Oaken: Fast and Efficient LLM Serving with Online-Offline Hybrid KV Cache Quantization

Minsu Kim*
KAIST
Daejeon, Republic of Korea
mskim@casys.kaist.ac.kr

Seongmin Hong*
HyperAccel
Seoul, Republic of Korea
sm.hong@hyperaccel.ai

RyeoWook Ko
KAIST
Daejeon, Republic of Korea
ryeowookko@kaist.ac.kr

Soongyu Choi
KAIST
Daejeon, Republic of Korea
soongyu1291@kaist.ac.kr

Hunjong Lee
HyperAccel
Seoul, Republic of Korea
hj.lee@hyperaccel.ai

Junsoo Kim
HyperAccel
Seoul, Republic of Korea
js.kim@hyperaccel.ai

Joo-Young Kim
HyperAccel
Seoul, Republic of Korea
jy.kim@hyperaccel.ai

Jongse Park
KAIST
Daejeon, Republic of Korea
jspark@casys.kaist.ac.kr

Abstract

Modern Large Language Model (LLM) serving system batches multiple requests to achieve high throughput, while batching attention operations is challenging, rendering *memory bandwidth* a critical bottleneck. Today, to mitigate this issue, the community relies on high-end GPUs with multiple high-bandwidth memory (HBM) channels. Unfortunately, HBM's high bandwidth often comes at the expense of limited *memory capacity*, necessitating systems to scale, which reduces core utilization and increases costs. Moreover, recent advancements enabling longer contexts for LLMs have substantially increased the key-value (KV) cache size, further intensifying the pressures on memory capacity. To lower the pressure, the literature has explored KV cache quantization techniques, which commonly use low bitwidth (e.g., INT4) for most values, selectively using higher bitwidth (e.g., FP16) for outlier values. While this approach helps achieve high accuracy and low bitwidth simultaneously, it comes with the limitation that the cost for online outlier detection is excessively high, negating the advantages of quantization.

Inspired by these insights, we propose Oaken, an acceleration solution that achieves high accuracy and high performance simultaneously through co-designing algorithm and hardware. To effectively find a sweet spot in the accuracy-performance trade-off space of KV cache quantization, Oaken employs an online-offline hybrid approach, setting outlier thresholds offline, which are then used to determine the quantization scale online. To translate the proposed algorithmic technique into tangible performance gains, Oaken also comes with custom quantization/dequantization engines and memory management units that can be integrated with any LLM accelerators. We built an Oaken accelerator on top of an LLM accelerator, LPU, and conducted a comprehensive evaluation. Our experiments show that for a batch size of 256, Oaken achieves up to 1.58× throughput improvement over NVIDIA A100 GPU, incurring a minimal accuracy loss of only 0.54% on average, compared to state-of-the-art KV cache quantization techniques.

CCS Concepts

• **Computer systems organization** → **Neural networks**; Multi-core architectures; • **Computing methodologies** → *Neural networks.*

Keywords

Accelerator; Large Language Models (LLM); Serving; Batched Inference; Quantization; Key-Value (KV) Cache

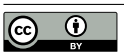
ACM Reference Format:
Minsu Kim, Seongmin Hong, RyeoWook Ko, Soongyu Choi, Hunjong Lee, Junsoo Kim, Joo-Young Kim, and Jongse Park. 2025. Oaken: Fast and Efficient LLM Serving with Online-Offline Hybrid KV Cache Quantization. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25), June 21–25, 2025, Tokyo, Japan.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3695053.3731019

1 Introduction

The recent advent of large language models (LLMs) has significantly impacted the computing industry. Almost every sector of the modern economy is exploring the adoption of LLMs, with many already actively using them for various applications. Most real-world applications rely on hyperscaler-provided LLM serving systems because on-premise LLM deployment requires prohibitive costs.

As LLM inferencing is for multi-tenant environments, the serving systems batch multiple requests to parallelize the inference computation [12, 14, 32, 36, 50, 64, 77]. While batching promises a significant throughput boost for operations where operand matrices can be shared across requests on GPUs and other AI accelerators, attention layers in transformer-based LLMs consist of *un*-batchable operations with request-specific, *un*-shareable operands, lowering

*Co-first authors who contributed equally to this work.



ISCA '25, Tokyo, Japan

ACM ISBN 979-8-4007-1261-6/2025/06
https://doi.org/10.1145/3695053.3731019

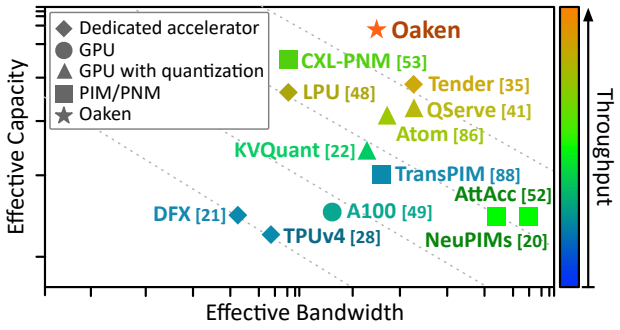


Figure 1: Existing solutions for LLM inference serving systems plotted on the bandwidth-capacity trade-off space. The "effective bandwidth" and "effective capacity" represent the scale of data that can be transmitted to/from and stored on memory, respectively. We also delineate their corresponding throughput (i.e., tokens/sec) using the colors presented on a 1D heatmap on the right side.

hardware utilization. As the *un*-batchable operations cannot exploit on-chip data reuse, they produce enormous memory read, causing ***memory bandwidth*** to become the key system bottleneck.

Additionally, request batching in LLM inference has another resource implication. LLMs generate key-value activation and cache them in memory for computation reuse, often called *KV cache*. As this KV cache is not shared across different user requests, the large batch size is directly translated into a large KV cache size, requiring massive ***memory capacity***. Furthermore, as KV cache size scales linearly with the sequence length, the recent trend of supporting very long sequences (e.g., 2 million tokens [76]) places even greater pressure on memory capacity.

Consequently, LLM serving systems require both high bandwidth and high capacity to enable fast inferencing. This resource demand aligns with the immense user demand for LLM services, making it challenging for service providers to build cost-effective LLM serving systems. Existing solutions often choose to trade-off one resource for the other, as visualized in Figure 1. Below, we classify the existing solutions into the following three categories:

(1) Prioritizing *bandwidth* over *capacity*: Currently, using HBM-equipped GPUs is the de-facto standard solution for LLM inference processing [28, 45, 49]. While this approach achieves massive bandwidth, it often compromises capacity, forcing systems to *scale out*, which not only reduces core utilization but also increases the cost of building the system.

(2) Leveraging PIM and/or PNM: Even with HBMs, LLM inferencing systems still face bandwidth bottlenecks. To address this challenge, recent works have explored the near-data processing (NDP) paradigm, leveraging PIM [20, 52, 59] and/or PNM [53]. While these approaches mitigate the bandwidth bottleneck, their inherent nature requires further reductions in memory capacity, limiting their viability as a fundamental solution.

(3) Exploring LLM quantization strategies: One fundamental strategy for jointly addressing the conflicting objectives is to minimize the memory footprint required for LLM inferencing. To achieve this goal, a large body of prior work [10, 17, 22, 30, 31, 33, 40, 41, 43, 75, 78, 79, 86] have recently developed LLM-targeted quantization techniques. While these techniques successfully achieve significant reductions in bitwidth, they often prioritize minimizing bitwidth over effectively translating these reductions into practical inferencing speedups.

Alone, none of the solutions is sufficient for building fast and efficient LLM serving systems, which motivates us to develop an acceleration solution, namely Oaken. By jointly leveraging algorithmic and hardware techniques, Oaken achieves otherwise unattainable levels of *effective* bandwidth and capacity, resulting in substantially higher throughput than alternatives, as shown in Figure 1. Oaken comprises (1) **Algorithmic technique:** an online-offline hybrid KV cache quantization technique, and (2) **Hardware technique:** the hardware incarnations of the proposed algorithms, including quantization/dequantization engines and memory management units that can be integrated with any existing LLM accelerators. Oaken makes the following contributions:

(1) Online-offline hybrid KV cache quantization: Many of the recently-proposed KV cache quantization techniques commonly use low bitwidth (e.g., INT4) for most values, while selectively using higher bitwidth (e.g., FP16) for outlier values [15, 19, 22, 33, 80, 86]. While this approach achieves high accuracy and low bitwidth, the prohibitively high cost of online threshold calculation or mixed-precision computation renders it nearly impractical for real-world use cases. Thus, there is a critical need for a cost-effective solution to identify thresholds that distinguish outliers from inliers, enabling the translation of KV cache quantization into tangible speedups. To achieve this goal, Oaken employs an online-offline hybrid approach, where data-agnostic outlier thresholds are determined at *offline* and subsequently applied to set the quantization scale at *online*. Furthermore, Oaken introduces a quantization loss mitigation technique that shifts values toward a smaller range, converting outliers into inliers prior to quantization. Finally, Oaken stores the quantized values by using dense tensors for inliers and fusing sparse outliers into the dense tensors.

(2) Quantization-aware hardware modules for LLM accelerators: We devise custom quantization/dequantization engines and memory management units, which are aware of the proposed quantization algorithm. These hardware modules can be integrated with any existing LLM accelerators such as GPUs, NPUs, and LLM-customized ones [21, 23, 53, 82]. We place these modules in the DMA unit that is commonly present in modern LLM accelerators. In designing the memory management unit (MMU), the challenge is to achieve the maximal bandwidth, which is close to the physical limit, while effectively laying out the dense and sparse matrices in memory. We design the MMU with two management tables for dense and sparse data, respectively, to handle virtual-to-physical address mappings and manage the single address space at page granularity. This design maximizes memory bandwidth utilization while avoiding fragmentation and burst order issues.

To evaluate the effectiveness of Oaken, we use eight different LLMs that include OPT [84], Llama2 [66], Mistral [25], and Mixtral [26], with varying sizes. We use Wikitext, PIQA, Winogrande, and Hellaswag datasets, which are widely used in prior

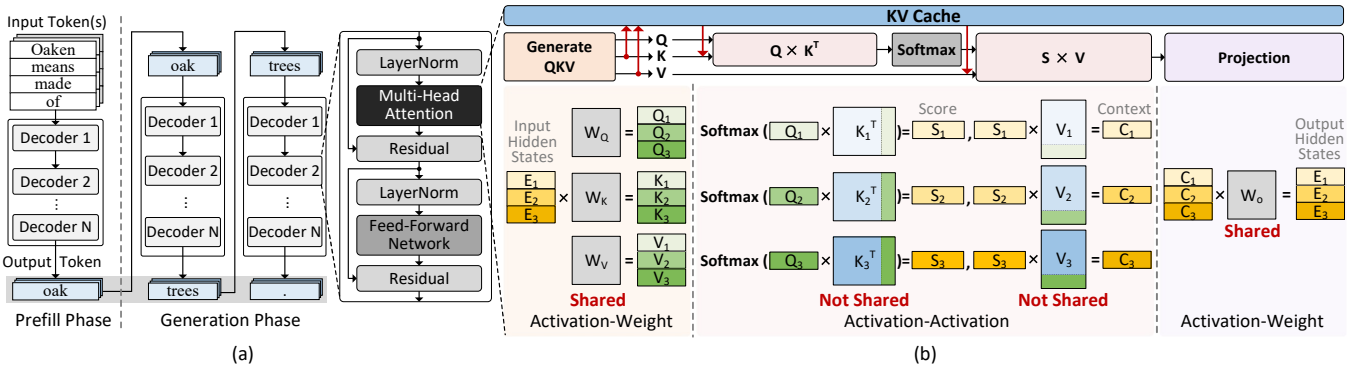


Figure 2: (a) Structure of LLM inference and decoder layer during the prefill and generation phases. (b) Operations in the multi-head attention layer, including activation-weight and activation-activation operations, during the generation phase of batched inference for three requests.

works [11, 22, 31, 33, 37, 41, 73, 86, 87]. We synthesize the SystemVerilog RTL code of our accelerator in TSMC 28nm technology using Synopsys Design Compiler, which offers the area information of each component in the accelerator. Our experimental results report that compared to NVIDIA A100 using state-of-the-art KV cache quantization techniques, Oaken offers up to 1.58× throughput speedup, owing to the bitwidth reduction that reaches up to 70.0%. We achieve this speedup by only introducing a modest 0.87% accuracy degradation, which demonstrates the algorithmic robustness of our KV cache quantization technique. Furthermore, Oaken modules incur only 8.21% area overhead, which is negligible given the significant performance benefits they enable. These advantages demonstrate that Oaken achieves the dual objectives of bandwidth and capacity, representing an important step toward building fast and cost-effective LLM inference serving systems.

2 Background

2.1 A Primer on LLM Inference

As illustrated in Figure 2(a), inference of large language model is mainly divided into two phases: the prefill phase and the generation phase [4, 8, 25, 66, 84]. The prefill phase takes input tokens and passes them through decoder layers to generate a single output token. In the generation phase, the token generated in the previous iteration is used to produce the next token. This process is autoregressive, with each iteration generating one token.

LLMs typically consist of multiple decoder layers, with multi-head attention being one of the key operations. As visualized in Figure 2(b), the multi-head attention begins with generating the query, key, and value activations. The key and value are buffered in the on-chip memory for subsequent operations and are also stored as *KV cache* in the off-chip memory for future iterations. The query is then multiplied with the transposed key, which is directly fetched from the on-chip memory during the prefill phase, while it is loaded from the off-chip KV cache during the generation phase. Similarly, subsequent computation using the value requires access to either on-chip memory or the KV cache in the same manner as the key.

2.2 Batching for High-Throughput LLM Inference

Batching is a commonly used method to enhance inference throughput in LLM serving systems [1, 20, 32, 64, 77]. It entails processing multiple requests simultaneously and boosts inference throughput by converting memory-bound operations into compute-bound ones through on-chip data reuse. For instance, in the feed-forward network, weights read from memory can be reused across multiple requests, reducing memory access and increasing throughput.

Figure 2(b) depicts the two types of operations that comprises the multi-head attention layer. In activation-weight operations, including Generate QKV and Projection, the activations of the requests share the same weights. This enables on-chip data reuse, reducing memory access and execution time. However, in activation-activation operations, where the query is multiplied by the transposed key or the score is multiplied by the value, each request requires distinct keys or values, making batched processing challenging. As a result, batching fails to reduce memory accesses in this case, providing no performance benefits in terms of latency.

2.3 LLM Quantization

Recently, as demands for LLM inference ever-increasing memory capacity, the community has explored solutions to mitigate this demand through quantization techniques that reduce the bit precision of weights or activations. Many of these studies focus on easing the memory pressure caused by model weights, which is particularly effective for accelerating small-batch inference [17, 30, 40, 42, 60, 69, 70, 78]. However, these methods achieve limited speedup for larger batches and long sequence lengths due to the KV cache, whose size scales with the batch size and sequence length. Approaches to quantize both weights and activations also face similar limitations [15, 34, 64, 71]. To this end, researchers have recently started directing their attention to KV cache quantization. Some of these works propose quantizing the keys and values of the attention layers on a per-vector basis [22, 41, 43], while others propose quantizing the KV cache into multiple vector groups of similar magnitudes, applying channel reordering technique [5, 35, 39, 86].

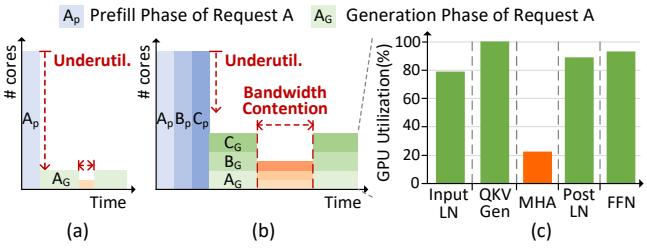
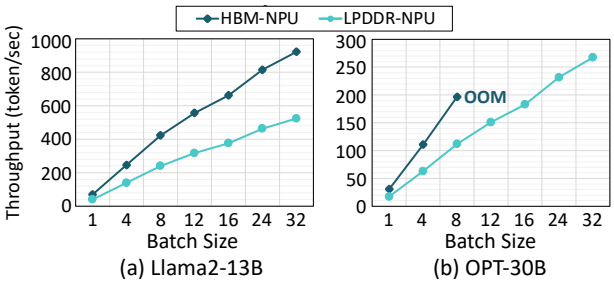


Figure 3: Characteristic analysis of LLM inference for (a) single request and (b) batched multiple requests. (c) Utilization measurement during the generation phase with batched multiple requests using NVIDIA A100 GPU.



	HBM-NPU	LPDDR-NPU
# Compute cores	256	256
Matrix unit dim. / core	32×32	32×32
Vector unit dim. / core	32	32
Peak FP16 TFLOPS	270.3	270.3
Memory capacity	80 GB	256 GB
Memory bandwidth	2.0 TB/s	1.1 TB/s

(c)

Figure 4: Throughput of accelerators equipped with HBM and LPDDR memory when using (a) Llama2-13B and (b) OPT-30B (OOM refers to "Out-of-Memory."). (c) Accelerator specification with HBM and LPDDR memory.

3 Motivation

This paper focuses on effectively utilizing batched LLM inference to enhance throughput. To accomplish this, we thoroughly analyze the characteristics of LLM inference, particularly comparing the *un*-batched and batched execution in this section.

3.1 Characterization of Batched LLM Inference

Single-instance inferencing. We analyze the bottlenecks of batched inference by examining accelerator core utilization in both single and batched request scenarios. Figure 3(a) shows the difference in core utilization between the prefill and generation phases when processing a single request. During the prefill phase, multiple cores are utilized to process multiple input tokens in a parallelized manner. However, the generation phase processes only one token from the previous iteration, making it highly sequential process that limits parallelization. Therefore, the accelerator can utilize only a few cores, resulting in high latency in generating all output tokens.

Batched inferencing. Figure 3(b) illustrates core utilization for batched inference. Similar to the single-request scenario, the prefill phase for batched requests is also parallelizable across multiple cores, resulting in high core utilization. However, the generation phase when processing batched requests still shows underutilization, engendering even longer latency due to bandwidth contention. This is because key-value from attention layers cannot be batched and shared across requests, and the accelerator utilizes a limited number of cores per request. Figure 3(c) shows GPU core utilization during the generation phase of Llama2-13B model on an NVIDIA A100 GPU, indicating that underutilization primarily arises from the multi-head attention operations.

Batching: the double-edged solution. A straightforward solution to this issue is to increase the batch size to fully utilize all the cores, but this introduces a new challenge. The key-value size required for attention operations is proportional to the batch size. However, as the batch size increases, the memory footprint due to the KV cache also grows, resulting in longer operation latency which is bounded by memory bandwidth. Recent techniques, such as grouped and multi-query attentions, reduce the memory overhead by shrinking the KV cache size but cannot fully address this issue [3, 61].

Our analysis suggests the **two key observations**:

(1) While the prefill phase fully utilizes the compute resources, the generation phase often fails to do so. This imbalance offers an opportunity to enhance throughput with a larger batch size, while it also increases the demand for ***memory capacity*** due to the KV cache, driving up system construction costs.

(2) The increased KV cache size due to the aggressive batched processing also causes a high demand for ***memory bandwidth*** to efficiently compute attention operations, delaying the entire generation phase during batched inference.

3.2 Trade-off between Bandwidth and Capacity

While LLM serving demands both high bandwidth and large capacity, memory technologies exhibit a trade-off between these two resources. High-bandwidth memory (HBM) sacrifices a substantial portion of capacity to deliver exceptional bandwidth, whereas Low-Power Double Data Rate (LPDDR) DRAM occupies the opposite end of the trade-off space. Motivated by these observations and insights, we conduct preliminary studies to better understand the performance implications of these conflicting resources on LLM inferencing throughput.

Figure 4(a) and (b) present throughput comparison results for Llama2-13B and OPT-30B models using two accelerator variants with different memory types. We set both the input and output sequence length to 1K and use an existing LLM-customized accelerator [48]. Figure 4(c) lists the specifications of the two accelerators evaluated. For the smaller LLM model, both accelerators achieve sublinearly-scaling throughput as the batch size increases, with the HBM-based accelerator achieving the highest performance due to its superior bandwidth. However, for larger LLM models and batch sizes, scaling challenges become evident for the HBM-based accelerator. In contrast, the LPDDR-based accelerator can accommodate larger batches, demonstrating the best performance.

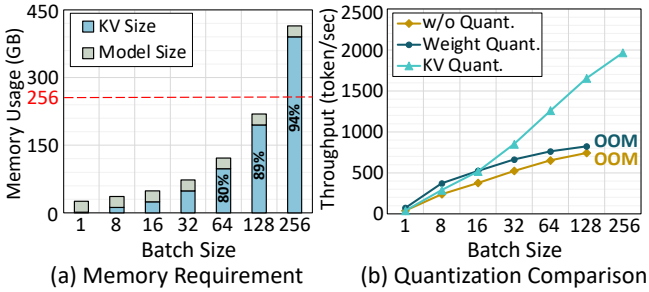


Figure 5: (a) Memory usage breakdown to KV cache and model parameters of Llama2-13B model as batch size sweeps from 1 to 256. (b) Throughput comparison among no quantization, weight and KV cache quantization of Llama2-13B model inference. Experiment is conducted with LPDDR-NPU.

To summarize, while high memory bandwidth is crucial for achieving high throughput, sufficient memory capacity is also essential to efficiently serve LLMs, particularly with larger models and batches. Several recent studies also emphasize the importance of memory capacity in LLM inference and suggest introducing large-capacity memory into the LLM accelerators [53, 83]. Despite the industry's growing demand for large-scale LLM inference, current memory technologies struggle to meet both capacity and bandwidth requirements, necessitating a choice. We explore this trade-off by evaluating our solution, equipped with either LPDDR or HBM memory, across various batched inference scenarios.

3.3 KV Cache Quantization

Leveraging the memory that offers either sufficient bandwidth or capacity, combined with scaling out the system, can help address the bottlenecks in batched inference. However, this approach incurs substantial system-building costs and severe resource underutilization, making it neither a fundamental nor a sustainable solution.

Algorithmic approach to the memory wall. To tackle the memory wall challenge, the community has explored algorithmic approaches to reduce the demands on both conflicting resources fundamentally. Quantization is one such direction, widely recognized for its ability to alleviate both capacity and bandwidth bottlenecks simultaneously. As discussed in Section 2.3, many efforts focus on weight quantization and reducing the computation workload to accelerate LLM inference [15, 17, 33, 51, 71, 78]. However, our experimental results show that weight-only quantization has a limited impact on addressing the memory wall in the batched scenarios.

Limitations of weight-only quantization techniques. Figure 5(a) illustrates the memory capacity requirements for model weights and KV cache as batch size increases. While the memory usage for weights remains constant, the KV cache size grows, eventually dominating the entire device memory. Figure 5(b) compares the performance between 4-bit weight quantization and 4-bit KV cache quantization. The result shows a minimal performance improvement from weight quantization, showing its ineffectiveness in addressing the memory wall for batched LLM inference. However, KV cache quantization delivers larger performance gains, demonstrating its effectiveness in relieving memory pressure.

Limitations of existing KV cache quantization solutions. Recently, a large body of prior works has explored the KV cache quantization methods [5, 35, 41, 43, 86]. While these works have pioneered a novel research direction, they are constrained by substantial runtime overhead to enable quantization, compromises in accuracy for faster quantization, or a combination of both. QServe [41] and Atom [86] reorder key-value channels by applying the transformation matrix adopted in RPTQ [78], while QServe also handles outliers by applying the scaling matrix introduced by SmoothQuant [71]. Tender [35] performs channel reordering via indirect indexing and groups key-value channels with similar magnitude. These solutions have a limitation in that they come with accuracy degradation due to their low quantization granularity and suffer from additional runtime overhead of channel reordering. KIVI [43] and KVQuant [22] propose to use per-vector mixed-precision quantization based on the insight that each KV channel exhibits a distinct pattern in the magnitude of its values. These methods minimize accuracy loss by isolating outlier channels from the rest, but they incur substantial overhead from sorting operations or mixed-precision computations, which largely offsets the performance gains of quantization.

These limitations in existing solutions emphasize the need for a KV cache quantization technique that achieves low bitwidth without compromising accuracy. Furthermore, such low bitwidth must translate into tangible throughput improvements at the hardware level. To meet these demanding requirements, we propose an algorithm-hardware co-designed acceleration solution for efficient batched LLM inference, Oaken.

4 Oaken's KV Quantization Algorithm

Our quantization algorithm is driven by three primary empirical properties we obtain from observing the value distribution of the KV cache during LLM inference. Building upon the three observed properties, we incorporate three algorithmic techniques into Oaken's KV cache quantization. Below, we will first discuss the empirical analysis results for KV cache distribution and then describe the three techniques one by one.

4.1 Observations in KV Distribution

Existing LLM quantization methods often suggest that dealing with large values has a significant impact on model accuracy [15, 19, 22, 33, 71, 80, 86]. Moreover, other prior works suggest that small values near zero can vanish due to underflow during quantization, leading to larger error [2, 13, 27, 34]. These observations underscore the importance of analyzing the distribution and characteristics of the quantization targets. We examine the value distribution of the KV cache across several LLMs and datasets and derive key insights for designing effective quantization techniques.

Observation 1. Figure 6(a) presents the minimum and maximum range of KV cache values for each decoder layer across various LLMs using the Wikitext2 dataset. Notably, the magnitude of keys and values varies across models and among decoder layers within each model. These variations are distinctive properties of each model and decoder layer, driven by differences in their model weights. From this observation, we gain the insight that the quantization factor should be determined separately for each model and its individual decoder layers.

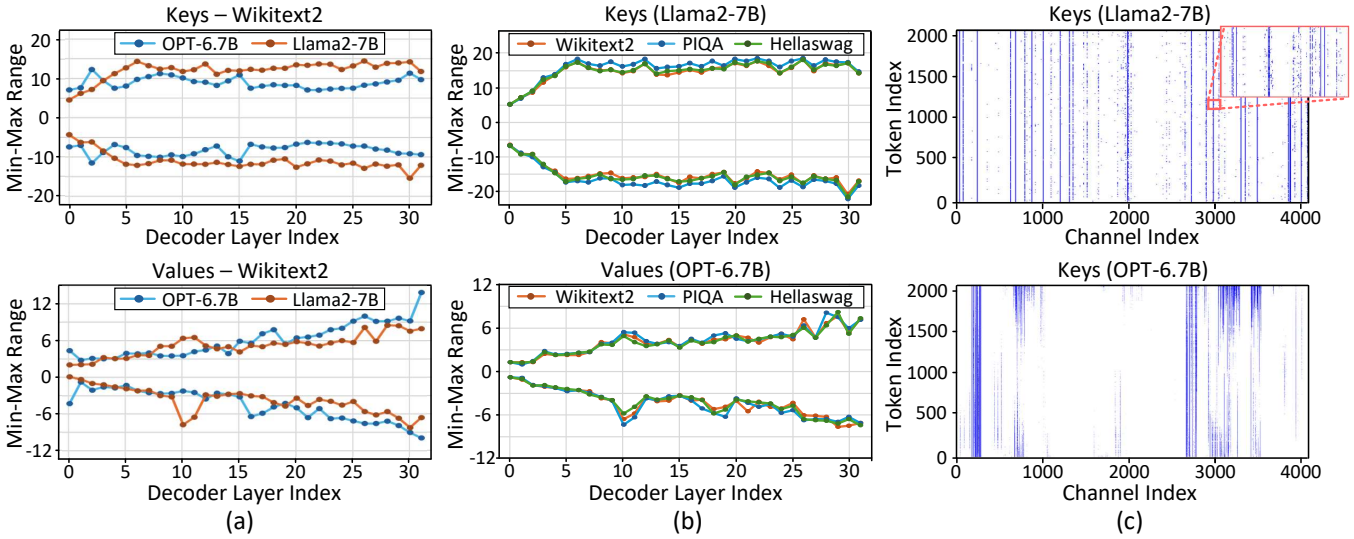


Figure 6: (a) Range of KV values from each decoder layer of OPT-6.7B and Llama2-7B models using Wikitext2 dataset. (b) Range of KV values from each decoder layer of Llama2-7B model using Wikitext2, PIQA, and Hellaswag dataset. (c) Distribution of the top 4% keys from the 6th decoder layer of Llama2-7B and OPT-6.7B models using Wikitext2 dataset.

Observation 2. Figure 6(b) shows the minimum and maximum range of KV values when using Llama2-7B model with Wikitext2, PIQA, and Hellaswag datasets. We see that the range of KV cache values remains consistent across these datasets. This observation implies there is no need for the quantization factor to be tailored to the input sequences but only a global quantization factor per layer.

Observation 3. Prior works have identified a pattern in the magnitude of channel within the KV cache and proposed a per-vector or per-vector-group quantization technique to leverage this pattern [22, 41, 43, 86]. Figure 6(c) presents the distribution of the top 4% values of the key from the sixth layer of the Llama2-7B and OPT-6.7B models. The distributions exhibit multiple vertical lines, indicating that high-magnitude values are concentrated in specific channels. This pattern in the channel magnitudes, aligning with previous observations, suggests the need for per-vector or per-vector-group quantization. However, this result also reveals exceptions to this pattern, appearing as discontinuous lines and dots, which cause an accuracy drop when using only a single quantization scale per vector or vector group. Based on this observation, we introduce multiple quantization groups within a channel to maintain accuracy, splitting the key-value vector into groups based on the magnitude of each element.

In summary, we derive the following **three insights** for designing Oaken's quantization technique:

- Oaken should determine the quantization factor separately for each model and decoder layer.
- Oaken can employ a common scaling factor regardless of input prompts, showing its insensitivity to data patterns.
- Oaken should use multiple quantization groups segmented by the magnitude of the values within each vector.

4.2 Algorithm Overview

We propose a quantization algorithm designed to improve performance by maximizing the compression ratio of the KV cache while minimizing quantization loss, based on the observations and insights from Section 4.1. Figure 7 illustrates the overall flow of Oaken's quantization algorithm. Oaken's quantization technique consists of three components: (1) Threshold-based online-offline hybrid quantization that separates and quantizes inlier and outlier values, (2) Group-shift quantization that quantizes values with larger magnitude, and (3) Fused dense-and-sparse encoding that minimizes capacity overhead due to sparse matrices. The following sections introduce a detailed design of each component.

4.3 Threshold-based Online-Offline Hybrid Quantization

Oaken minimizes quantization loss by isolating *outlier* values that are either exceptionally large or exceptionally small compared to typical *inlier* values. We propose a threshold-based online-offline hybrid quantization method for more fine-grained grouping. Oaken separates the *per-token* KV vector into three quantization groups: *outer*, *middle*, and *inner* (Figure 7(a)). The *middle* group consists of inliers where most KV values belong, while the *outer* and *inner* groups consist of outliers, with large and small magnitudes, respectively. Oaken prevents the quantization scale from being skewed due to large-magnitude outliers by isolating the outer group, and ensures the small-magnitude outliers do not vanish during quantization by isolating the inner group [2, 13, 27, 34].

Offline outlier threshold profiling. To separate outliers from inliers, the topK operation is typically used to maintain a constant ratio of outliers [22]. While this approach results in minimal quantization loss, the topK operation, essentially a sorting with a time

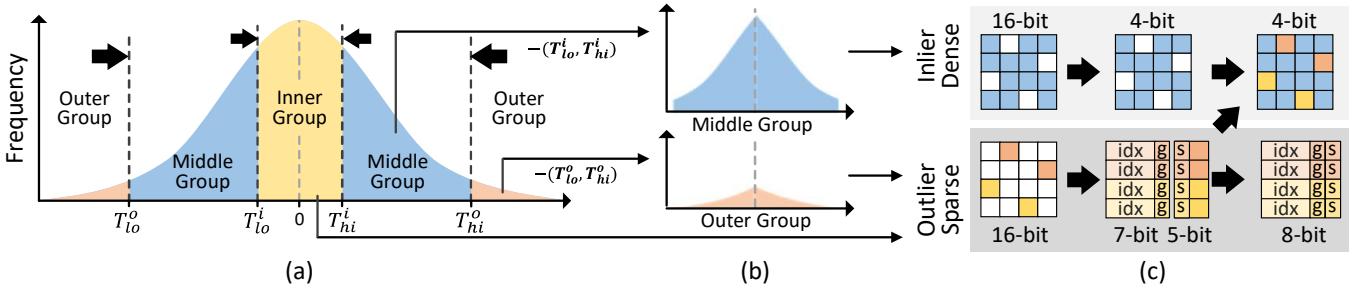


Figure 7: Oaken's quantization algorithm consisting of three components: (a) threshold-based online-offline hybrid quantization, (b) group-shift quantization, and (c) fused dense-and-sparse encoding.

complexity of $O(n \log n)$, introduces significant overhead when performed during inference, degrades end-to-end performance. Oaken employs offline outlier threshold profiling, leveraging the consistent characteristics in the distribution of KV cache discussed in Section 4.1 to avoid expensive online operations.

The criteria for splitting the KV cache into three groups are based on four thresholds determined through offline threshold profiling: T^o_{lo}, T^i_{lo}, T^i_{hi}, and T^o_{hi}. Using these thresholds, the outer group, middle group, and inner group are defined as follows:

$$\begin{aligned} G_o &= \{x \mid x < T^o_{lo} \text{ or } T^o_{hi} < x\}, \\ G_m &= \{x \mid T^o_{lo} \le x < T^i_{lo} \text{ or } T^i_{hi} < x \le T^o_{hi}\}, \\ G_i &= \{x \mid T^i_{lo} \le x \le T^i_{hi}\} \end{aligned} \tag{1}$$

Oaken performs approximately a hundred offline inferences with sample input prompts to gather distribution information from the KV cache of each decoder layer. The four group thresholds are extracted during the profiling process from the KV cache of each inference run using topK operations, and their averages are computed for each decoder layer. These statistics are then used to establish group thresholds for the KV cache. As discussed in Section 4.1, Oaken's offline profiling should be performed separately for each model, whereas the KV cache distribution and the profiling are independent of both the profiling dataset and future inputs. It effectively minimizes accuracy loss with only a small number of inferences, ensuring minimal and one-time overhead.

Uniform quantization. Oaken adopts uniform quantization, where the scaling factor σ is calculated using only simple statistics to minimize hardware complexity:

$$\sigma = \frac{2^m - 1}{\text{Max} - \text{Min}}, \tag{2}$$

Where m is the bitwidth of the quantized value, and Max and Min represent the maximum and minimum of the values to be quantized. The uniform quantization function, which converts a value x into its quantized value, is defined using the scaling factor from Eq. 2:

$$Q(x) = \text{round}((x - \text{Min}) \times \sigma). \tag{3}$$

Oaken finds the minimum and maximum values for each of the three quantization groups and computes the quantization scaling factor for each group dynamically online.

Online KV cache quantization. As previously discussed, Oaken performs per-token quantization on the KV cache, focusing only on the key-value vector newly generated in each attention layer. Oaken dynamically separates KV cache values into three groups using four group thresholds obtained from offline profiling. It then retrieves the minimum and maximum values for each quantization group, calculates the scaling factor online, and quantizes the value within each group accordingly.

4.4 Group-Shift Quantization

Oaken's threshold-based online grouping effectively mitigates information loss by splitting values into three groups: *outer*, *middle*, and *inner* group. However, this approach poses a new challenge. When quantizing the outer group, whose values have large magnitudes, directly applying uniform quantization results in information loss. Previous works have addressed this by using mixed precision (e.g., FP16) for outliers, distinct from the precision used for inliers (e.g., INT4) [22, 30, 33, 86]. However, using mixed precision for outliers introduces sparsity, incurring a storage cost of 23 bits per entry: 16 bits for the value, 6 bits for the index, and 1 bit to indicate the group. Dealing with sparse, mixed-precision outliers also requires additional hardware modules, which adds complexity to the accelerator. While these overheads are negligible when the outlier fraction is small, they become substantial as the proportion increases. Moreover, this extra complexity makes it challenging to explore the sweet spot in the accuracy-performance trade-off by adjusting the group thresholds.

One straightforward way to address this issue is to quantize outliers as well. However, as discussed in Section 4.1, quantizing outer groups is challenging due to their wide magnitude range. To address this, we propose group-shift quantization, a hardware-efficient algorithm that minimizes quantization loss while compressing outliers to a lower bitwidth.

Group-shift algorithm. The core idea of Oaken's group-shift algorithm is to shift the entire group using the thresholds obtained from offline profiling to narrow the range of values, making low-bit quantization possible. For instance, for the outer group, we subtract T^o_{hi} from values larger than T^o_{hi}, and subtract T^o_{lo} from values smaller than T^o_{lo}. While the middle group corresponds to inliers, our group-shift algorithm can also be applied in the same manner. Figure 7(b) shows that applying the aforementioned method shifts the distribution of both the outer and middle groups, concentrating them within a narrower range. As a result, Oaken can quantize groups spanning wide ranges to low bitwidth using group-shift, minimizing the quantization loss. Note that the group-shift method

does not require additional information beyond those obtained from the offline profiling described in Section 4.3.

In summary, Oaken's quantization function $Q_o(x)$, which converts the value x to the quantized one, is defined as follows:

$$Q_o(x) = \begin{cases} Q(x - T^o_{hi}) & x \in G_o \text{ and } x > T^o_{hi} \\ Q(x - T^o_{lo}) & x \in G_o \text{ and } x < T^o_{lo} \\ Q(x - T^i_{hi}) & x \in G_m \text{ and } x > T^i_{hi} \\ Q(x - T^i_{lo}) & x \in G_m \text{ and } x < T^i_{lo} \\ Q(x) & x \in G_i \end{cases}, \quad (4)$$

where $Q(x)$ is a quantization function defined in Eq. 3. Oaken quantizes middle group into 4-bit, inner and outer groups into 5-bit.

4.5 Fused Dense-and-Sparse Encoding

Oaken employs a dense-and-sparse encoding strategy, as proposed in prior works, to efficiently store dense inliers and sparse outliers [22, 30]. In Oaken, the middle group, which consists of inliers and makes up the majority of the KV cache, is stored in a dense matrix. The outer and inner groups, which consist of outliers, are stored using a sparse matrix format, Coordinate List (COO) [16, 18, 72, 85], with the corresponding elements in the dense matrix being zeroed. COO format used in Oaken requires extra 6 bits to indicate the location of each value, along with 1 bit to denote the quantization group, and the bits used to represent the value for each entry.

To further reduce capacity overhead, we propose leveraging the zeroed elements in the dense matrices. These zeroed elements, corresponding to positions originally occupied by outliers in the KV cache, remain unused after separating the KV cache into dense and sparse matrices. We introduce a *fused dense-and-sparse encoding* method that repurposes these unused 4 bits to store part of the outliers, as illustrated in Figure 7(c). Specifically, four bits of the quantized 5-bit outliers are embedded in the zeroed elements of the dense matrix, while the remaining 6 index bits, 1 group bit, and 1 sign bit are stored in the sparse COO format.

Since the index bits in the COO format already indicate the location of outliers within the dense matrix, a dedicated flag to denote their presence is unnecessary. Moreover, with each entry in the sparse matrix fixed at 8 bits and memory-aligned, the memory management unit can efficiently handle both dense and sparse matrices on a page basis, the details of which will be described in Section 5.2. By combining our fused dense-and-sparse encoding strategy with group-shift quantization, we reduce the bitwidth of each outlier entry from 23 to just 8 bits, increasing the compression ratio of the KV cache, while keeping memory alignment.

5 Oaken's Accelerator Architecture

5.1 Architecture Overview

Figure 8 illustrates the overview of the proposed architecture, which mainly consists of compute cores, memory controllers, a host interface, and an interconnect. The compute cores are designed to support end-to-end LLM inference operations. The memory controllers handle device memory to read data, including model parameters, keys, and values, and are also responsible for writing keys and values back to memory. The host interface employs a PCIe-based connection to communicate with the host system. This interface

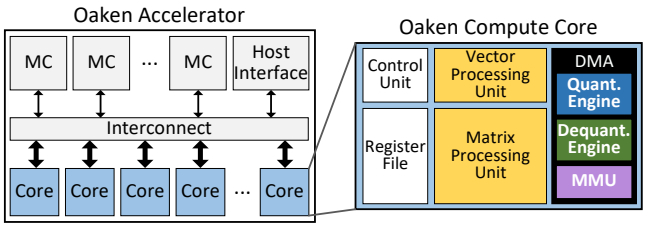


Figure 8: Overall Oaken accelerator architecture.

also manages the scheduling of incoming requests and distributing them across compute cores for efficient processing. An interconnect links these components and is optimized to maximize bandwidth utilization during memory read, ensuring efficient data transfer to the compute cores. This design enables the concurrent use of all memory controllers for reading model parameters and distributing them across the compute cores. On the other hand, memory writes from the compute cores are less frequent and involve smaller data sizes, reducing bandwidth consumption and simplifying the logic design without compromising performance.

5.2 Oaken Compute Core

Overall design. The main modules of the proposed accelerator are the compute cores, which are adapted from the architecture introduced in LPU [48] to enable end-to-end LLM inference. Each compute core consists of a Matrix Processing Unit (MPU) and Vector Processing Unit (VPU) designed to execute LLM inference operations token-by-token. These processing units are designed to maintain high utilization throughout the entire process while minimizing inefficient logic and ensuring low latency. MPU is designed to stream weight read from memory to perform efficient matrix-vector multiplication, while VPU handles element-wise operations between matrix-vector computations. The direct memory access (DMA) unit facilitates data transfer by reading weights from memory to feed the processing units and writing KV cache back to memory. This DMA unit also incorporates quantization/dequantization engines and a memory management unit (MMU), all of which are critical for implementing the proposed KV quantization technique.

Quantization engine. Figure 9(a) shows the quantization engine in DMA unit, designed to perform online KV cache quantization.

First, the ① **decomposer module** partitions incoming activations into three quantization groups based on outlier thresholds determined offline. It then performs group shift for the outer and middle groups by subtracting their streamed thresholds from the values of these groups. Finally, the middle group is directed to the inlier quantization path, while outliers, whether from inner or outer groups, are routed to the outlier quantization path, with zeros inserted in the alternate path accordingly.

Both the ② **inlier** and ③ **outlier quantizer modules** handle quantization for each key and value vector. Quantization scaling factors are dynamically computed during runtime, based on per-token min and max values for each group. After calculating the scaling factors, these two modules finally perform 4/5-bit uniform quantization. The quantized inlier and outlier values are merged using an OR gate and sent to the quantized dense matrix. The outlier

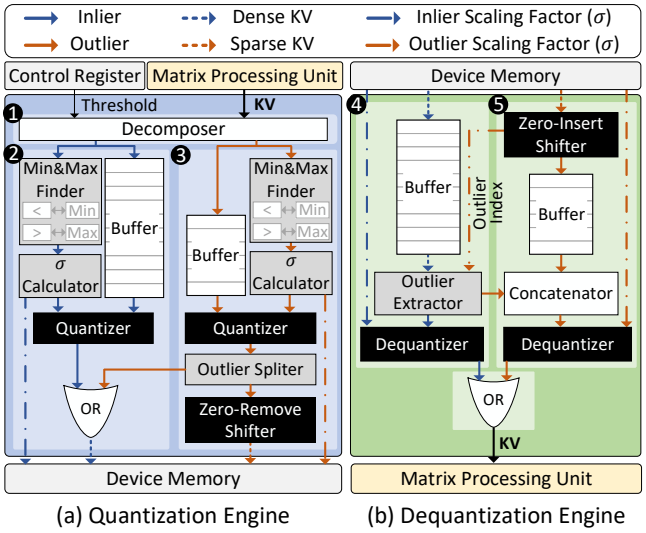


Figure 9: Quantization and dequantization engines of Oaken compute core.

module generates an index and a group flag for COO transformation. It employs a zero-remove shifter [57, 67] to implement fused dense-and-sparse encoding, optimizing memory usage.

Dequantization engine. The dequantization engine, illustrated in Figure 9(b), is also integrated into the DMA unit to dequantize the KV cache retrieved from memory.

The ④ **inlier dequantizer module** buffers incoming dense data to synchronize with sparse data processed by the outlier dequantizer module. The ⑤ **outlier dequantizer module** handles sparse COO data by performing a zero-insert [57, 67] operation to restore the original data alignment. It identifies the original positions of fused outliers using the index and group information of the sparse data and inserts the necessary zeros accordingly. Both dequantizer modules then restore the data, which is buffered to be aligned with the outputs of the counterpart module. Finally, the outputs from both dequantizer modules are merged via an OR gate and forwarded to the processing unit.

Since the dequantization engine does not require the entire KV cache for its operation, we designed it to function in a streaming manner. This design allows the dequantization engine to maintain low latency, while efficiently processing all past KV cache.

Memory management unit. Figure 10 illustrates the operations of memory management unit, which manages the reading and writing of quantized KV cache. We design the MMU unit to handle dense-and-sparse matrices in a page-based manner, optimizing bandwidth utilization. It supports multiple memory accesses in burst mode and streamlined operations to hide latency of memory and quantization/dequantization operations. Since Oaken's MMU units share a common address space, MMU operates in each compute core independently, preventing interference. Without this specialized MMU, processing variable-sized sparse matrices would require additional overhead for indexing, reshaping, and subsequent operations. There are two major challenges associated with the design of MMU unit:

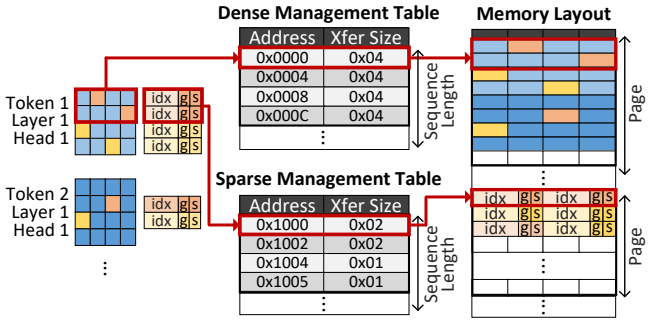


Figure 10: Operations of memory management unit (MMU) for handling dense and sparse data.

(1) Addresses and transfer size management: Dense matrices have predictable sizes that are well-aligned within memory spaces, while sparse matrices vary in size. Thus, management tables are needed for both dense and sparse data to accommodate this variability. These tables contain the virtual-to-physical address mappings and transfer sizes for the KV cache, considering up to the maximum sequence length per attention head. Physical addresses and transfer sizes are dynamically calculated during inference by checking available pages on demand.

(2) Read-write granularity and order determination: To maximize memory bandwidth utilization, burst access should be leveraged whenever possible to reduce the total number of memory transactions. Writing KV cache involves relatively small sizes, as it only includes the key-value for the current token, whereas reading requires retrieving the KV cache for all previous tokens. To address this, Oaken organizes KV cache for the current token in a layout that facilitates burst reads in subsequent operations. Key-value vectors generated in the current layer are divided by attention head and written to distinct pages, as explained in Section 5.2. When the KV cache for the next token is generated, it is divided similarly and written sequentially, immediately following the previous tokens' KV cache. This sequential arrangement allows that the KV cache for all previous tokens can be read in burst mode, allowing Oaken to efficiently utilize bandwidth.

5.3 Token-level Batch Scheduling

Efficient scheduling is crucial for Oaken to efficiently serve LLM inference. Each compute core in Oaken is optimized to process a single token efficiently. During the prefill phase, input tokens from each request are scheduled for parallel processing across multiple cores. However, in the generation phase, each core handles a single output token from one request, which reduce hardware utilization. For larger batches, Oaken improves overall core utilization by processing multiple requests in parallel.

Although the overhead for KV cache quantization and dequantization is minimal, Oaken further minimizes this by overlapping them with other operations. In batched inference, KV cache cannot be shared across cores because each core processes distinct requests, forcing each core to monopolize the memory bandwidth. Oaken employs a scheduling strategy that hides latency by overlapping KV quantization and dequantization with DMA reads and attention computations from other requests.

Table 1: Hardware specification of NVIDIA A100 GPU and Oaken equipped with either HBM or LPDDR memory.

	NVIDIA A100	Oaken-HBM	Oaken-LPDDR
Peak FP16 TFLOPS	312	270	270
Operating frequency	1.4 GHz	1.0 GHz	1.0 GHz
Memory type	HBM	HBM	LPDDR
Memory capacity	80 GB	80 GB	256 GB
Memory bandwidth	2.0 TB/s	2.0 TB/s	1.1 TB/s

6 Evaluation

6.1 Methodology

Models and datasets. For evaluation, we use Llama2-7B, 13B, and 70B [66], OPT-6.7B, 13B, and 30B [84], Mistral-7B [25] and Mixtral-8x7B [26] models. Llama2, Mistral, and Mixtral models implement grouped-query attention [3], while Mistral and Mixtral models also incorporate a sliding window [6]. Additionally, Mixtral model further integrates mixture-of-experts (MoE) layers [62]. To evaluate model accuracy, we utilize Wikitext2, PIQA, Winogrande, and Hellaswag datasets, which are widely evaluated in prior studies [22, 31, 33, 35, 41, 86, 87]. Wikitext2 [46] dataset consists of tokens extracted from Wikipedia articles, while PIQA [7], Winogrande [58], and Hellaswag [81] constitute questions and answers. We report zero-shot accuracy (%) for PIQA, Winogrande, and Hellaswag datasets and perplexity for Wikitext2 dataset. Note that for perplexity, lower values indicate better performance. For real-world benchmarking, we use two open-source production traces from Azure LLM inference services, *Conversation* [47, 54] and *BurstGPT* [68]. We follow the methodology established in prior work to simulate inference serving scenarios [20]. Requests are sampled from the trace over a time period, and batches are synthesized with varying input and output sequence lengths. We repeat this process across multiple batches, measuring the average performance.

Accelerator platforms. For the end-to-end performance evaluation, we developed a hardware simulator for the Oaken accelerator by extending the existing hardware simulator of LPU [21, 53]. LPU was initially optimized for low-latency inference without batching support, while the follow-up work scaled it to accommodate larger batches [48]. We extend the LPU architecture to integrate Oaken by incorporating quantization/dequantization engines and memory management units into the LPU's DMA units. For GPU baselines, we use NVIDIA A100 GPUs equipped with 80 GB HBM [49]. We use a single GPU for Llama2-7B, 13B, and Mistral-7B models, as it could accommodate all the model parameters. For larger models including OPT-30B, Mixtral-8x7B, and Llama2-70B models, we use two GPUs, employing pipeline parallelism to keep computation capability and memory bandwidth consistent, while scaling capacity to 160 GB.

Hardware specifications and implementation. Table 1 summarizes the specifications of the NVIDIA A100 GPU and the Oaken accelerator used in our evaluations. Oaken-LPDDR is equipped with an LPDDR memory module matching the specifications used in prior works [53, 83]. Oaken-HBM is configured with HBM memory identical to that of the A100 GPU. The HBM memory offers higher bandwidth but has a smaller capacity compared to LPDDR memory, making a trade-off between them. We implement the Oaken hardware in RTL using SystemVerilog and verify its functionality with Synopsys VCS functional verification solution. The RTL is synthesized using Synopsys Design Compiler for a target clock frequency of 1 GHz on TSMC 28nm technology.

Baselines. For accuracy evaluation, we use Tender [35], Atom [86], QServe [41], KIVI [43], and KVQuant [22] as baselines. For performance evaluation, we first use vLLM [32] as the FP16-operating GPU baseline, as it represents the state-of-the-art LLM serving system and delivers superior performance compared to other alternatives. We also use most of the baselines for accuracy evaluation, excluding Atom, which lacks open-source code availability. KIVI, QServe, and KVQuant serve as additional GPU baselines, running on A100 GPUs. Tender [35] is an LLM inference accelerator employing quantization, which offers an open-source simulator. For a fair comparison, we align Tender's memory specifications and compute capabilities with those of the A100 GPU. All quantization-based baselines employ 4-bit KV cache-only quantization. While QServe and Tender offer weight and activation quantization, we disable these features for fair comparison with the other baselines.

Thresholds. Throughout the evaluation, we set the outer, middle, and inner group ratio to 4%, 90%, and 6%, respectively. This global configuration applies to all models and datasets for the following two reasons. First, as discussed in Section 4.1, KV cache distribution is independent of the input dataset. Second, although the optimal group ratio varies slightly across LLMs, its impact on inference performance and accuracy is marginal. Section 6.2 explores the threshold search space and group count, justifying this choice.

Offline profiling. Oaken's offline profiling is performed by collecting topK values, which represent four boundaries of the quantization groups, during inference and averaging the gathered values. As mentioned, we use the same group ratio and the Wikitext2 dataset for all LLMs. However, since the KV cache distribution varies by model, requiring different group thresholds and individual profiling. Despite this, Oaken's offline profiling requires only about a hundred inferences and takes approximately ten minutes, even for the Llama2-70B model. Since this process is required only once before serving LLM inference online, the overhead is negligible.

6.2 Experimental Results

Throughput. Figure 11 presents the end-to-end throughput comparison results among GPU baselines (vLLM, KVQuant, KIVI, and QServe) and ASIC accelerators (Tender and Oaken). The results are omitted when the baseline system lacks support for the corresponding models. For a batch size of 256, Oaken-LPDDR achieves an average throughput improvement of 1.79× over vLLM and 1.58× over QServe. This improvement is attributed to the reduced execution time of the attention operations, which accounts for the majority of inference time. Oaken alleviates bandwidth and capacity bottlenecks by minimizing memory access to the KV cache.

GPU baselines perform well for small batches and models, but as the batch size grows, they cannot accommodate the entire batch due to capacity constraints, leading to performance saturation. Tender, which uses the same HBM memory as A100 GPUs, also does not scale for large batches. Oaken-HBM outperforms other baselines and Oaken-LPDDR for small models and batch sizes. However, it

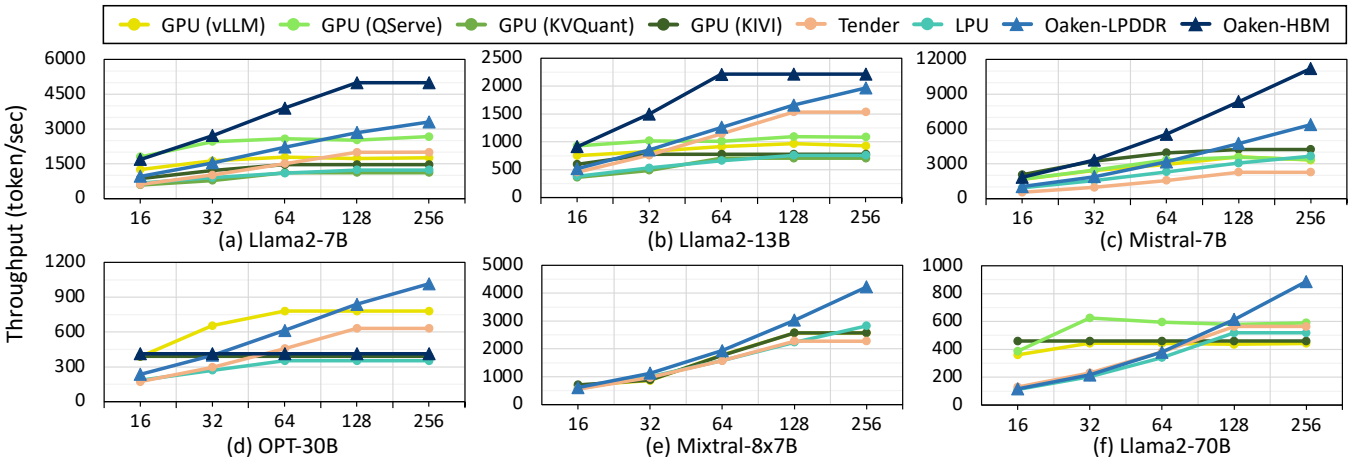


Figure 11: Throughput results of GPU baselines, LPU, Tender, and Oaken equipped with LPDDR and HBM across six LLMs. We sweep the batch size from 16 to 256. The input and output sequence lengths are set to 1K:1K.

Table 2: Perplexity results on Wikitext2, and zero-shot accuracy results on PIQA, Winogrande, and Hellaswag datasets with effective bitwidth of each quantization technique.

Model	Llama2			OPT			Mistral	Mixtral	Llama2			Llama2			Llama2			Llama2		
	7B	13B	70B	6.7B	13B	30B	7B	8x7B	7B	13B	70B	7B	13B	70B	7B	13B	70B	7B	13B	70B
Task	Wikitext2								PIQA			Winogrande			Hellaswag			-		
Metric	Perplexity (↓)								Accuracy (%)			Accuracy (%)			Accuracy (%)			Effective Bitwidth		
Original	**5.47**	**4.88**	**3.32**	**10.86**	**10.13**	**9.56**	**5.25**	**3.84**	**79.05**	**80.52**	**82.70**	**69.13**	**72.80**	**80.20**	**75.98**	**79.38**	**83.82**	**16.00**	**16.00**	**16.00**
KVQuant	5.49	4.94	3.33	10.88	10.14	9.58	5.33	3.87	78.35	79.33	82.21	67.80	71.74	77.98	75.82	79.25	83.70	4.82	4.81	5.01
KIVI	5.50	4.90	3.33	10.88	10.16	9.58	5.34	3.84	78.07	79.05	78.07	67.84	70.96	76.81	75.57	78.97	83.47	4.99	4.99	4.99
Tender	6.42	5.74	4.25	11.80	11.05	10.44	5.54	NaN	74.27	76.12	77.91	62.90	65.69	74.59	73.89	77.16	75.04	4.07	4.07	4.10
Atom	5.62	4.98	3.37	11.01	10.22	9.64	5.42	4.05	76.17	76.99	81.34	66.46	67.09	75.77	72.22	76.21	80.52	4.25	4.25	4.63
QServe	5.67	5.12	3.36	10.95	10.28	9.62	5.42	4.03	77.37	77.48	81.77	65.29	66.80	76.09	74.41	76.69	83.24	4.25	4.25	4.25
Oaken	**5.53**	**4.93**	**3.34**	**10.88**	**10.16**	**9.58**	**5.35**	**3.90**	**78.29**	**79.71**	**82.59**	**67.64**	**70.56**	**76.64**	**73.72**	**78.24**	**83.50**	**4.82**	**4.82**	**4.89**

faces challenges in accommodating large models such as Mixtral-8x7B and Llama2-70B, or handling large batches due to its insufficient memory capacity. Mistral-7B, Mixtral-8x7B, and Llama2-70B models employ grouped-query attention to reduce KV cache size, helping to alleviate bandwidth bottlenecks even without KV quantization. However, for larger batch sizes, capacity limitations still cause saturation, while Oaken accelerators offer scalability, demonstrating the effectiveness of our KV cache quantization technique.

Accuracy. Table 2 presents the accuracy results of each baselines across eight LLMs on Wikitext2, PIQA, Winogrande, and Hellaswag datasets, along with the effective bitwidth on the Llama2 models. Oaken exhibits an average accuracy loss of 0.87% compared to the original FP16 baseline, with 0.54% and 0.32% lower accuracy than KVQuant and KIVI, respectively, while achieving 1.38% higher accuracy than QServe. KVQuant and KIVI requires a larger effective bitwidth due to their use of sparse layout for outlier values and fine-grained grouping, respectively. They achieve higher accuracy than Oaken, but their advantages are largely offset by the prohibitively high overhead of online sorting and mixed-precision operations. On the other hand, Tender, Atom, and QServe employ an indirect indexing technique and a transformation matrix to reorder KV channels and group those with similar magnitudes. This approach impose minimal overhead due to their low effective bitwidth, as they do not require individual processing of outliers; however, this comes at the cost of larger accuracy losses, as they rely on coarse-grained per-group or per-channel quantization without considering exceptions in KV distribution, as discussed in Section 4.1.

Trade-off between accuracy and compression ratio. Figure 12(a) illustrates the trade-off between accuracy and compression ratio in Oaken's KV cache quantization on Llama2-7B model, which can also be interpreted as a trade-off between accuracy and performance. We sweep the group ratio and measure the perplexity on Wikitext2 dataset. The effective bits in Oaken are determined by the ratio of inner and outer groups, as their location is stored in the sparse matrix. All points on the same horizontal line share the same outlier ratio and effective bits, but differ in group composition. We select a ratio of 4%, 90%, and 6% for the outer, middle, and inner groups, respectively, throughout the entire evaluation, as they are one of the Pareto-optimal points highlighted as a light blue line. While using higher effective bits might improve accuracy even better than the current configuration, it negatively impacts the inference performance due to its low compression ratio.

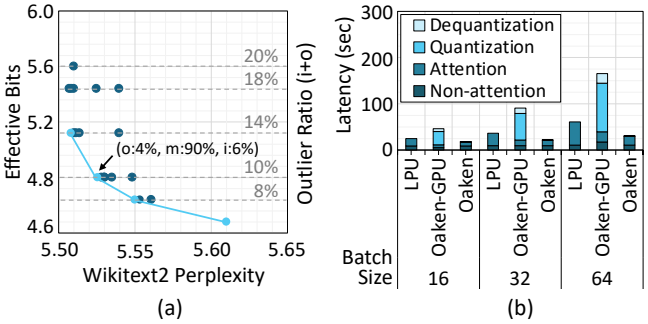


Figure 12: (a) Accuracy and effective bits with varying quantization group ratios. (b) Latency breakdown for non-attention, attention, quantization and dequantization operations using Llama2-7B model across varying batch size.

Table 3 presents the perplexity and effective bitwidth for different numbers of groups evaluated on the Llama2-7B model. We fix the total ratio of inner and outer groups at 10%. Oaken's fused dense-and-sparse encoding eliminates the need for a bit to represent outliers when using two groups. However, this disrupts memory alignment, as each sparse COO entry consists of 6 index bits and 1 sign bit. To mitigate hardware overhead, an extra padding bit is added, maintaining the same effective bitwidth. Using four or five groups improves accuracy but increases bitwidth, as 9-bit COO entries require two bits for inner and outer groups. This also misaligns memory layout, requiring additional padding. While using 4-bit outliers to keep 8-bit alignment preserves the effective bitwidth, it slightly reduces accuracy. In summary, Oaken's three-group quantization offers the optimal balance between cost and accuracy.

Latency breakdown. To better understand the impact of KV cache quantization on performance, we break down the end-to-end inference latency of LPU and Oaken-LPDDR as we vary the batch size. We also implement Oaken's quantization algorithm on GPU and measure its operation latencies. Figure 12(b) shows that the latency of attention operations increases proportionally with the batch size. While Oaken does not directly affect the execution time of non-attention operations, it alleviates bandwidth bottleneck through KV quantization. As a result, the execution time of attention operations is, on average, 55.0% shorter than that of LPU, contributing to a reduction in end-to-end latency. When the batch size is 64, quantization and dequantization account for only 1.29% and 3.23% of the entire latency, respectively. On the contrary, Oaken algorithm on GPU demonstrates long quantization and dequantization latencies due to warp divergence in CUDA, which is required to separate multiple quantization groups. Note that Oaken hides both quantization and dequantization latencies by overlapping them with other operations and processing them in a streaming manner.

Sensitivity to sequence length. Figure 13 shows throughput results when sweeping the total sequence length from 1K to 32K. For shorter sequence lengths below 8K, the proportion of compute-bound, batchable operations is larger than memory-bound, *non*-batchable attention operations. Therefore, the performance of QServe and vLLM outperform Oaken in this range by leveraging the higher parallelizable resources available on GPUs. However, as the sequence length increases, Oaken-HBM surpasses other baselines,

Table 3: Accuracy and effective bits using the Llama2-7B model with varying number of groups and group ratios while keeping the total inner and outer group ratio at 10%.

Group Ratio (o / m / i)	Outlier Bits	Effective Bitwidth	Wikitext2 Perplexity
4 / 90 / 6	**5**	**4.8**	**5.526**
90 / 10	5	4.8	5.804
10 / 90	5	4.8	5.546
4 / 90 / 3 / 3	5	5.6	5.523
2 / 2 / 90 / 6	5	5.6	5.516
2 / 2 / 90 / 3 / 3	5	5.6	5.516
4 / 90 / 3 / 3	4	4.8	5.572
2 / 2 / 90 / 6	4	4.8	5.531
2 / 2 / 90 / 3 / 3	4	4.8	5.532

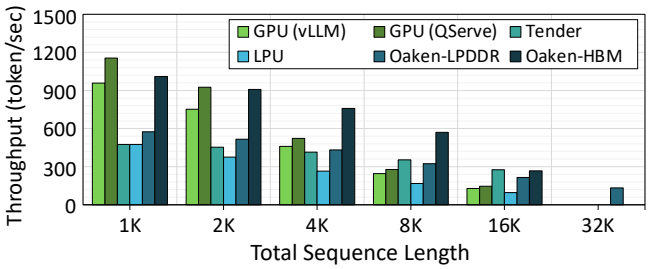


Figure 13: Throughput results on Llama2-13B model with a batch size of 16 when increasing total sequence length from 1K to 32K. The ratio of input and output length is set to 1:1.

including Oaken-LPDDR, with its high memory bandwidth and KV cache quantization. However, HBM-based systems including QServe and Oaken-HBM cannot handle sequences longer than 16K, making it difficult to complete the entire batch due to insufficient capacity. Oaken-LPDDR, on the other hand, can accommodate longer sequences of up to 32K by mitigating both bandwidth and capacity pressure through KV quantization and large-capacity memory.

Real-world benchmark. Figure 14 presents the generation throughput results for Llama2-13B and Mixtral-8x7B models, evaluated using two real-world LLM inference traces. We exclude Oaken-HBM and QServe for Mixtral-8x7B model, as Oaken-HBM's memory cannot accommodate the entire model and QServe lacks support for MoE layers. Tender, which employs systolic arrays, suffers underutilization due to the padding required by varying prompt lengths within a batch. *Conversation* trace features short output lengths, resulting in a brief generation phase. As the bandwidth bottleneck due to the KV cache is noticeable in generation phase, a short generation length reduces the advantage of Oaken's KV cache quantization. Conversely, *BurstGPT* trace features longer output lengths, where KV cache quantization in Oaken becomes more beneficial. Mixtral-8x7B model utilizes grouped query attention to reduce its KV cache size compared to multi-head attention. Quantization baselines, including Oaken-LPDDR and Tender, show little to no performance gain over full-precision baselines. However, as batch size increases or with the BurstGPT trace with longer generation lengths, Oaken-LPDDR demonstrates greater performance gains. In summary, Oaken delivers an advantage over existing solutions in real-world scenarios for the Llama2-13B and Mixtral-8x7B models.

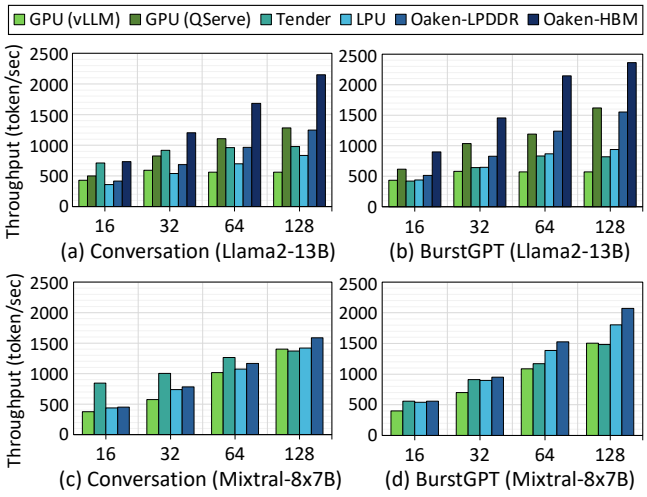


Figure 14: Real-world benchmark results for generation throughput on Llama2-13B and Mixtral-8x7B models evaluated with two LLM inference traces: *Conversation* [47] and *BurstGPT* [68]. We sweep the batch size from 16 to 128.

Area and power. As reported in Table 4, the quantization and dequantization engines account for a minor percentage of the *total compute core* area at 1.86% and 6.35%, respectively. In addition, the power consumption of the entire accelerator embedded with Oaken modules is 222.7W, which is 44.3% lower than the 400W TDP of the A100 GPU. These results clearly indicate that integrating Oaken's quantization and dequantization modules only imposes minimal hardware overhead while improving performance and achieving better energy efficiency compared to the GPU.

7 Related Work

LLM quantization. Most prior work on LLM quantization focuses on weight and/or activation quantization [17, 40, 42, 60, 69, 70] to reduce inference computation costs. RPTQ [78], SpinQuant [44], and QuaRot [5] introduce transformation matrices for weight and activation quantization. SmoothQuant [71] mitigates the quantization difficulty by transferring activation outlier scales to weights. SqueezeLLM [30] applies dense-and-sparse quantization for storing weight outliers in full precision. However, many existing approaches overlook the KV cache, whose size scales with sequence length and batch size, often becoming a major bottleneck for latency and throughput in batched LLM inference. Oaken overcomes this issue by employing an offline-online KV cache quantization algorithm with a customized hardware module, achieving high throughput with minimal accuracy degradation.

LLM inference accelerator. DFX [21] is one of the first LLM accelerators, which is designed to accelerate the entire GPT2 model using HBM and FPGA. CXL-PNM [53] introduces a LLM accelerator largely leveraging the DFX design, while employing LPDDR for striking a sweet spot in the bandwidth-capacity tradeoff space for large-scale LLM serving. LPU [48] is another LLM accelerator that differs from prior work in optimizing its design for minimal latency.

Table 4: Area overhead analysis of compression and decompression engines on TSMC 28nm.

Module	Area (mm2)	Area ratio (%)
Matrix processing unit	0.908	22.86
Vector processing unit	0.239	6.03
Quantization engine	0.074	1.86
Dequantization engine	0.252	6.35
Compute core	3.971	100

TransPIM [88] proposes a PIM accelerator targeted for encoder-based transformer models such as BERT. AttAcc [52], IANUS [59], and NeuPIMS [20] employ PIM technologies for decoder-based transformer LLM serving, while they impose capacity pressure on the large-batch long-sequence serving scenarios. Unlike these prior works, this work devises Oaken, which jointly employs KV quantization and LPDDR for unleashing larger capacity and increased bandwidth to enable fast and efficient LLM serving.

Acceleration for quantized model inference. There have been several prior works on accelerating quantized neural network inference [9, 24, 29, 38, 55, 56, 63, 65, 74, 82]. Mokey [80] and Olive [19] apply outlier-aware quantization methods to transformer-based LLMs. LUT-GEMM [51] proposes a lookup-table-based GPU kernel to eliminate dequantization overhead in quantized LLM inference. Tender [35] quantizes KV cache as well as weights and activations, but its accuracy loss is significant. In contrast, Oaken is an LLM inference acceleration solution optimized for KV cache quantization, offering a scalable solution while minimizing accuracy loss.

8 Conclusion

Batched LLM inference faces significant challenges from high memory bandwidth and capacity demands, exacerbated by the growing size of KV caches in modern LLMs that produce long-sequence outputs. This paper tackles this challenge by proposing an acceleration solution, Oaken, that jointly exploits (1) an offline-online hybrid KV cache compression technique and (2) custom hardware modules tailored for the proposed algorithm that can be integrated with LLM accelerators. Oaken effectively unlocks sufficient bandwidth and capacity, which would otherwise be unattainable, leading to significant throughput improvements with only marginal accuracy loss. These compelling advantages demonstrate that Oaken efficiently addresses the two primary bottlenecks of modern LLM serving.

Acknowledgments

We thank the anonymous reviewers for their comments and feedback. This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) (No.2018-0-00503, No.RS-2024-00459797, Development of ML compiler framework for on-device AI), IITP under the Graduate School of Artificial Intelligence Semiconductor (IITP-2025-RS-2023-00256472), and the National Research Foundation of Korea (NRF) (RS-2024-00342148), grant funded by the Korea government (MSIT). This work was also partly supported by HyperAccel.

References

[1] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. *arXiv preprint arXiv:2403.02310* (2024).

[2] Arash Ahmadian, Saurabh Dash, Hongyu Chen, Bharat Venkitesh, Zhen Stephen Gou, Phil Blunsom, Ahmet Üstün, and Sara Hooker. 2023. Intriguing Properties of Quantization at Scale. In *Thirty-seventh Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=IYe8j7Gy8f

[3] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. https://arxiv.org/abs/2305.13245

[4] Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernandez Abrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vlad Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, Guy Gur-Ari, Steven Hand, Hadi Hashemi, Le Hou, Joshua Howland, Andrea Hu, Jeffrey Hui, Jeremy Hurwitz, Michael Isard, Abe Ittycheriah, Matthew Jagielski, Wenhao Jia, Kathleen Kenealy, Maxim Krikun, Sneha Kudugunta, Chang Lan, Katherine Lee, Benjamin Lee, Eric Li, Music Li, Wei Li, YaGuang Li, Jian Li, Hyeontaek Lim, Hanzhao Lin, Zhongtao Liu, Frederick Liu, Marcello Maggioni, Aroma Mahendru, Joshua Maynez, Vedant Misra, Maysam Moussalem, Zachary Nado, John Nham, Eric Ni, Andrew Nystrom, Alicia Parrish, Marie Pellat, Martin Polacek, Alex Polozov, Reiner Pope, Siyuan Qiao, Emily Reif, Bryan Richter, Parker Riley, Alex Castro Ros, Aurko Roy, Brennan Saeta, Rajkumar Samuel, Renee Shelby, Ambrose Slone, Daniel Smilkov, David R. So, Daniel Sohn, Simon Tokumine, Dasha Valter, Vijay Vasudevan, Kiran Vodrahalli, Xuezhi Wang, Pidong Wang, Zirui Wang, Tao Wang, John Wieting, Yuhuai Wu, Kelvin Xu, Yunhan Xu, Linting Xue, Pengcheng Yin, Jiahui Yu, Qiao Zhang, Steven Zheng, Ce Zheng, Weikang Zhou, Denny Zhou, Slav Petrov, and Yonghui Wu. 2023. PaLM 2 Technical Report. *arXiv preprint arXiv:2305.10403* (2023).

[5] Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, 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). https://arxiv.org/abs/2404.00456

[6] Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The Long-Document Transformer. https://arxiv.org/abs/2004.05150

[7] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2019. PIQA: Reasoning about Physical Commonsense in Natural Language. *arXiv preprint arXiv:1911.11641* (2019).

[8] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. *arXiv preprint arXiv:2005.14165* (2020).

[9] S. Chang, Y. Li, M. Sun, R. Shi, H. H. So, X. Qian, Y. Wang, and X. Lin. 2021. Mix and Match: A Novel FPGA-Centric Deep Neural Network Quantization Framework. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 208–220. doi:10.1109/HPCA51647.2021.00027

[10] Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher De Sa. 2024. QuIP: 2-bit 'quantization of large language models with guarantees. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 196, 34 pages.

[11] Wenhua Cheng, Weiwei Zhang, Haihao Shen, Yiyang Cai, Xin He, and Kaokao Lv. 2023. Optimize Weight Rounding via Signed Gradient Descent for the Quantization of LLMs. *arXiv preprint arXiv:2309.05516* (2023).

[12] Zhoujun Cheng, Jungo Kasai, and Tao Yu. 2023. Batch Prompting: Efficient Inference with Large Language Model APIs. *arXiv preprint arXiv:2301.08721* (2023).

[13] Brian Chmiel, Ron Banner, Elad Hoffer, Hilla Ben Yaacov, and Daniel Soudry. 2024. Accurate Neural Training with 4-bit Matrix Multiplications at Standard Formats. https://arxiv.org/abs/2112.10769

[14] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. *arXiv preprint arXiv:2205.14135* (2022).

[15] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. *arXiv preprint arXiv:2208.07339* (2022).

[16] Axel Feldmann and Daniel Sanchez. 2023. Spatula: A Hardware Accelerator for Sparse Matrix Factorization. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '23)*. 91–104. doi:10.1145/3613424.3623783

[17] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. OPTQ: Accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*.

[18] Christina Giannoula, Ivan Fernandez, Juan Gómez Luna, Nectarios Koziris, Georgios Goumas, and Onur Mutlu. 2022. SparseP: Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-In-Memory Architectures. *Proc. ACM Meas. Anal. Comput. Syst.* 6, 1, Article 21 (feb 2022), 49 pages. doi:10.1145/3508041

[19] Cong Guo, Jiaming Tang, Weiming Hu, Jingwen Leng, Chen Zhang, Fan Yang, Yunxin Liu, Minyi Guo, and Yuhao Zhu. 2023. OliVe: Accelerating Large Language Models via Hardware-friendly Outlier-Victim Pair Quantization. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 3, 15 pages. doi:10.1145/3579371.3589038

[20] Guseul Heo, Sangyeop Lee, Jaehong Cho, Hyunmin Choi, Sanghyeon Lee, Hyungkyu Ham, Gwangsun Kim, Divya Mahajan, and Jongse Park. 2024. NeuPIMs: NPU-PIM Heterogeneous Acceleration for Batched LLM Inferencing. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 722–737. doi:10.1145/3620666.3651380

[21] Seongmin Hong, Seungjae Moon, Junsoo Kim, Sungjae Lee, Minsub Kim, Dongsoo Lee, and Joo-Young Kim. 2022. DFX: A Low-latency Multi-FPGA Appliance for Accelerating Transformer-based Text Generation. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 616–630. doi:10.1109/MICRO56248.2022.00051

[22] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. *arXiv preprint arXiv:2401.18079* (2024).

[23] Suyeon Hur, Seongmin Na, Dongup Kwon, Joonsung Kim, Andrew Boutros, Eriko Nurvitadhi, and Jangwoo Kim. 2023. A fast and flexible FPGA-based accelerator for natural language processing neural networks. *ACM Transactions on Architecture and Code Optimization* 20, 1 (2023), 1–24.

[24] Ahmet Inci, Siri Virupaksha, Aman Jain, Ting-Wu Chin, Venkata Thallam, Ruizhou Ding, and Diana Marculescu. 2023. QUIDAM: A Framework for Quantization-aware DNN Accelerator and Model Co-Exploration. *ACM Trans. Embed. Comput. Syst.* 22, 2, Article 33 (Jan. 2023), 21 pages. doi:10.1145/3555807

[25] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, 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 preprint arXiv:2310.06825* (2023).

[26] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of Experts. https://arxiv.org/abs/2401.04088

[27] Qing Jin, Jian Ren, Richard Zhuang, Sumant Hanumante, Zhengang Li, Zhiyu Chen, Yanzhi Wang, Kaiyuan Yang, and Sergey Tulyakov. 2022. F8Net: Fixed-Point 8-bit Only Multiplication for Network Quantization. In *International Conference on Learning Representations*. https://openreview.net/forum?id=_CfpJazzXT2

[28] Norman P. Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Cliff Young, Xiang Zhou, Zongwei Zhou, and David Patterson. 2023. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. *arXiv preprint arXiv:2304.01433* (2023).

[29] Ben Keller, Rangharajan Venkatesan, Steve Dai, Stephen G. Tell, Brian Zimmer, William J. Dally, C. Thomas Gray, and Brucek Khailany. 2022. A 17–95.6 TOPS/W Deep Learning Inference Accelerator with Per-Vector Scaled 4-bit Quantization for Transformers in 5nm. In *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. 16–17. doi:10.1109/VLSITechnologyandCir46769.2022.9830277

[30] Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. 2023. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629* (2023).

[31] Young Jin Kim, Rawn Henry, Raffy Fahim, and Hany Hassan Awadalla. 2023. Finequant: Unlocking efficiency with fine-grained weight-only quantization for llms. *arXiv preprint arXiv:2308.09723* (2023).

[32] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient

Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23)*. 611–626. doi:10.1145/3600006.3613165

[33] Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. 2024. OWQ: Outlier-Aware Weight Quantization for Efficient Fine-Tuning and Inference of Large Language Models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 13355–13364.

[34] Janghwan Lee, Minsoo Kim, Seungcheol Baek, Seok Hwang, Wonyong Sung, and Jungwook Choi. 2023. Enhancing Computation Efficiency in Large Language Models through Weight and Activation Quantization. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. doi:10.18653/v1/2023.emnlp-main.910

[35] Jungi Lee, Wonbeom Lee, and Jaewoong Sim. 2024. Tender: Accelerating Large Language Models via Tensor Decomposition and Runtime Requantization. In *Proceedings of the 51st Annual International Symposium on Computer Architecture*.

[36] Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. 2023. Accelerating Distributed MoE Training and Inference with Lina. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. 945–959.

[37] Liang Li, Qingyuan Li, Bo Zhang, and Xiangxiang Chu. 2024. Norm Tweaking: High-Performance Low-Bit Quantization of Large Language Models. *Proceedings of the AAAI Conference on Artificial Intelligence* (Mar. 2024), 18536–18544.

[38] Wenjie Li, Aokun Hu, Ningyi Xu, and Guanghui He. 2024. Quantization and Hardware Architecture Co-Design for Matrix-Vector Multiplications of Large Language Models. *IEEE Transactions on Circuits and Systems I: Regular Papers* (2024).

[39] Haokun Lin, Haobo Xu, Yichen Wu, Jingzhi Cui, Yingtao Zhang, Linzhan Mou, Linqi Song, Zhenan Sun, and Ying Wei. 2024. DuQuant: Distributing Outliers via Dual Transformation Makes Stronger Quantized LLMs. *arXiv preprint arXiv:2406.01721* (2024). https://arxiv.org/abs/2406.01721

[40] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. 2023. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978* (2023).

[41] Yujun Lin*, Haotian Tang*, Shang Yang*, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. 2024. QServe: W4A8KV4 Quantization and System Co-design for Efficient LLM Serving. *arXiv preprint arXiv:2405.04532* (2024).

[42] Jing Liu, Ruihao Gong, Xiuying Wei, Zhiwei Dong, Jianfei Cai, and Bohan Zhuang. 2023. Qllm: Accurate and efficient low-bitwidth quantization for large language models. *arXiv preprint arXiv:2310.08041* (2023).

[43] 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 preprint arXiv:2402.02750* (2024).

[44] Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. 2025. SpinQuant: LLM quantization with learned rotations. (2025). arXiv:2405.16406 [cs.LG] https://arxiv.org/abs/2405.16406

[45] Eitan Medina and Eran Dagan. 2020. Habana Labs Purpose-Built AI Inference and Training Processor Architectures: Scaling AI Training Systems Using Standard Ethernet With Gaudi Processor. *IEEE Micro* 40 (2020), 17–24.

[46] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer Sentinel Mixture Models. *arXiv preprint arXiv:1609.07843* (2016).

[47] Microsoft. 2024. AzurePublicDataset. https://github.com/Azure/AzurePublicDataset.

[48] Seungjae Moon, Junsoo Kim, Jung-Hoon Kim, Junseo Cha, Gyubin Choi, Seongmin Hong, and Joo-Young Kim. 2023. HyperAccel Latency Processing Unit (LPUTM) Accelerating Hyperscale Models for Generative AI . In *2023 IEEE Hot Chips 35 Symposium (HCS)*. IEEE Computer Society, Los Alamitos, CA, USA, 1–1. doi:10.1109/HCS59251.2023.10254693

[49] NVIDIA. 2020. NVIDIA A100 Tensor Core GPU Architecture. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf.

[50] NVIDIA. 2023. NVIDIA TensorRT-LLM. https://github.com/NVIDIA/TensorRT-LLM.

[51] Gunho Park, Baeseong Park, Minsub Kim, Sungjae Lee, Jeonghoon Kim, Beomseok Kwon, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. 2024. LUT-GEMM: Quantized Matrix Multiplication based on LUTs for Efficient Inference in Large-Scale Generative Language Models. In *ICLR*. https://openreview.net/forum?id=gLARhFLE0F

[52] Jaehyun Park, Jaewan Choi, Kwanhee Kyung, Michael Jaemin Kim, Yongsuk Kwon, Nam Sung Kim, and Jung Ho Ahn. 2024. AttAcc! Unleashing the Power of PIM for Batched Transformer-based Generative Model Inference. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 103–119. doi:10.1145/3620665.3640422

[53] Sang-Soo Park, KyungSoo Kim, Jinin So, Jin Jung, Jonggeon Lee, Kyoungwan Woo, Nayeon Kim, Younghyun Lee, Hyungyo Kim, Yongsuk Kwon, Jinhyun Kim, Jieun Lee, YeonGon Cho, Yongmin Tai, Jeonghyeon Cho, Hoyoung Song, Jung Ho Ahn, and Nam Sung Kim. 2024. An LPDDR-based CXL-PNM Platform for TCO-efficient Inference of Transformer-based Large Language Models. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 970–982.

[54] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Inigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient generative LLM inference using phase splitting. In *ISCA*. https://www.microsoft.com/en-us/research/publication/splitwise-efficient-generative-llm-inference-using-phase-splitting

[55] Yubin Qin, Yang Wang, Dazheng Deng, Zhiren Zhao, Xiaolong Yang, Leibo Liu, Shaojun Wei, Yang Hu, and Shouyi Yin. 2023. FACT: FFN-attention Co-optimized transformer architecture with eager correlation prediction. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–14.

[56] Enrico Reggiani, Alessandro Pappalardo, Max Doblas, Miquel Moreto, Mauro Olivieri, Osman Sabri Unsal, and Adrián Cristal. 2023. Mix-GEMM: An efficient HW-SW Architecture for Mixed-Precision Quantized Deep Neural Networks Inference on Edge Devices. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 1085–1098. doi:10.1109/HPCA56546.2023.10071076

[57] Minsoo Rhu, Mike O'Connor, Niladrish Chatterjee, Jeff Pool, Youngeun Kwon, and Stephen W Keckler. 2018. Compressing DMA engine: Leveraging activation sparsity for training deep neural networks. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 78–91.

[58] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. WinoGrande: An Adversarial Winograd Schema Challenge at Scale. *arXiv preprint arXiv:1907.10641* (2019).

[59] Minseok Seo, Xuan Truong Nguyen, Seok Joong Hwang, Yongkee Kwon, Guhyun Kim, Chanwook Park, Ilkon Kim, Jaehan Park, Jeongbin Kim, Woojae Shin, Jongsoon Won, Haerang Choi, Kyuyoung Kim, Daehan Kwon, Chunseok Jeong, Sangheon Lee, Yongseok Choi, Wooseok Byun, Seungcheol Baek, Hyuk-Jae Lee, and John Kim. 2024. IANUS: Integrated Accelerator based on NPU-PIM Unified Memory System. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 545–560. doi:10.1145/3620666.3651324

[60] Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2023. Omniquant: Omnidirectionally calibrated quantization for large language models. *arXiv preprint arXiv:2308.13137* (2023).

[61] Noam Shazeer. 2019. Fast Transformer Decoding: One Write-Head is All You Need. https://arxiv.org/abs/1911.02150

[62] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. https://arxiv.org/abs/1701.06538

[63] Xuan Shen, Peiyan Dong, Lei Lu, Zhenglun Kong, Zhengang Li, Ming Lin, Chao Wu, and Yanzhi Wang. 2024. Agile-Quant: Activation-Guided Quantization for Faster Inference of LLMs on the Edge. *Proceedings of the AAAI Conference on Artificial Intelligence* (2024).

[64] 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.

[65] Zhuoran Song, Bangqi Fu, Feiyang Wu, Zhaoming Jiang, Li Jiang, Naifeng Jing, and Xiaoyao Liang. 2020. DRQ: Dynamic Region-based Quantization for Deep Neural Network Acceleration. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 1010–1021. doi:10.1109/ISCA45697.2020.00086

[66] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv preprint arXiv:2307.09288* (2023).

[67] Hanrui Wang, Zhekai Zhang, and Song Han. 2021. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 97–110.

[68] Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Zhenheng Tang, Xin He, Rui Guo, Xin Wang, Qiang Wang, Amelie Chi Zhou, and Xiaowen Chu. 2024. BurstGPT: A Real-world Workload Dataset to Optimize LLM Serving Systems. arXiv:2401.17644 [cs.DC] https://arxiv.org/abs/2401.17644

[69] Xiuying Wei, Yunchen Zhang, Yuhang Li, Xiangguo Zhang, Ruihao Gong, Jinyang Guo, and Xianglong Liu. 2023. Outlier suppression+: Accurate quantization of large language models by equivalent and optimal shifting and scaling. *arXiv preprint arXiv:2304.09145* (2023).

[70] Xiuying Wei, Yunchen Zhang, Xiangguo Zhang, Ruihao Gong, Shanghang Zhang, Qi Zhang, Fengwei Yu, and Xianglong Liu. 2022. Outlier suppression: Pushing the limit of low-bit transformer language models. *Advances in Neural Information Processing Systems* 35 (2022), 17402–17414.

[71] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*. PMLR, 38087–38099.

[72] Xinfeng Xie, Zheng Liang, Peng Gu, Abanti Basak, Lei Deng, Ling Liang, Xing Hu, and Yuan Xie. 2021. SpaceA: Sparse Matrix Vector Multiplication on Processing-in-Memory Accelerator. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 570–583. doi:10.1109/HPCA51647.2021.00055

[73] Yuhui Xu, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhengsu Chen, Xiaopeng Zhang, and Qi Tian. 2023. QA-LoRA: Quantization-Aware Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2309.14717* (2023).

[74] Jianxun Yang, Fengbin Tu, Yixuan Li, Yiqi Wang, Leibo Liu, Shaojun Wei, and Shouyi Yin. 2022. GQNA: Generic Quantized DNN Accelerator With Weight-Repetition-Aware Activation Aggregating. *IEEE Transactions on Circuits and Systems I: Regular Papers* (2022).

[75] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers. In *Advances in Neural Information Processing Systems*.

[76] Yiran Ding and Li Lyna Zhang and Chengruidong Zhang and Yuanyuan Xu and Ning Shang and Jiahang Xu and Fan Yang and Mao Yang. 2024. LongRoPE: Extending LLM Context Window Beyond 2 Million Tokens. In *ICML*.

[77] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 521–538.

[78] Zhihang Yuan, Lin Niu, Jiawei Liu, Wenyu Liu, Xinggang Wang, Yuzhang Shang, Guangyu Sun, Qiang Wu, Jiaxiang Wu, and Bingzhe Wu. 2023. Rptq: Reorder-based post-training quantization for large language models. *arXiv preprint arXiv:2304.01089* (2023).

[79] A. Zadeh, I. Edo, O. Awad, and A. 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)*. 811–824. doi:10.1109/MICRO50266.2020.00071

[80] Ali Hadi Zadeh, Mostafa Mahmoud, Ameer Abdelhadi, and Andreas Moshovos. 2022. Mokey: Enabling narrow fixed-point inference for out-of-the-box floating-point transformer models. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 888–901.

[81] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a Machine Really Finish Your Sentence? *arXiv preprint arXiv:1905.07830* (2019).

[82] Shulin Zeng, Jun Liu, Guohao Dai, Xinhao Yang, Tianyu Fu, Hongyi Wang, Wenheng Ma, Hanbo Sun, Shiyao Li, Zixiao Huang, Yadong Dai, Jintao Li, Zehao Wang, Ruoyu Zhang, Kairui Wen, Xuefei Ning, and Yu Wang. 2024. FlightLLM: Efficient Large Language Model Inference with a Complete Mapping Flow on FPGA. *arXiv preprint arXiv:2401.03868* (2024).

[83] Hengrui Zhang, August Ning, Rohan Baskar Prabhakar, and David Wentzlaff. 2024. LLMCompass: Enabling Efficient Hardware Design for Large Language Model Inference. In *Proceedings of the 51st Annual International Symposium on Computer Architecture*.

[84] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle 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 preprint arXiv:2205.01068* (2022).

[85] Zhekai Zhang, Hanrui Wang, Song Han, and William J. Dally. 2020. SpArch: Efficient Architecture for Sparse Matrix Multiplication. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 261–274. doi:10.1109/HPCA47549.2020.00030

[86] Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. 2024. Atom: Low-bit Quantization for Efficient and Accurate LLM Serving. *arXiv preprint arXiv:2310.19102* (2024).

[87] 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).

[88] 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)*. 1071–1085. doi:10.1109/HPCA53966.2022.00082