by providing the model a book title and a short description and setting the sequence generation length to 8192. Kelle is also evaluated over different zero-shot tasks, including PIQA (PQ) [10], Lambada (LA) [64], Arc Easy (A-e) [18], Arc Challenge (A-c) [18], TriviaQA (TQ) [36], and Qasper (QP) [19]. We use the LM Evaluation Harness [26] with default parameters.

For KV vector eviction, the number of tokens retained in the KV cache is dynamically adjusted based on the dataset during both pre-filling and decoding phases. To simulate bit flip error from low eDRAM refresh frequency, bit-level retention failure is introduced with a predefined probability based on the refresh interval. We set the refresh interval as 0.36ms, 5.4ms, 1.44ms, and 7.2ms for the MSBs (bits 15-8) of HST, LSBs (bits 7-0) of HST, MSBs of LST, LSBs of LST, respectively, with an average retention time of 1.05ms. This achieves an averaged retention failure rate at 2e-3.

Table 2: Accuracy performance of each method. FP16 denotes the LLM accuracy under FP16 without KV cache reduction.

Model Method	LLaMA2-7B				LLaMA2-13B				LLaMA3.2-3B				LLaMA3-8B		Mistral-7B		QWEN2-7B		OPT-6.7B					
	FP16	SL	H2O	QR	Kelle	FP16	SL	H2O	QR	Kelle	FP16	SL	H2O	Kelle	FP16	Kelle	FP16	H2O	Kelle	FP16	H2O	Kelle		
WK2 (↓)	5.47	6.89	5.70	5.73	5.74	4.88	6.21	5.44	5.83	5.62	6.32	7.93	6.57	6.65	6.14	6.59	5.25	5.86	5.32	6.11	6.23	12.3	13.8	14.6
PG19 (↓)	10.51	NA	12.34	11.77	12.59	8.75	NA	9.84	9.15	9.80	8.95	NA	9.84	9.66	11.82	12.97	10.42	13.30	9.3	11.2	11.4	17.4	20.1	19.8
A-c (↑)	46.33	38.40	46.10	45.80	45.93	49.06	46.08	48.35	47.59	48.83	50.34	46.59	50.08	50.03	53.16	51.89	55.03	54.98	57.6	56.1	55.8	45.2	44.1	44.8
A-e (↑)	74.62	52.99	73.02	72.89	72.78	77.48	58.25	76.71	76.45	76.33	76.98	70.45	76.31	76.45	77.69	75.81	80.22	78.90	70.5	67.8	68.2	60.1	58.8	58.6
PQ (↑)	79.11	75.14	78.05	77.42	77.35	80.52	78.92	79.05	78.78	78.92	79.7	75.78	79.43	78.64	80.52	77.62	82.15	80.81	80.9	78.4	78.5	76.4	75.2	74.9
LA (↑)	73.90	NA	NA	72.39	72.81	76.75	NA	NA	75.67	75.98	73.58	NA	NA	71.26	75.72	73.38	75.49	75.54	68.4	NA	67.1	61.5	NA	59.2
TQ (↑)	48.95	NA	47.53	47.56	47.40	58.73	NA	57.46	56.86	57.55	59.84	NA	58.62	58.70	61.53	58.79	61.57	59.96	63.5	61.1	60.7	43.2	40.3	40.6
QP (↑)	12.69	NA	12.31	12.06	12.18	13.07	NA	11.75	11.63	11.86	12.52	NA	10.87	11.01	13.84	11.88	13.18	11.67	21.3	19.4	19.5	9.3	7.8	7.5

Table 3: LLaMA2-7B accuracies over different cache sizes.

N'	Full	512	256	128	64	32	16
A-c (↑)	46.02	46.02	45.92	45.93	44.63	44.20	38.52
A-e (↑)	73.05	73.04	73.05	72.78	72.38	70.42	67.27
PQ (↑)	78.75	78.49	77.61	77.35	75.31	74.14	71.63

Table 4: LLaMA2-7B accuracies of different refresh intervals.

Uniform (μs)	540	1050	2062			
HST (μs)	180, 3600	360, 5400	720, 9000			
LST (μs)	720, 5400	1440, 7200	2880, 10800			
Accuracy	Uniform	2DRP	Uniform	2DRP	Uniform	2DRP
A-c (↑)	45.31	46.26	44.19	45.93	38.52	39.78
A-e (↑)	72.85	73.12	70.29	72.78	66.50	67.05
PQ (↑)	76.83	77.44	76.43	77.35	74.97	75.21

We compare the accuracy performance of the Kelle algorithm with a state-of-the-art quantization framework, QuaRot (QR) [6]. Additionally, we include StreamLLM [83] and H2O [98], two recent KV cache eviction techniques, for comparison. The model weights are quantized with 8-bit across all the approaches. To keep the same KV cache budget between quantization and KV cache eviction baselines, we configure QuaRot to quantize the KV vectors to 4-bit, while StreamLLM, H2O, and Kelle are left unquantized as 16-bit. For Kelle, we maintain a token storage budget of $N'=128$ for the PQ, LA, A-e and A-c, $N'=512$ for WK2, $N'=1024$ for the TQ and QP, and $N'=2048$ for PG19. Within the token budget, the most recent token window size is configured as 64 for PQ, LA, A-e, and A-c; 256 for WK2; 512 for TQ and QP; and 1024 for PG-19. 10 initial tokens are also preserved across the datasets. StreamLLM and H2O are set to have the same token storage budget as Kelle. We also compare with the original FP16 models without KV cache eviction, denoted as FP16. As shown in Table 2, Kelle maintains a comparable accuracies as the original full KV cache model and outperforms or achieves a comparable performance as the rest methods, showing the superior accuracy performance of AERP and 2DRP algorithms.

7.2 Ablation Study

We adjust the budget size N' for the Llama2-7B model and examine its impact across various tasks. All other settings (e.g., quantization bitwidth, retention failure rate) for Kelle remain the same. From Table 3, we observe a consistent decline in accuracy as the budget N' decreases, but still achieves reasonable performance for $N' \geq 128$, comparing with the KV cache without pruning (Full in Table 3).

Next, we examine the impact of 2DRP on LLM accuracy. Specifically, we compare 2DRP with a condition where all eDRAM cells

Table 5: Kelle Qualitative Metrics

Model	Method	CNN	Truth	BBQ
LLaMA2	FP16	40.58	34.28	95.21
7B	Kelle	38.54	33.26	93.75

Table 6: Accuracy of Kelle with Quantization

Method	Kelle W8A16	Kelle W4A8
WK2 (↓)	5.74	6.51
AC (↑)	45.93	44.89
AE (↑)	72.78	69.96
PQ (↑)	77.35	76.70

share the same refresh interval, while maintaining the same average retention failure rate as 2DRP. All other conditions remain the same for Kelle. In Table 4, we present the accuracy change with varying refresh intervals on different tasks for the Llama2-7B model. In Table 4, Uniform (μs) denotes the uniform refresh interval applied to eDRAM. The two numbers in the HST row represent the refresh intervals for the MSBs and LSBs of HSTs, same for the LST row. We observe that 2DRP improves accuracy compared to the uniform eDRAM refresh across all the conditions and datasets.

Since edge deployment often involves human-facing applications, it is important to evaluate the impact of the approximations in memory behavior introduced by 2DRP on text generation qualitative metrics. To evaluate coherence, we run Kelle on the LLaMA2-7B and Mistral-7B models using the CNN/DailyMail [53] (CNN) summarization dataset and report the ROUGE-1 scores. To evaluate factual correctness, we test Kelle on the TruthfulQA benchmark [43] (Truth) and report the multiple-choice, single-answer accuracy. To assess bias tendencies, we use the BBQ benchmark [58] and report the corresponding bias evaluation scores for both models. The results in Table 5 show that Kelle achieves performance comparable to the FP16 model across all criteria.

Finally, we quantize the Llama2-7B model using the Quartot framework [6] which adopts Hadamard Transformation to enable low bit LLM quantization. We quantize the model weights to 4-bit, KV vectors, and activations to 8-bit. With quantization, Kelle's system performance is expected to improve further, while the impact on accuracy remains minimal, as shown in Table 6. This demonstrates Kelle's compatibility with model quantization techniques.

8 Hardware Evaluation

In this section, we report the hardware evaluation results of Kelle edge accelerator described in Section 5. The Kelle edge accelerator consists of a 2D 32×32 RSA, an SFU, essential interfaces, and a memory controller, all implemented in RTL using SystemVerilog, with frequency set to 1GHz. We report the area and power of Kelle accelerator by synthesizing the components using 45nm NanGate Open Cell Library [1] with Synopsys Design Compiler [9]. The size of SRAM for weight storage is set to 2MB. The size of eDRAM for KV

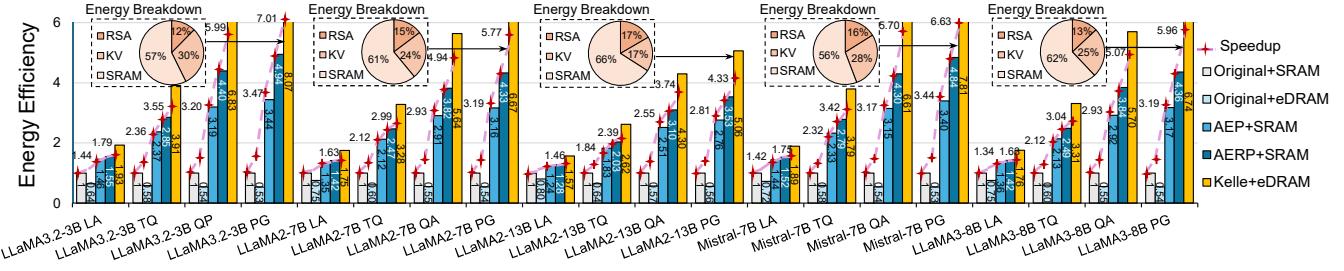


Figure 13: Comparison between Kelle and baseline systems. The performance are evaluated in terms of normalized energy efficiency and speedup. The pie charts show the on-chip energy breakdown of major components within Kelle+eDRAM. The dotted red lines depict the speedup of the corresponding settings.

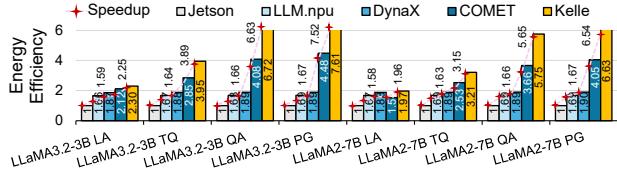


Figure 14: Comparison between LLM accelerators

cache and activation storage is set to 4MB and 256KB respectively. SRAM and eDRAM bandwidths are set to 128GB/s and 256 GB/s, respectively. We utilize Destiny [60] to evaluate the area, power, and timing performance of the eDRAM and SRAM with 65nm tech node at 105°C. The eDRAM retention time distribution aligns with the data shown in Figure 4 [38, 97]. Notably, eDRAM operating at temperatures below 105°C exhibits an even longer retention time, further enhancing system performance. We utilize Cacti 7 [8] to simulate the performance of a 16GB LPDDR4 DRAM, with 64GB/s bandwidth, similar to the DRAM in the Google Coral edge device [72]. With these settings, the total on-chip area is 9.5mm² and the area breakdown of RSA, eDRAM, SRAM, SFU are 23%, 33%, 37%, 7%, respectively. The DRAM takes an area of 16mm². The on-chip power is 6.52W and the power breakdown of RSA, eDRAM, SRAM, SFU are 17%, 29%, 41%, 13%, respectively. The DRAM power is 11.74W. Kelle accelerator achieves 4.13 INT8 TOPs. The Kelle scheduler described in Section 6 further reduces the eDRAM cost.

We assess the hardware performance of the Kelle accelerator across various LLM architectures over multiples tasks including Lambada (LA) [64], TriviaQA (TQ) [36], Qasper (QA) [19], and PG19 [66], with the context length set to 128, 512, 1024, and 512, and the decoding length set to 512, 2048, 5120, and 8192, respectively. The batch size is set to 16. The off-chip DRAM access latency and energy are included in all evaluation results.

8.1 End-to-End Performance Evaluation

8.1.1 Evaluation Baseline. To understand the separate contributions of the Kelle algorithm and eDRAM-based accelerator discussed in Section 5, we compare the Kelle algorithm paired with an eDRAM-based Kelle accelerator, referred to as **Kelle+eDRAM**, against four baseline solutions.

The first baseline, **Original+SRAM**, runs the original LLM on a system using SRAM as the primary on-chip storage. The model

weights are quantized to 8 bits, activations and KV vectors remain 16 bits and processed using Kelle RSA configured for 8-bit MAC operations. The KV cache remains intact, with no AERP applied. The SRAM-based system is configured to match the total on-chip area of Kelle+eDRAM. We adjust the SRAM and systolic size to achieve balanced compute/memory IO ratio, resulting in a systolic array with 24×24 8-bit PEs, 4MB of on-chip SRAM, 16GB of off-chip DRAM. The second baseline, **Original+eDRAM**, involves running the original LLM on an eDRAM-based Kelle accelerator while keeping the KV cache intact. The models are processed using Kelle RSA configured for 8-bit MAC operations. This baseline removes all algorithmic innovations and evaluates only the performance of the eDRAM-based system. In the third baseline, **AEP+SRAM**, we apply the attention-based eviction techniques with the settings described in Section 7.1 for KV cache pruning, implemented on the same SRAM-based system of Original+SRAM. The goal is to evaluate the impact of the cache eviction algorithm on the SRAM-based system. Note that this baseline does not involve any recomputation. The fourth baseline, **AERP+SRAM**, runs the AERP algorithm on the SRAM-based Kelle accelerator.

8.1.2 End-to-End Performance Improvement. Figure 13 compares the above baseline solutions in terms of energy efficiency and processing latency on multiple LLM and datasets. On average, Kelle+eDRAM achieves a 3.94× and 4.46× improvement in latency and energy efficiency compared to the Original+SRAM, and performance gap gets larger as the decoding sequence gets longer. The superior performance of Kelle stems from the algorithmic innovations of AERP and 2DRP, combined with hardware advantages such as the efficient eDRAM memory controller, the systolic evictor design, and the proposed Kelle scheduler.

8.1.3 Individual Contribution on the Performance Improvement. In this section, we investigate the individual impact of Kelle optimization techniques. First, on average, compared to Original+SRAM, Original+eDRAM improves the speedup by 32% but degrades the energy efficiency by 39%. The increased energy consumption is attributed to eDRAM refresh operations without algorithmic or hardware-level optimizations. The eDRAM enhances speedup due to its larger capacity and faster access speeds compared to SRAM. Second, the attention-based eviction policy accelerated by the systolic evictor reduces the latency by 2.39× and improves the energy efficiency by 2.41×, when comparing the AEP+SRAM system with

Table 7: Energy efficiency over multiple KV cache budgets.

N' in PG19	2048	3500	5250	7000	8750
LLaMA3.2-3B	8.07 \times	6.89 \times	5.77 \times	5.13 \times	4.55 \times
LLaMA2-13B	5.06 \times	4.62 \times	4.02 \times	3.46 \times	3.11 \times

the Original+SRAM system. Next, thanks to the attention-based recomputation policy, AERP+SRAM system improves the speedup and energy efficiency by $1.19\times$ and $1.27\times$ to the AEP+SRAM system. Finally, for the system executing models with AERP, the eDRAM optimized by 2DRP and Kelle scheduler provides a $1.29\times$ improvement in speedup and a $1.45\times$ improvement in energy efficiency when comparing Kelle+eDRAM to AERP+SRAM system. Specifically, the 2DRP mechanism greatly reduces refresh energy, enabling the Kelle to take full advantage of eDRAM.

8.1.4 Overhead Analysis. The pie charts in Figure 13 display the energy breakdown for the Kelle+eDRAM system. The reduced energy share for the KV cache highlights how eDRAM coupled with Kelle algorithms alleviates memory access bottlenecks. Thanks to the efficiency of systolic arrays for matrix-matrix multiplication, the hardware overhead from KV recomputation is minimal, with RSA consuming only a small portion of on-chip energy.

To accelerate the token eviction process, we introduce a Systolic Evictor unit, which is a small computation unit coupled with the RSA. It takes an area of $0.06mm^2$ (0.6% of the on-chip area) and consumes power of $0.028W$ (0.4% of the on-chip power). The systolic evictor avoids the stall of the LLM execution for KV cache eviction and redundant memory and computation access. It improves the system’s energy efficiency and reduces the latency by 5% and 7%.

8.2 Comparison with Other Accelerators

We compare Kelle+eDRAM with other cutting-edge LLM accelerators. LLM.npu [86] enhances the on-device Neural Processing Unit (NPU) offloading to reduce pre-filling latency by re-constructing the prompt and model. DynaX [85] proposes dynamic fine-grained structured pruning to enhance the efficiency of sparse attention computation and achieves a 90% attention sparsity. Dynax alleviates the computation bottleneck during the pre-filling stage. COMET [46] quantizes the LLMs to 4-bit and designs high performance GPU kernel to support the mixed-precision computation. As advanced quantization techniques are not the main focus of this paper, we configure COMET to quantize LLM weights to 8-bit and both activations and KV vectors to 4-bit, ensuring a comparable KV cache storage budget to that of Kelle+eDRAM. Finally, we compare Kelle with the NVIDIA Jetson Orin edge GPU [2] implementation of LLM using FP8, measured with pynvml [3] and nvidia-smi [55].

As shown in Figure 14, Kelle+eDRAM achieves better improvements in speedup and energy efficiency over other LLM accelerators. LLM.npu and Dynax optimize the computation-intensive pre-filling stage but do not address the KV cache bottleneck encountered during the LLM decoding stage. The performance gains of Kelle over COMET underscore the limitations of relying solely on KV cache compression without dedicated hardware accelerator support.

8.3 Ablation Study

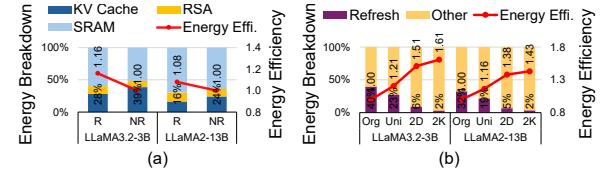


Figure 15: (a) Impact of KV cache recomputation in Kelle+eDRAM. (b) Evaluation on 2DRP and Kelle scheduler.

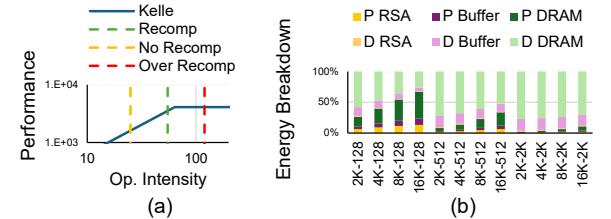


Figure 16: (a) KV Cache Recomputation impact. (b) Evaluation on long input sequences. P and D denote the prefilling and decoding stage, respectively.

8.3.1 Impact of KV Cache Budget. Table 7 illustrates the energy efficiency improvement of Kelle+eDRAM over different KV cache budgets N' . Without eviction, the largest possible number of tokens will be $N' = 8750$ for PG19. Results indicate that even under this condition, Kelle achieves approximately $3\times$ greater energy efficiency over Original+SRAM, highlighting the robustness of Kelle.

8.3.2 Impact of the Recomputation. We compare the energy consumption of Kelle+eDRAM with and without KV cache recomputation. As shown in Figure 15 (a), the recomputation algorithm effectively reduces the energy consumption of KV cache, with minimal increase in RSA energy consumption. Moreover, we profile the popularity change of tokens during the pre-filling and decoding stage across LLM architectures and tasks. On average, over 86% of the popular tokens in the pre-filling stage continue to be popular in the decoding stage, validating the execution strategy outlined in Section 4.1.2.

Recomputation allows Kelle to store more tokens on-chip, reducing DRAM access. When processing the LLaMA2-7B model, accessing one KV vector from DRAM takes approximately $1.1\ \mu s$. In comparison, recomputing a KV vector using the RSA introduces an additional latency of $3.2\ \mu s$. Recomputation helps hide memory stalls by overlapping compute with memory access, reducing overall latency and improving energy efficiency by an average of 25%. For example, loading four KV vectors from DRAM requires $4.4\ \mu s$. With recomputation, three vectors are loaded, and one is recomputed in parallel during the load, reducing the total latency to $3.3\ \mu s$. In terms of energy, the RSA remains active regardless of the number of input vectors, so the incremental energy cost of recomputation is negligible.

Figure 16 (a) presents the roofline model of Kelle under three settings: No Recomp (without recomputation), Recomp (with a moderate recomputation workload), and Over Recomp (with excessive

Table 8: Energy efficiency across refresh intervals.

LLaMA3.2-3B Interval (μ s)	Task	
	TriviaQA	PG19
1050	3.91 \times	8.07 \times
525	3.65 \times	7.31 \times
131	3.06 \times	6.05 \times

recomputation). Recomputation improves performance by increasing the effective memory bandwidth. However, as more KV vectors are recomputed, the RSA becomes a bottleneck. This behavior is reflected in the Over Recomp line, where Kelle transitions from a memory-bound regime to a compute-bound regime.

8.3.3 Impact of 2DRP and Kelle Scheduler. We evaluate Kelle+eDRAM running the Llama2-7B model executing the PG19 task under four strategies. **Org** strategy refreshes the eDRAM at its retention time with a 45μ s interval, ensuring almost no data corruption. **Uni** strategy uses a uniform refresh interval of $0.36ms$, the interval that enables the same LLM accuracy achieved by 2DRP. 2DRP, denoted as **2D** applies varying refresh intervals based on attention scores and bit positions. **2K** strategy combines both 2DRP and the Kelle scheduler. As depicted in Figure 15 (b), the finer-grained refresh policy in 2DRP improves energy efficiency. With both 2DRP and Kelle scheduler, the Kelle achieves the optimal performance.

8.3.4 Impact of eDRAM Retention Time. We assess the impact of eDRAM retention time on Kelle’s performance, considering its influence on the bit failure rate. Retention time is affected by various factors, including design, technology nodes, and temperature [29, 38, 97]. We evaluate Kelle+eDRAM on the TriviaQA and PG19 tasks using 2DRP with different retention times. Specifically, we reduce the Kelle retention time (45μ s) to average refresh intervals of 525μ s, 262μ s, and 131μ s, respectively. Table 8 shows the energy efficiency of these two settings compared to the Original+SRAM system. Thanks to AERP, the KV cache access overhead remains a small fraction of the total energy consumption. Consequently, the energy increase due to the retention time decrease is small, allowing Kelle+eDRAM to maintain performance gains.

8.3.5 Impact of Input Sequence Length. We evaluate the energy consumption of Kelle+eDRAM under long input sequence lengths using the Llama2-7B model on the PG-19 dataset across different input output sequence lengths. We use the format input length - output length (e.g., "16K-128") to denote each experiment setting. As shown in Figure 16 (b), when the input sequence is long and the decoding length is short, the prefilling stage dominates the overall energy consumption and the system becomes compute-bound. In this case, Kelle achieves a moderate energy efficiency improvement of $2.1\times$ over the Original+SRAM baseline. As both the input and output sequence lengths increase, the DRAM access energy for activations grows correspondingly. Under this more memory-intensive scenario, Kelle delivers an average energy efficiency improvement of $5.6\times$ over Original+SRAM and $1.8\times$ over AERP+SRAM, owing to its efficient KV cache management strategies.

8.3.6 Impact of Batch Size. We compare Kelle performance across different batch sizes using the Llama2-7B model on the PG-19

Table 9: Energy efficiency across different Batch Sizes.

Batch size	PG			
	Original +SRAM	AEP +SRAM	AERP +SRAM	Kelle +eDRAM
16	1 \times	3.16 \times	4.33 \times	6.67 \times
4	1 \times	1.71 \times	1.81 \times	2.23 \times
1	1 \times	1.24 \times	1.36 \times	1.71 \times

dataset, as shown in Table 9. While the energy efficiency improvement of Kelle over the Original+SRAM baseline is less significant at smaller batch sizes due to reduced utilization of the RSA and lower data transfer efficiency for model weights, Kelle still consistently outperforms all baselines. At a batch size of 1, Kelle achieves a speedup of 71% over Original+SRAM, 37% over AEP+SRAM, and 25% over AERP+SRAM.

8.3.7 Impact of eDRAM Bandwidth. We conduct experiments to evaluate Kelle under reduced eDRAM bandwidth (128GB/s), achieved by halving the number of banks and doubling the capacity per bank, while keeping the total eDRAM area and capacity constant. Using the Llama2-7B model on PG-19 and TriviaQA, Kelle+eDRAM achieves $1.47\times$ and $1.35\times$ energy gains over AERP+SRAM, and $6.31\times$ and $5.42\times$ over Original+SRAM. Though slightly lower than full-bandwidth Kelle, these results show that increasing eDRAM capacity, even with reduced bandwidth, effectively cuts costly DRAM accesses and improves bandwidth efficiency.

8.4 Discussion

8.4.1 Handle Long-Context Inference. For long context inference, due to limited eDRAM capacity, excess KV data is offloaded to 16 GB DRAM. A simple analysis with Llama 2 7B shows that Kelle can support up to 19,000 input tokens without AERP, assuming 8-bit weights occupy 6.5 GB out of 16 GB DRAM and each token’s 16-bit KV pair across 32 layers. Introducing AERP enables immediate KV cache reduction after each layer’s execution, freeing memory to accommodate the full input sequence in later layers. This allows Kelle to support input sequences of up to around 60K tokens. Additionally, quantizing KV vectors to 4-bit enables support for up to 240K tokens. While an upper limit remains, it exceeds typical LLM input lengths up to tens of thousands of tokens [7, 73, 75].

Although longer input sequences increase overhead, the permutation invariant property of Equations 1 and 2 allows new KV vectors to be placed in the same positions as evicted ones, greatly simplifying the paging process. Additionally, the vectors can be prefetched sequentially without requiring complex lookup mechanisms. As a result, the prefetching overhead increases linearly with the input length, avoiding disproportionate growth.

8.4.2 Integrate Kelle with GPU. While Kelle is implemented with a systolic array, AERP can be adapted to GPUs; however, identifying the token with the lowest attention score may be inefficient due to the lack of a systolic evictor. 2DRP is specific to eDRAM to reduce the refresh energy consumption. The eDRAM can be coupled with the GPU’s existing memory system to store KV vectors. Finally, the Kelle scheduler can be readily implemented using CUDA.

9 Conclusion

The KV caching technique is crucial for enhancing the efficiency of LLMs. However, storing the extensive KV vectors results in a substantial memory footprint and increased data access costs. In this work, we introduce Kelle system that utilizes eDRAM as the primary storage medium for KV vectors. The superior performance of Kelle highlights the significant potential of eDRAM in implementing the KV caching mechanism, paving the way for future research.

References

- [1] [n.d.]. Nangate freepdk45 open cell library. <https://silvaco.com/services/library-design/>
- [2] [n.d.]. NVIDIA Jetson Orin. <https://www.siliconhighwaydirect.com/product-p/900-13767-0000-000.htm>.
- [3] [n.d.]. pynvml: Python Bindings for the NVIDIA Management Library. <https://pypi.org/project/pynvml/.>
- [4] Griffin Adams, Faisal Ladak, Hailey Schoelkopf, and Raja Biswas. 2024. Cold Compress: A Toolkit for Benchmarking KV Cache Compression Approaches. <https://www.answer.ai/posts/2024-08-01-cold-compress.html>
- [5] Aditya Agrawal, Amin Ansari, and Josep Torrellas. 2014. Mosaic: Exploiting the spatial locality of process variation to reduce refresh energy in on-chip eDRAM modules. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 84–95.
- [6] Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. 2024. Quarot: Outlier-free 4-bit inference in rotated llms. *arXiv preprint arXiv:2404.00456* (2024).
- [7] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. 2025. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923* (2025).
- [8] Rajeev Balasubramonian, Andrew B. Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. 2017. CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories. *ACM Trans. Archit. Code Optim.* 14, 2, Article 14 (June 2017), 25 pages. <https://doi.org/10.1145/3085572>
- [9] B. Jayant Baliga. 2019. Synopsys. *Wide Bandgap Semiconductor Power Devices* (2019). <https://api.semanticscholar.org/CorpusID:239327327>
- [10] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2019. PIQA: Reasoning about Physical Commonsense in Natural Language. In *AAAI Conference on Artificial Intelligence*. <https://api.semanticscholar.org/CorpusID:208290939>
- [11] Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).
- [12] Fenglong Cai, Dong Yuan, Zhe Yang, and Lizhen Cui. 2024. Edge-llm: A collaborative framework for large language model serving in edge computing. In *2024 IEEE International Conference on Web Services (ICWS)*. IEEE, 799–809.
- [13] Mu-Tien Chang, Paul Rosenfeld, Shih-Lien Lu, and Bruce Jacob. 2013. Technology comparison for large last-level caches (LLCs): Low-leakage SRAM, low write-energy STT-RAM, and refresh-optimized eDRAM. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. 143–154. <https://doi.org/10.1109/HPCA.2013.6522314>
- [14] Jiasi Chen and Xukan Ran. 2019. Deep learning with edge computing: A review. *Proc. IEEE* 107, 8 (2019), 1655–1674.
- [15] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2014. DaDianNao: A Machine-Learning Supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. 609–622. <https://doi.org/10.1109/MICRO.2014.58>
- [16] Kyungsang Cho, Yongjun Lee, Young H Oh, Gyoo-cheol Hwang, and Jae W Lee. 2014. eDRAM-based tiered-reliability memory with applications to low-power frame buffers. In *Proceedings of the 2014 international symposium on Low power electronics and design*. 333–338.
- [17] Ki Chul Chun, Pulkit Jain, Jung Hwa Lee, and Chris H. Kim. 2011. A 3T Gain Cell Embedded DRAM Utilizing Preferential Boosting for High Density and Low Power On-Die Caches. *IEEE Journal of Solid-State Circuits* 46, 6 (2011), 1495–1505. <https://doi.org/10.1109/JSSC.2011.2128150>
- [18] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *ArXiv abs/1803.05457* (2018). <https://api.semanticscholar.org/CorpusID:3922816>
- [19] Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. 2021. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011* (2021).
- [20] Jyotikrishna Dass, Shang Wu, Huihong Shi, Chaoqian Li, Zhifan Ye, Zhongfeng Wang, and Yingyan Lin. 2022. ViTALiT: Unifying Low-rank and Sparse Approximation for Vision Transformer Acceleration with a Linear Taylor Attention. *arXiv:2211.05109 [cs.CV]* <https://arxiv.org/abs/2211.05109>
- [21] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Srivankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Bin Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marr, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esibov, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junting Jia, Kalyan Vasudevan Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantalay-Yearly, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadar, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenying Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaozhengqiang Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuwei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattaflori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenber, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharame, Assaf Eisenman, Azadeh Yazdan, Beatt James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Nissim, Dingkang Wang, Dic Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcauta, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesca Caggioni, Francisco Guzman, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badre, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory Sizov, Guanyi, Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspasia, Hunter Goldman, Ibrahim Damlaj, Igor Molbyog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carville, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madiyan Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martyna Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Kenealy, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Ritter, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghavamurthy Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah

- Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghai Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Sounith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vitor Albiero, Vlad Ionescu, Vlad Poenaru, Vlad Tibor Mihailescu, Vladimir Ivanov, Wei Li, Wencheng Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaofang Wang, Xiaoqian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoadu Wen, Zhenyu Yang, and Zhiwei Zhao. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] <https://arxiv.org/abs/2407.21783>
- [22] EE Times. 2013. Intel eDRAM Attacks Graphics in Pre 3-D IC Days. http://www.eetimes.com/document.asp?doc_id=1263303
- [23] Chao Fang, Shouliang Guo, Wei Wu, Jun Lin, Zhongfeng Wang, Ming Kai Hsu, and Lingzhi Liu. 2022. An efficient hardware accelerator for sparse transformer neural networks. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2670–2674.
- [24] Eric J Fluehr, Steve Baumgartner, David Boerstler, John F Bulzacchelli, Timothy Diemitz, Daniel Dreps, George English, Joshua Friedrich, Anne Gattiker, Tilman Gloeckler, et al. 2014. The 12-core power8™ processor with 7.6 tb/s io bandwidth, integrated voltage regulation, and resonant clocking. *IEEE Journal of Solid-State Circuits* 50, 1 (2014), 10–23.
- [25] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323* (2022).
- [26] Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muenninghoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2021. A framework for few-shot language model evaluation. <https://doi.org/10.5281/zenodo.5371628>
- [27] Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2023. Model Tells You What to Discard: Adaptive KV Cache Compression for LLMs. *ArXiv abs/2310.01801* (2023). <https://api.semanticscholar.org/CorpusID:263609075>
- [28] Robert Giterman, Amir Shalom, Andreas Burg, Alexander Fish, and Adam Teman. 2020. A 1-Mbit Fully Logic-Compatible 3T Gain-Cell Embedded DRAM in 16-nm FinFET. *IEEE Solid-State Circuits Letters* 3 (2020), 110–113. <https://doi.org/10.1109/LSSC.2020.3006496>
- [29] Robert Giterman, Amir Shalom, Andreas Burg, Alexander Fish, and Adam Teman. 2020. A 1-Mbit Fully Logic-Compatible 3T Gain-Cell Embedded DRAM in 16-nm FinFET. *IEEE Solid-State Circuits Letters* 3 (2020), 110–113. <https://doi.org/10.1109/LSSC.2020.3006496>
- [30] Cong Guo, Jiaming Tang, Weiming Hu, Jingwen Leng, Chen Zhang, Fan Yang, Yun-Bo Liu, Minyu Guo, and Yuhao Zhu. 2023. OLiVe: Accelerating Large Language Models via Hardware-friendly Outlier-Victim Pair Quantization. *Proceedings of the 50th Annual International Symposium on Computer Architecture* (2023). <https://api.semanticscholar.org/CorpusID:258179335>
- [31] Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415* (2016).
- [32] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amit Gholami. 2024. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. arXiv:2401.18079 [cs.LG] <https://arxiv.org/abs/2401.18079>
- [33] Yunhai Hu, Zining Liu, Zhenyuan Dong, Tianfan Peng, Bradley McDanel, and Sai Qian Zhang. 2025. Speculative decoding and beyond: An in-depth survey of techniques. *arXiv preprint arXiv:2502.19732* (2025).
- [34] Yunhai Hu, Tianhua Xia, Zining Liu, Rahul Raman, Xingyu Liu, Bo Bao, Eric Sather, Vithursan Thangarasa, and Sai Qian Zhang. 2025. DREAM: Drafting with Refined Target Features and Entropy-Adaptive Cross-Attention Fusion for Multimodal Speculative Decoding. *arXiv preprint arXiv:2505.19201* (2025).
- [35] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lampe, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. arXiv:2310.06825 [cs.CL] <https://arxiv.org/abs/2310.06825>
- [36] Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551* (2017).
- [37] Kingston Technology. 2024. *Embedded Flash and DRAM Components*. <https://www.kingston.com/en/solutions/embedded-and-industrial>
- [38] W. Kong, P. C. Parries, G. Wang, and S. S. Iyer. 2008. Analysis of Retention Time Distribution of Embedded DRAM - A New Method to Characterize Across-Chip Threshold Voltage Variation. In *2008 IEEE International Test Conference*. 1–7. <https://doi.org/10.1109/TEST.2008.4700556>
- [39] HT Kung. 1986. Memory requirements for balanced computer architectures. *ACM SIGARCH Computer Architecture News* 14, 2 (1986), 49–54.
- [40] Jungi Lee, Wonbeom Lee, and Jaewoong Sim. 2024. Tender: Accelerating Large Language Models via Tensor Decomposition and Runtime Requantization. *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)* (2024), 1048–1062. <https://api.semanticscholar.org/CorpusID:270620037>
- [41] 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.
- [42] Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*. PMLR, 19274–19286.
- [43] Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. TruthfulQA: Measuring How Models Mimic Human Falsehoods. arXiv:2109.07958 [cs.CL] <https://arxiv.org/abs/2109.07958>
- [44] Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. 2024. MiniCache: KV Cache Compression in Depth Dimension for Large Language Models. *arXiv preprint arXiv:2405.14366* (2024).
- [45] Hanxiao Liu, Zihang Dai, David So, and Quoc V Le. 2021. Pay attention to mlps. *Advances in neural information processing systems* 34 (2021), 9204–9215.
- [46] Lian Liu, Haimeng Ren, Long Cheng, Zhaohui Xu, Yudong Pan, Mengdi Wang, Xiaowei Li, Yinhe Han, and Ying Wang. 2024. COMET: Towards Partial W4A4KV4 LLMs Serving. arXiv:2410.12168 [cs.AR] <https://arxiv.org/abs/2410.12168>
- [47] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyriolidis, and Anshumali Shrivastava. 2023. Scissorhands: Exploiting the Persistence of Importance Hypothesis for LLM KV Cache Compression at Test Time. *ArXiv abs/2305.17118* (2023). <https://api.semanticscholar.org/CorpusID:258947558>
- [48] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache. *ArXiv abs/2402.02750* (2024). <https://api.semanticscholar.org/CorpusID:267413049>
- [49] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen (Henry) Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. KIVI: a tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning (Vienna, Austria) (ICML'24)*. JMLR.org, Article 1311, 13 pages.
- [50] Siyuan Lu, Meiqi Wang, Shuang Liang, Jun Lin, and Zhongfeng Wang. 2020. Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer. In *2020 IEEE 33rd International System-on-Chip Conference (SOCC)*. IEEE, 84–89.
- [51] Bradley McDanel, Sai Qian Zhang, Yunhai Hu, and Zining Liu. 2025. PipeSpec: Breaking Stage Dependencies in Hierarchical LLM Decoding. *arXiv preprint arXiv:2505.01572* (2025).
- [52] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer Sentinel Mixture Models. arXiv:1609.07843 [cs.CL]
- [53] Ramesh Nallapati, Bowen Zhou, Cicero Nogueira dos santos, Caglar Gulcehre, and Bing Xiang. 2016. Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond. arXiv:1602.06023 [cs.CL] <https://arxiv.org/abs/1602.06023>
- [54] Duy-Thanh Nguyen, Nhut-Minh Ho, and Ik-Joon Chang. 2019. St-DRC: Stretchable DRAM refresh controller with no parity-overhead error correction scheme for energy-efficient DNNs. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.
- [55] NVIDIA Corporation. 2024. NVIDIA System Management Interface (nvidia-smi). <https://developer.nvidia.com/system-management-interface>.
- [56] 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).
- [57] Eunhyeok Park, Dongyoung Kim, and Sungjoo Yoo. 2018. Energy-efficient neural network accelerator based on outlier-aware low-precision computation. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 688–698.
- [58] Alicia Parrish, Angelica Chen, Nikita Nangia, Vishakh Padmakumar, Jason Phang, Jana Thompson, Phu Mon Htut, and Samuel R. Bowman. 2022. BBQ: A Hand-Built Bias Benchmark for Question Answering. arXiv:2110.08193 [cs.CL] <https://arxiv.org/abs/2110.08193>

- [59] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems* 5 (2023), 606–624.
- [60] Matt Poremba, Sparsh Mittal, Dong Li, Jeffrey S. Vetter, and Yuan Xie. 2015. DESTINY: A tool for modeling emerging 3D NVM and eDRAM caches. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1543–1546. <https://doi.org/10.7873/DATE.2015.0733>
- [61] Team PyTorch. 2023. Accelerating generative ai with pytorch ii: Gpt, fast. <https://pytorch.org/blog/accelerating-generative-ai-2/>
- [62] Jiajun Qin, Tianhua Xia, Cheng Tan, Jeff Zhang, and Sai Qian Zhang. 2025. PICACHU: Plug-In CGRA Handling Upcoming Nonlinear Operations in LLMs. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 845–861.
- [63] Alec Radford. 2018. Improving language understanding by generative pre-training. (2018).
- [64] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. <https://api.semanticscholar.org/CorpusID:160025533>
- [65] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.
- [66] Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. 2019. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507* (2019).
- [67] Haihao Shen, Hanwen Chang, Bo Dong, Yu Luo, and Hengyu Meng. 2023. Efficient LLM Inference on CPUs. *arXiv:2311.00502 [cs.LG]* <https://arxiv.org/abs/2311.00502>
- [68] 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.
- [69] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE internet of things journal* 3, 5 (2016), 637–646.
- [70] J. Soriano. 2023. *Accelerating Generative AI at the Edge*. <https://www.qualcomm.com/news/onq/2023/11/accelerating-generative-ai-at-the-edge> [Online; accessed 29-Mar-2025].
- [71] Jacob R. Stevens, Rangharajan Venkatesan, Steve Dai, Brucek Khailany, and Anand Raghunathan. 2021. Softermax: Hardware/Software Co-Design of an Efficient Softmax for Transformers. *arXiv:2103.09301 [cs.AR]* <https://arxiv.org/abs/2103.09301>
- [72] Manu Suryavansh. 2020. Google Coral Edge TPU Board Vs NVIDIA Jetson Nano Dev board Hardware Comparison.
- [73] Qwen Team. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671* (2024).
- [74] 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).
- [75] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).
- [76] Fengbin Tu, Weiwei Wu, Shouyi Yin, Leibo Liu, and Shaojun Wei. 2018. RANA: Towards Efficient Neural Acceleration with Refresh-Optimized Embedded DRAM. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 340–352. <https://doi.org/10.1109/ISCA.2018.00037>
- [77] Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Huaijie Wang, Lingxiao Ma, Fan Yang, Ruiping Wang, Yi Wu, and Furu Wei. 2023. BitNet: Scaling 1-bit Transformers for Large Language Models. *arXiv:2310.11453 [cs.CL]* <https://arxiv.org/abs/2310.11453>
- [78] Hanrui Wang, Zhekai Zhang, and Song Han. 2020. SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning. *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)* (2020), 97–110. <https://api.semanticscholar.org/CorpusID:229298088>
- [79] Xiaotian Wang, Teng Tian, Letian Zhao, Wei Wu, and Xi Jin. 2022. Exploration of balanced design in resource-constrained edge device for efficient CNNs. *IEEE Transactions on Circuits and Systems II: Express Briefs* 69, 11 (2022), 4573–4577.
- [80] Dieter F Wendel, Ron Kalla, James Warnock, Robert Cargnoni, Sam G Chu, Joachim G Clabes, Daniel Dreps, David Hrusecky, Josh Friedrich, Saiful Islam, et al. 2010. POWER™, a highly parallel, scalable multi-core high end server processor. *IEEE Journal of Solid-State Circuits* 46, 1 (2010), 145–161.
- [81] Tianhua Xia and Sai Qian Zhang. 2024. Hyft: A Reconfigurable Softmax Accelerator with Hybrid Numeric Format for both Training and Inference. In *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design (Newport Beach, CA, USA) (ISLPED '24)*. Association for Computing Machinery, New York, NY, USA, 1–6. <https://doi.org/10.1145/3665314.3670816>
- [82] Jingyang Xiang and Sai Qian Zhang. 2024. DFRot: Achieving Outlier-Free and Massive Activation-Free for Rotated LLMs with Refined Rotation. *arXiv preprint arXiv:2412.00648* (2024).
- [83] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient Streaming Language Models with Attention Sinks. *arXiv:2309.17453 [cs.CL]* <https://arxiv.org/abs/2309.17453>
- [84] Kai Xiao, Jing Wan, Hui Xie, Yuxuan Zhu, Tian Tian, Wei Zhang, Yingxin Chen, Jinshu Zhang, Lihui Zhou, Sheng Dai, et al. 2024. High performance Si-MoS2 heterogeneous embedded DRAM. *Nature Communications* 15, 1 (2024), 9782.
- [85] Xiao Xiong, Zhaorui Chen, Yue Liang, Minghao Tian, Jiaxing Shang, Jiang Zhong, and Dajiang Liu. 2025. DynaX: Sparse Attention Acceleration with Dynamic X:M Fine-Grained Structured Pruning. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Rotterdam, Netherlands) (ASPLOS '25). Association for Computing Machinery, New York, NY, USA, 260–274. <https://doi.org/10.1145/3676641.3715991>
- [86] Daliang Xu, Hao Zhang, Liming Yang, Ruiqi Liu, Gang Huang, Mengwei Xu, and Xuanzhe Liu. 2024. Fast On-device LLM Inference with NPUs. *arXiv:2407.05858 [cs.AI]* <https://arxiv.org/abs/2407.05858>
- [87] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingen Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xiancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. 2024. Qwen2 Technical Report. *arXiv:2407.10671 [cs.CL]* <https://arxiv.org/abs/2407.10671>
- [88] Dongjie Yang, XiaoDong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. 2024. PyramidInfer: Pyramid KV Cache Compression for High-throughput LLM Inference. *arXiv preprint arXiv:2405.12532* (2024).
- [89] Chengshuo Yu, Taegeun Yoo, Hyunjoon Kim, Tony Tae-Hyoung Kim, Kevin Chai Tshun Chuan, and Bongjin Kim. 2020. A logic-compatible eDRAM compute-in-memory with embedded ADCs for processing neural networks. *IEEE Transactions on Circuits and Systems I: Regular Papers* 68, 2 (2020), 667–679.
- [90] Zhongkai Yu, Shengwen Liang, Tianyun Ma, Yunke Cai, Ziyuan Nan, Di Huang, Xinkang Song, Yifan Hao, Jie Zhang, Tian Zhi, et al. 2024. Cambricon-llm: A chiplet-based hybrid architecture for on-device inference of 70B llm. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1474–1488.
- [91] Zhongzhi Yu, Zheng Wang, Yuhuan Li, Ruijie Gao, Xiaoya Zhou, Sreenidhi Reddy Bommu, Yang Zhao, and Yingyan Lin. 2024. Edge-llm: Enabling efficient large language model adaptation on edge devices via unified compression and adaptive layer voting. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.
- [92] Ali Hadi Zadeh, Isak Edö, Omar Mohamed Awad, and Andreas Moshovos. 2020. Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 811–824.
- [93] Hengrui Zhang, August Ning, Rohan Baskar Prabhakar, and David Wentzlaff. 2024. Llmcompas: Enabling efficient hardware design for large language model inference. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1080–1096.
- [94] Mingjin Zhang, Xiaoming Shen, Jianlong Cao, Zeyang Cui, and Shan Jiang. 2024. EdgeShard: Efficient LLM Inference via Collaborative Edge Computing. *IEEE Internet of Things Journal* (2024), 1–1. <https://doi.org/10.1109/JIOT.2024.3524255>
- [95] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuhui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myla Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open Pre-trained Transformer Language Models. *arXiv:2205.01068 [cs.CL]* <https://arxiv.org/abs/2205.01068>
- [96] Sai Qian Zhang, Bradley McDanel, and H. T. Kung. 2021. FAST: DNN Training Under Variable Precision Block Floating Point with Stochastic Rounding. *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)* (2021), 846–860. <https://api.semanticscholar.org/CorpusID:240288620>
- [97] Sai Qian Zhang, Thierry Tambe, Nestor Cuevas, Gu-Yeon Wei, and David Brooks. 2023. CAMEL: Co-Designing AI Models and Embedded DRAMs for Efficient On-Device Learning. *arXiv:2305.03148 [cs.AR]* <https://arxiv.org/abs/2305.03148>
- [98] Zhenyu (Allen) Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruiqi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. *ArXiv abs/2306.14048* (2023). <https://api.semanticscholar.org/CorpusID:259263947>

- [99] Youpeng Zhao, Di Wu, and Jun Wang. 2024. ALISA: Accelerating Large Language Model Inference via Sparsity-Aware KV Caching. *arXiv preprint arXiv:2403.17312* (2024).
- [100] Minxuan Zhou, Weihong Xu, Jaeyoung Kang, and Tajana Rosing. 2022. Transpim: A memory-based acceleration via software-hardware co-design for transformer. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 1071–1085.