# Adana: Accelerating Large Language Models via Adaptive Nonuniform Asymmetric Quantization

Xinkuang Geng[1], Siting Liu[2*], Hui Wang[1], Leibo Liu[3], Jie Han[4], Honglan Jiang[1*]

[1] School of Integrated Circuits, Shanghai Jiao Tong University, Shanghai, China
[2] School of Information Science and Technology, ShanghaiTech University, Shanghai, China
[3] School of Integrated Circuits, Tsinghua University, Beijing, China
[4] Department of Electrical and Computer Engineering, University of Alberta, Alberta, Canada
xinkuang@sjtu.edu.cn, liust@shanghaitech.edu.cn, ihuiwang@sjtu.edu.cn, liulb@tsinghua.edu.cn, jhan8@ualberta.ca, honglan@sjtu.edu.cn

## Abstract

We propose Adana, a hardware-software co-design that enables efficient low-bit group-wise quantization for LLMs based on the adaptive nonuniform asymmetric numeric type. First, Adana introduces a novel numeric type that precisely captures the nonuniformity and asymmetry of data within small groups. In addition, an approximate metric for quantization error is proposed to facilitate efficient implementation of online adaptive activation quantization. Finally, a dedicated LLM acceleration microarchitecture is developed for Adana. Compared to state-of-the-art designs, Adana achieves 1.42×–2.10× speedups and 18.9%–48.5% power savings on LLMs, while maintaining superior accuracy.

## Keywords

large language model, low-bit quantization, group-wise quantization, adaptive numeric type, accelerator

**ACM Reference Format:**
Xinkuang Geng[1], Siting Liu[2*], Hui Wang[1], Leibo Liu[3], Jie Han[4], Honglan Jiang[1*]. 2026. Adana: Accelerating Large Language Models via Adaptive Nonuniform Asymmetric Quantization. In *63rd ACM/IEEE Design Automation Conference (DAC '26), July 26–29, 2026, Long Beach, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3770743.3804291

## 1 Introduction

Large Language Models (LLMs) such as OPT [40], LLaMA [9, 30, 31] and Qwen [35] have achieved unprecedented success across a wide range of natural language processing tasks [2, 4, 22, 24, 39]. In general, increasing parameter sizes leads to stronger model capability [15]. However, a large model imposes substantial demands on memory and computation, which becomes especially challenging on edge or portable devices with limited storage and power budgets [8, 18, 25, 33]. As a structured compression technique, quantization proportionally reduces memory usage [21]. In addition, when both weights and activations are quantized, most computations during inference can be performed in low-bit numeric formats. Since activation–activation matrix multiplications also occur during LLM inference [32], employing unified low-bit hardware inevitably requires quantizing activations as well, underscoring the importance of online quantization [7, 10, 17, 34]. Given the long-tailed data distribution in LLMs [5, 6, 38], group-wise quantization assigns a scaling factor to each group, effectively isolating outliers and preventing them from degrading the quantization resolution of the remaining values [3, 12, 16, 23]. Compared to tensor-wise or channel-wise quantization schemes, this strategy significantly reduces quantization error, especially at low bit widths.

**Table 1: Comparison of adaptive numeric types.**

| Property | ANT | M-ANT | BitMod | Adana |
|---|---|---|---|---|
| weight precision | 4/8 | 4 | 3/4 | 3/4 |
| activation precision | 4/8 | 4/8 | 16 | 3/4 |
| group size | – | 64 | 128 | 16–128 |
| nonuniformity | medium | high | medium | medium |
| asymmetry | low | low | medium | high |
| adaptation method | search | mapping | search | rounding |
| quantization efficiency | low | medium | low | high |

Group-wise quantization increases the diversity of statistical characteristics within each group. To address this, adaptive numeric types have been proposed [3, 12]. They assign different sub numeric types to each group based on its distribution, thereby reducing quantization errors. However, existing adaptive methods still have several limitations, as shown in Table 1. First, they overlook small groups and fail to exploit their data distributions; notably, they do not adapt well to asymmetric data that we find to be common in group-wise quantization [3, 11, 12]. Second, they insufficiently address low-bit activation quantization and do not provide efficient online quantization units.

To tackle these issues, we propose Adana, a hardware-software co-design featuring adaptive nonuniform and asymmetric quantization. Our main contributions are summarized as follows.

- Based on the analysis of the distributional characteristics of small groups in LLMs, we introduce the Adana numeric type, which better captures their inherent nonuniformity and asymmetry.
- We propose the Adana quantization framework with a focus on optimizing the online adaptive quantization unit.
- We design the Adana microarchitecture to accelerate LLMs through low-bit group-wise quantization.
- Adana is evaluated on multiple LLMs in terms of accuracy and hardware performance, demonstrating higher accuracy along with lower latency and energy consumption compared to existing low-bit group-wise quantization accelerators.

## 2 Background and Motivation

### 2.1 Quantization Basics

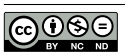
Let $\mathbf{x_f} \in \mathbb{R}^G$ denote a vector consisting of $G$ elements. Quantization can be viewed as approximating $\mathbf{x_f}$ using discrete values belonging to a numeric type $T$. This projection can be written as

$$\mathbf{x_q} = q(\mathbf{x_f}, T) = \arg\min_{\mathbf{t}} \|\mathbf{t} - \frac{\mathbf{x_f}}{\Delta}\|, \quad \mathbf{t} \in T^G, \tag{1}$$

where $\Delta$ is a shared scaling factor that maps all elements of $\mathbf{x_f}$ into the representable range of $T$. The corresponding dequantization process simply applies the inverse scaling as

$$\mathbf{x_{qf}} = \Delta \cdot \mathbf{x_q} \approx \mathbf{x_f}. \tag{2}$$

*Corresponding authors: Siting Liu, Honglan Jiang.
This work was supported in part by the National Natural Science Foundation of China under Grant 62374108 and in part by the Natural Sciences and Engineering Research Council of Canada under Grant RES0048688, Grant RES0051374, and Grant RES0054326.




*DAC '26, Long Beach, CA, USA*

ACM ISBN 979-8-4007-2254-7/2026/07
https://doi.org/10.1145/3770743.3804291

In LLM inference, the dominant computation is matrix multiplication, which consists of vector dot products [7, 33]. An important observation is that dequantization does not need to occur element-wise prior to the dot product. Because all values within the vector share the same scale, the scaling can be deferred until the dot product is completed. Formally, the dot product of an activation vector $\mathbf{a_f}$ and a weight vector $\mathbf{w_f}$ can be reformulated as

$$\mathbf{a_f} \cdot \mathbf{w_f} \approx \mathbf{a_{qf}} \cdot \mathbf{w_{qf}} = \Delta_a \cdot \Delta_w \cdot \left(\mathbf{a_q} \cdot \mathbf{w_q}\right), \tag{3}$$

where $\Delta_a$ and $\Delta_w$ are the scaling factors for $\mathbf{a_f}$ and $\mathbf{w_f}$, respectively. Thus, all intermediate arithmetic remains within the domain of $T$, while the dequantization is applied only once. This property is key to enabling efficient low-bit hardware acceleration.

## 2.2 Adaptive Numeric Types

An adaptive numeric type $\mathbf{T}$ provides $M$ selectable sub numeric types $\{T_0, T_1, \ldots\}$ that can be adaptively selected by each quantization group. ANT [11] selects four sub numeric types by minimizing the quantization error that is measured as the Euclidean distance between the original floating-point vector and its dequantized reconstruction, i.e.,

$$\hat{T} = \arg\min_{T_m} \|\mathbf{x_f} - \Delta \cdot q(\mathbf{x_f}, T_m)\|_2, \quad T_m \in \mathbf{T}, \tag{4}$$

This indicates that ANT chooses the sub numeric type that yields the lowest mean squared error (MSE). M-ANT [12] adopts 16 mathematically derived sub numeric types, each tailored to a specific variance range. Rather than explicitly computing the MSE, M-ANT matches the variance of the normalized data to pre-defined intervals as

$$\hat{T} = T_m \ \text{ s.t. } \ \mathrm{Var} \circ \mathrm{norm}(\mathbf{x_f}) \in [b_m, b_{m+1}), \quad T_m \in \mathbf{T}, \tag{5}$$

which reduces the computational overhead and enables efficient online quantization. BitMod [3] exploits redundant zero encodings to introduce four sub numeric types on top of the FP format. Its selection strategy follows the same MSE-based criterion as ANT.

## 2.3 Motivation

To analyze the data characteristics of LLMs, we sample weight values from the first layer of Llama-3.2-1B [9] using different group sizes (ranging from 2048 to 16). Figure 1 depicts eight data distribution plots for each group size. When the group size is large, the statistics of each group tend to form a stable, zero-mean Gaussian distribution, which aligns with the Central Limit Theorem (CLT). This property explains why channel-wise quantization with $G = 2048$ or coarse-grained group-wise quantization with $G > 32$ mainly requires the numeric type to adapt to the nonuniformity of the data distribution. However, as the group becomes smaller ($G \leq 32$), the CLT approximation no longer holds. The resulting distributions diverge significantly and exhibit two prominent forms of asymmetry: ① asymmetric data ranges, and ② biased distribution peaks.

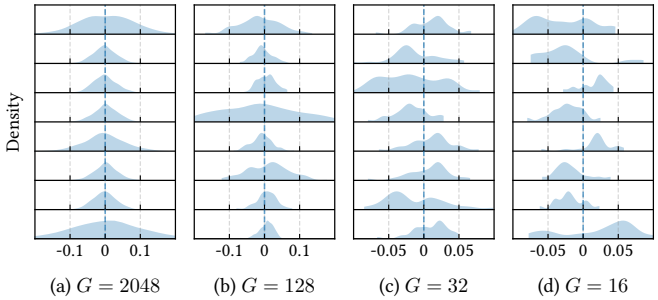



**Figure 1: Data distribution characteristics across different group sizes.**

However, existing quantization methods implicitly assume that the data distribution within each group is roughly symmetric, and therefore adapt only to the nonuniformity of the data. For example, both ANT and M-ANT employ strictly symmetric sub numeric types whose distribution peaks are fixed at zero. As shown in Figure 2, this constraint prevents them from capturing the asymmetric characteristics of small-group data, leading to substantial quantization errors. Although BitMod improves the alignment between the FP format and the empirical distribution, the inherent symmetry of the original quantization points (shown in grey) still prevents BitMod from accommodating diverse asymmetric ranges or biased peaks, resulting in a non-negligible mismatch in practice.

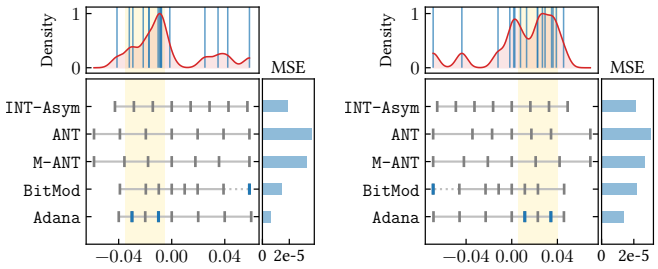



**Figure 2: Comparison of quantization using adaptive numeric types.**

Although introducing a zero point to plain INT enables INT-Asym to better match asymmetric data ranges, its uniform grid lacks the ability to assign a finer quantization resolution around the peak region of the data distribution, as shown in Figure 2. Existing adaptive numeric types also fall short, because they cannot incorporate a zero point to capture asymmetry, for two main reasons. First, when the entire quantization grid is biased to align with the data range, the high-resolution region remains fixed at the grid center. This is unsuitable when the actual peak lies closer to one side. Second, introducing a zero point adds a bias to both operands in arithmetic operations, which prevents the use of shift-based multiplication techniques in ANT, M-ANT, and BitMod.

To address this limitation, we propose an adaptive nonuniform asymmetric numeric type, Adana, which simultaneously captures the two key distribution characteristics of small-group data. Adana employs a unified and carefully designed rule to generate sub numeric types that adapt to diverse distribution patterns, while maintaining high computation and quantization efficiency. As shown in Figure 2, Adana effectively captures both the asymmetric range and the peak position for small-group data with different characteristics, achieving the lowest quantization error.

# 3 Adana Numeric Type

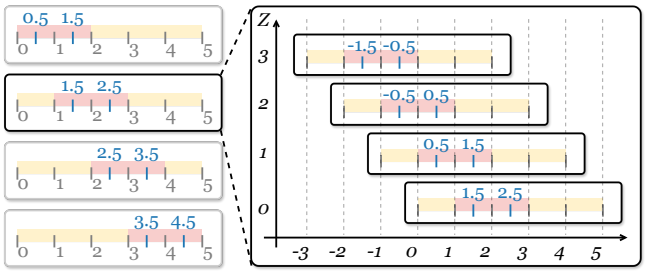
## 3.1 Formulation of the Adana Numeric Type

Since introducing a zero point can effectively align with asymmetric data ranges, we focus on designing an adaptive high-resolution grid independent of the numeric range, while a base grid maintains invariance with respect to bias operations. We construct $M$ sub numeric types as

$$T_{\text{Adana}-m} = \{0, \ldots, N_{\text{int}}-1\} \cup \{\, m + f + 0.5 \mid f = 0, \ldots, N_{\text{frac}}-1 \,\}, \tag{6}$$

where $N_{\text{int}}$ specifies the number of integer points in the base grid, and $N_{\text{frac}}$ specifies the number of half-step points that form the movable high-resolution grid. The index $m$ (ranging from 0 to $M-1$) of each sub numeric type determines the starting location of the high-resolution grid with respect to the base grid.

Figure 3 shows a 3-bit Adana numeric type with $(M, N_{\text{int}}, N_{\text{frac}})$ being $(4, 6, 2)$. It can be seen that the high-resolution grid is decoupled from the base grid and can be placed freely. Consequently, biasing the high-resolution grid (see Figure 3 (a)) to track the peak positions and biasing the base grid (see Figure 3 (b)) to match asymmetric data ranges become two independent adaptive mechanisms.




(a) Adapt to biased peak positions (b) Adapt to asymmetric ranges

**Figure 3: Overview of the formulation of the 3-bit Adana numeric type.**

For a given quantization bit width $B$, the total number of available quantization points is fixed to $2^B$. In addition, an implicit requirement is that the high-resolution grid must be able to sweep across the entire base grid, as shown in Figure 3 (a). Consequently, the three parameters of Adana are inherently coupled, satisfying

$$N_{\text{int}} + N_{\text{frac}} = 2^B \quad \text{and} \quad N_{\text{int}} = N_{\text{frac}} + M. \tag{7}$$

This indicates that once the bit width $B$ is specified, both $N_{\text{int}}$ and $N_{\text{frac}}$ can be determined by the number of sub numeric types $M$. To balance the relative coverage of the high-resolution grid within the base grid, we set $M = 2^{B-2}$, which uniquely defines the Adana numeric type for each bit width. For example, $(M, N_{\text{int}}, N_{\text{frac}})$ for 3-bit and 4-bit Adana numeric types are $(4, 6, 2)$ and $(8, 12, 4)$, respectively.

## 3.2 Mapping

To map the binary encoding to the Adana numeric type, a direct and codec–friendly rule is devised, i.e., the encodings smaller than $N_{\text{int}}$ are directly mapped to the integer points with the same value, while the remaining encodings are mapped sequentially to the $N_{\text{frac}}$ half-step points. When the zero point is taken into account, the decoding procedure can be formulated as shown in Algorithm 1. We first determine whether the encoding $x$ corresponds to an integer grid point or a half–step grid point by examining the difference $f$. If $f$ is negative, the encoding belongs to the integer grid, and the raw decoded value $u$ is the integer represented by $x$. Otherwise, $f$ indexes into the half-step grid, and we compute $u$ as per (6). Finally, we add the zero point $z$ to the raw decoded value $u$ to obtain the final decoded value.

**Algorithm 1** Decoding procedure for the Adana numeric type

**Input:** Encoded value $x$; group-wise zero point $z$, Adana mode $m$; $N_{\text{int}}$.
**Output:** Decoded value.

1: **function** DECODE $(x; z, m; N_{\text{int}})$
2: $f \leftarrow x - N_{\text{int}}$ ▷ *Determine point type*
3: **if** $f < 0$ **then** $u \leftarrow x$ ▷ *Integer point: forward integer encoding*
4: **else** $u \leftarrow m + f + 0.5$ ▷ *Half-step point: add mode offset and half-step*
5: **return** $u - z$

# 4 Adana Quantization Framework

## 4.1 Offline Weight Quantization

Since weight quantization can be performed offline, we use the same MSE-based criterion as (4) to search for the most suitable Adana sub numeric type for each weight group, and then quantize each element by selecting the closest discrete point within that sub numeric type, as in (1). As the adaptive quantization of different weight groups can be fully parallelized, GPUs enable highly efficient offline quantization. Specifically, on a single NVIDIA 3090 GPU, quantizing all weights of Llama-3.1-8B [9] to 3-bit and 4-bit Adana numeric types takes only 15.8 and 48.7 seconds, respectively.

## 4.2 Online Activation Quantization

*Efficient Online Adana Quantization Algorithm.* Since activations are generated on the fly, an efficient adaptive quantization strategy is required to simplify the quantization unit. The naive adaptive selection method described in (4) requires instantiating $M$ quantizers and $M$ MSE computing units, resulting in substantial floating-point operations. In addition, the nonuniform grids prevent direct use of rounding for quantization. Selecting the closest discrete value within a sub numeric type requires value-by-value comparisons, which are inherently inefficient in hardware. Thus, we refine this strategy by exploiting the structural properties of the Adana numeric type to avoid repeated quantization and repeated MSE computing across different sub numeric types.

A key property of the Adana formulation is that all sub numeric types share the same base grid, i.e., the same quantization range. This is significantly different from other adaptive numeric types and eliminates the need to scale the original data to multiple ranges. The distinction among different Adana sub numeric types lies solely in the placement of their high-resolution grids. Consequently, when one Adana sub numeric type is said to outperform another, the statement essentially means that its high-resolution grid placement yields a lower quantization error.

We observe that the union of the base grid with all high-resolution grids gives rise to a new numeric type, denoted as $\cup T_m$. The quantization error on $\cup T_m$ provides a lower bound for the quantization error of any individual Adana sub numeric type. Motivated by this, we propose to use the quantized values on $\cup T_m$ as a proxy for the original data to approximate the quantization error of each Adana sub numeric type. Accordingly, the adaptive selection process is approximated as

$$\hat{T} = \arg\min_{T_m} \|q(\mathbf{x_f}, \bigcup_{m=0}^{M-1} T_m) - q(\mathbf{x_f}, T_m)\|_2, \quad T_m \in \mathbf{T}. \tag{8}$$

Using this approximate metric avoids floating-point MSE computing. Since each error is either 0 or 0.5, the comparison is simplified as counting the number of input elements that incur quantization errors.

**Algorithm 2** Online Adana quantization algorithm

**Input:** Biased and scaled floating-point vector $\mathbf{x} = [x_0, \ldots, x_{G-1}]$; Adana configuration $N_{\text{int}}$, $N_{\text{frac}}$, and $M$.
**Output:** Encoded vector $\mathbf{y} = [y_0, \ldots, y_{G-1}]$ and Adana index $m$.

1: **function** QUANT$(\mathbf{x}; N_{\text{int}}, N_{\text{frac}}, M)$
2: $\mathbf{e} \leftarrow \mathbf{0}$ ▷ *Initialize the error table*
3: **for** $k = 0$ **to** $G - 1$ **do** ▷ *Sequential over input elements*
4: $q_k \leftarrow \text{round}(x_k)$ ▷ *Integer rounding*
5: $p_k \leftarrow \text{round}(2 \cdot x_k)/2$ ▷ *Half-step rounding*
6: **for** $j = 0$ **to** $M - 1$ **do** ▷ *Parallel across sub numeric types*
7: **if** $\text{frac}(p_k) \neq 0$ **and** $\text{int}(p_k) \notin [j,\ j + N_{\text{frac}})$ **then**
8: $e_j \leftarrow e_j + 1$
9: $m \leftarrow \arg\min_j e_j$ ▷ *Select the index with minimal error*
10: **for** $k = 0$ **to** $G - 1$ **do** ▷ *Parallel across rounding results*
11: **if** $\text{frac}(p_k) \neq 0$ **and** $\text{int}(p_k) \in [m,\ m + N_{\text{frac}})$ **then**
12: $y_k \leftarrow \text{int}(p_k) - m + N_{\text{int}}$
13: **else**
14: $y_k \leftarrow q_k$
15: **return** $(\mathbf{y}, m)$

Building on this metric, we further optimize the online quantization algorithm for Adana, as shown in Algorithm 2. Assume the input vector $\mathbf{x}$ has already been biased and scaled following the same procedure as used in `INT-Asym`. In Line 2, an error table $\mathbf{e}$ is maintained for all Adana sub numeric types. Each entry of $\mathbf{e}$ records the number of quantized outputs under the corresponding sub numeric type that differ from those under $\cup T_m$. In the first loop, each input of the group is rounded to its nearest integer and nearest 0.5, respectively, as indicated in Lines 4 and 5. If the 0.5-rounded value happens to be an integer, the table $\mathbf{e}$ is unchanged because every sub numeric type can represent this value

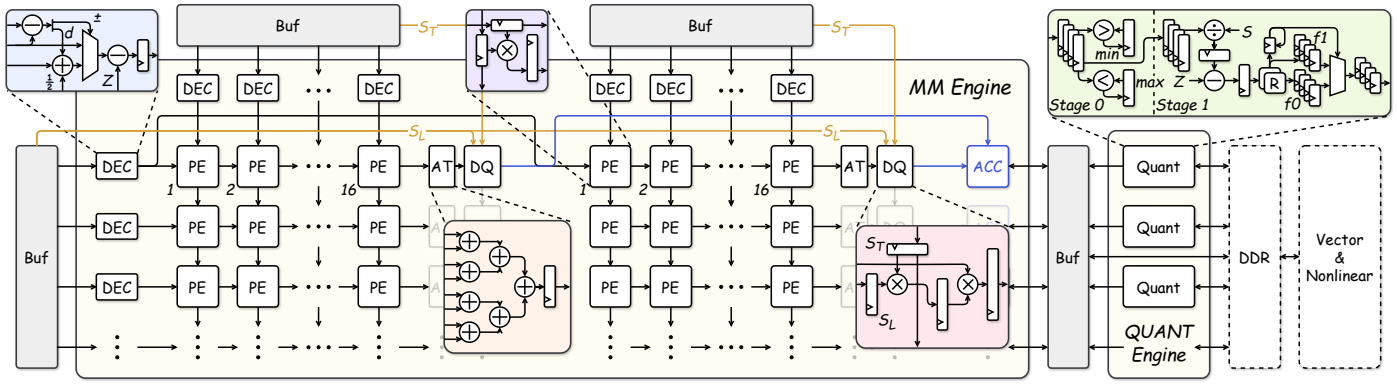



**Figure 4: Overview of the Adana accelerator, consisting of on-chip SRAM buffers, an MM engine, and a QUANT engine.**

exactly. If the 0.5-rounded value is not an integer, sub numeric types whose high-resolution grids do not contain this value cannot represent it precisely and incur an error; therefore, the corresponding entries in $\mathbf{e}$ are updated in parallel, as shown by the inner loop in Line 6. Once all inputs in the group have been processed, the sub numeric type with the smallest error entry is selected, as shown in Line 9. In the loop in Line 10, each element in the group is encoded in parallel with the selected sub numeric type. If the 0.5-rounded value is not an integer and falls within the high-resolution grid, we encode it as per (6). Otherwise, the element lies in the base grid and is encoded directly as the integer-rounded value.

*Quantization Layout for the KV Cache.* To ensure that the grouping direction in the matrix multiplications used for the attention mechanism is aligned with the direction of the associated vector dot products, we quantize the $L$ key vectors and $L$ value vectors in the KV cache along different axes, where $L$ denotes the sequence length. Since each key vector independently participates in a dot product with the query vector of the current token, grouping only needs to be applied within each individual key vector. In contrast, the components of the value vectors at the same positions participate in a weighted vector dot product with the attention scores to form the aggregated output. To enable the efficient computation illustrated in (3), the grouping dimension for the value cache must therefore follow the sequence dimension. As the sequence length $L$ grows during generation, we maintain a temporary FP16 buffer for newly generated value vectors. When the number of appended value vectors is smaller than one full group, this buffer is quantized at each update. Once the buffer accumulates an entire group, we quantize the group and then clear the temporary FP16 buffer, retaining only the low-precision Adana numeric types for that group.

## 5 Adana Microarchitecture

*Accelerator and Dataflow.* To accelerate the matrix multiplication (MM) under low-bit group-wise quantization in LLM inference, we propose the Adana microarchitecture incorporating a scalable accelerator and an optimized dataflow. As shown in Figure 4, the Adana accelerator consists of an MM engine and a QUANT engine, together with on-chip SRAM buffers. The unquantized activations are first processed by the QUANT engine and converted into low-bit Adana values. Once all inputs are quantized, the left-hand-side matrix is loaded into the MM engine. After passing through a column of decoders (DEC), the data are distributed across tiles and stored within the processing element (PE) arrays. Each tile contains 16 PE columns, matching the minimum group size of 16, and all PEs in a row within a tile store the decoded Adana values belonging to the same quantization group. The right-hand-side matrix is then streamed from the top of the MM engine. After decoding, the data are sent to the PE arrays and then multiplied row-by-row with the locally stored values. Within each tile, the 16 multipliers in every row operate in parallel. Their outputs are forwarded in the following cycle to a column of adder trees (AT). The resulting sums are subsequently passed to a column of dequantizers (DQ), where they are converted back to higher-precision values using the scaling factors corresponding to the two groups involved in the dot product. Finally, the dequantized results from all tiles are forwarded to a column of accumulators (ACC) to update the buffered partial sums. In addition, the number of tiles in the MM engine, the number of rows in each PE array, and the number of quantization units in the QUANT engine are configurable, enabling architectural scalability.

*MM Engine.* On the left side of the MM engine, each DEC processes one quantization group sequentially, so different DECs receive different group-wise parameters, including the Adana mode, zero point, and bit width configuration. On the top side of the MM engine, each tile contains 16 DECs that operate in parallel to decode one group, and these DECs share the same group-wise parameters. A 4-bit Adana value ranges from 0 to 11 with one fractional bit. After decoding, it is interpreted as a fixed-point number in the range of $[-11, 11]$, which requires 6 bits (5 integer bits and 1 fractional bit). Similarly, a 3-bit Adana value is interpreted as a fixed-point number in the range of $[-5, 5]$, requiring 5 bits. In this way, Adana values of different precisions can all participate in subsequent operations in the INT format.

*QUANT Engine.* The quantization unit adopts a two-stage pipeline. All FP16 values within the same group enter the first stage sequentially, where they are buffered while the running min/max values are updated. After the entire group is loaded, the buffered data are transferred to the input registers of the second stage, allowing the min/max module to begin processing the next group. In the second stage, a floating-point divider and subtractor first compute the scaling factor and zero point based on the min/max values. These two floating-point units are then reused to apply subtraction and scaling to each buffered FP16 value. After being mapped into the valid Adana range, each value is passed through two rounding units, producing integer-rounded and 0.5-rounded candidates and updating their corresponding error entries, as per Algorithm 2. Once all FP16 inputs in the current group are processed, the sub numeric type for the group can be selected. In the next cycle, all results are encoded in parallel using the cached integer-rounded and 0.5-rounded candidates. Throughout this process, the floating-point datapath requires only a divider and a subtractor, mirroring the basic requirements of INT-Asym quantization.

## 6 Experiments

### 6.1 Accuracy Evaluation

*Comparison with Adaptive Quantization Schemes.* To enable a like-for-like evaluation of Adana, we compare it against three state-of-the-art

**Table 2: Perplexity (lower is better) of 15 LLMs under W4/A4 and W3/A3 quantization with group size 16 using different adaptive numeric types.**

| W/A | Method | Llama-3.1/3.2 | | | Phi-1.5/2/3 | | | Yi-1.5 | | | Gemma-2/3 | | | Qwen-3 | | | Average |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1B | 3B | 8B | 1.3B | 2.7B | 4B | 6B | 9B | 34B | 1B | 2B | 4B | 4B | 8B | 14B | |
| 16/16 | FP16 | 9.75 | 7.81 | 6.24 | 21.82 | 9.71 | 6.01 | 5.73 | 5.48 | 4.90 | 10.58 | 8.71 | 7.38 | 7.90 | 6.99 | 6.38 | 8.36 |
| 4/4 | INT-Asym | 12.44 | 8.70 | 6.95 | 23.18 | 11.84 | 6.94 | 6.18 | 5.92 | 5.16 | 12.33 | 9.32 | 8.22 | 8.62 | 7.70 | 6.83 | 9.36 |
| | ANT | 12.78 | 8.90 | 7.10 | 23.05 | 11.82 | 7.00 | 6.18 | 5.93 | 5.15 | 11.83 | 9.36 | 7.94 | 8.69 | 7.61 | 6.95 | 9.35 |
| | M-ANT | 13.07 | 8.82 | 7.02 | 23.23 | 12.02 | 7.08 | 6.22 | 5.98 | 5.17 | 12.32 | 9.35 | 8.19 | 8.75 | 7.71 | 6.88 | 9.45 |
| | BitMod | 12.33 | 8.69 | 6.93 | 23.10 | 12.11 | 6.70 | 6.13 | 5.88 | 5.09 | 11.94 | 9.27 | 8.08 | 8.59 | 7.53 | 6.79 | 9.28 |
| | Adana | 11.55 | 8.44 | 6.78 | 22.61 | 11.04 | 6.58 | 6.06 | 5.74 | 5.08 | 11.55 | 9.12 | 7.95 | 8.44 | 7.43 | 6.66 | 9.00 |
| 3/3 | INT-Asym | 112.66 | 26.12 | 12.49 | 30.01 | 18.30 | 12.53 | 8.95 | 8.61 | 6.15 | 35.79 | 13.54 | 12.18 | 14.43 | 11.81 | 15.77 | 22.62 |
| | ANT | 471.14 | 57.45 | 35.79 | 37.79 | 26.79 | 29.25 | 14.70 | 14.89 | 7.13 | 86.58 | 19.03 | 17.43 | 28.75 | 20.05 | 39.08 | 60.39 |
| | M-ANT | 990.75 | 80.03 | 45.40 | 43.22 | 26.26 | 48.68 | 20.60 | 25.08 | 7.88 | 118.59 | 22.30 | 19.77 | 41.31 | 25.55 | 40.91 | 103.76 |
| | BitMod | 98.80 | 18.54 | 11.65 | 28.52 | 19.41 | 11.57 | 8.52 | 8.16 | 5.96 | 25.87 | 12.57 | 11.57 | 13.31 | 11.01 | 11.33 | 19.79 |
| | Adana | 53.95 | 13.17 | 10.31 | 26.98 | 17.07 | 10.12 | 7.93 | 7.54 | 5.73 | 21.60 | 11.65 | 10.52 | 11.59 | 10.12 | 9.27 | 15.17 |

adaptive numeric types, ANT [11], M-ANT [12], and BitMod [3]. For reference, we also include INT-Asym, which provides adaptivity through zero-point offsets. In their original designs, ANT employs tensor-wise quantization, whereas M-ANT and BitMod quantize weights with group sizes of 64 and 128, respectively, and keep activations in FP16 or INT8. For a fair and consistent comparison, all methods are configured with the same group size, apply joint weight-activation quantization (including KV caches and attention scores), and adopt INT8 group-wise scaling factors. This unified configuration provides a comprehensive and equitable basis for evaluating Adana and existing adaptive numeric types in low-bit group-wise LLM inference.

We first evaluate the perplexity on WikiText-2 [19] for the above methods using 15 LLMs from five model families [9, 13, 28, 35, 36], with a group size of 16. As shown in Table 2, M-ANT performs worse than ANT, primarily because its variance-based online adaptive quantization overly approximates the selection process, leading to suboptimal choices. Among existing methods, BitMod achieves the best performance for most models due to its limited yet meaningful ability to adapt to asymmetric distributions. Across different quantization bit widths and model architectures, Adana consistently attains the lowest perplexity, benefiting from its ability to simultaneously adapt to both nonuniformity and asymmetry of the data under small group sizes. Specifically, compared to the strongest baseline, BitMod, Adana reduces the average perplexity by 0.28 for W4/A4 quantization and 4.62 for W3/A3 quantization. Furthermore, when focusing on relatively small models, the improvement becomes more pronounced, as larger models inherently tolerate quantization errors better across all methods. This highlights the potential of Adana for improving low-bit LLM inference accuracy on edge or portable devices.

The same trend can be observed in the zero-shot accuracy evaluated on LAMBADA [22], PIQA [2], ARC-Easy [4], ARC-Challenge [4], HellaSwag [39] and their average for Mistral-7B-v0.3 [14] under W3/A3 quantization, as shown in Table 3. Compared to existing methods, Adana improves the average accuracy across five datasets by 2.00% to 16.34%, further narrowing the gap between 3-bit quantized and the original floating-point models on downstream tasks.

**Table 3: Zero-shot accuracy (higher is better) of Mistral-7B-v0.3 under W3/A3 quantization with group size 16.**

| Method | Lambada | Piqa | Arc-e | Arc-c | Hella | Average |
|---|---|---|---|---|---|---|
| FP16 | 75.32 | 82.26 | 78.24 | 52.22 | 80.43 | 73.69 |
| INT-Asym | 65.73 | 76.88 | 69.53 | 43.43 | 75.50 | 66.21 |
| ANT | 55.72 | 71.49 | 59.05 | 37.71 | 66.25 | 58.04 |
| M-ANT | 52.41 | 64.96 | 50.93 | 32.00 | 59.05 | 51.87 |
| BitMod | 66.68 | 75.57 | 68.14 | 42.49 | 75.14 | 65.60 |
| Adana | 68.35 | 77.48 | 72.31 | 45.65 | 77.24 | 68.21 |

Although quantization using a smaller group size can improve the accuracy, it requires proportionally more memory for group-wise quantization parameters. To further assess how these adaptive numeric types perform under different additional memory budgets, we measure their perplexity on WikiText-2 across a range of group sizes using Falcon-3-7B [27], as shown in Table 4. Under W4/A4 quantization with relatively large group sizes (e.g., 128), INT-Asym yields the highest perplexity. In this scenario, the benefits of adapting to distribution nonuniformity are greater than asymmetry, yet INT-Asym fails to capture the nonuniformity. With smaller group sizes (e.g., 32 or 16) or under lower-bit quantization (W3/A3), INT-Asym performs better and surpasses ANT and M-ANT that cannot adapt to asymmetry. Across all group sizes, Adana consistently achieves the lowest perplexity. Notably, decreasing the group size brings perplexity closer to floating-point results. For example, when using Adana, reducing the group size from 32 to 16 lowers the perplexity increase of Falcon-3-7B from 0.29 to 0.18 under W4/A4, and from 1.77 to 1.04 under W3/A3. These results suggest that hardware architectures should support group sizes as small as 16. Indeed, NVIDIA's Blackwell architecture [29] via the 4-bit NVFP format [1] already provides support for groups of 16 values per scale.

**Table 4: Perplexity (lower is better) of Falcon-3-7B with group sizes from 16 to 128. The perplexity of FP16 baseline is 6.11.**

| Method | W/A 4/4 | | | | W/A 3/3 | | | |
|---|---|---|---|---|---|---|---|---|
| | 16 | 32 | 64 | 128 | 16 | 32 | 64 | 128 |
| INT-Asym | 6.38 | 6.52 | 6.77 | 7.23 | 7.64 | 8.99 | 13.07 | 31.67 |
| ANT | 6.39 | 6.52 | 6.66 | 6.82 | 8.88 | 12.30 | 24.99 | 89.45 |
| M-ANT | 6.40 | 6.58 | 6.80 | 7.15 | 10.22 | 13.54 | 29.00 | 109.95 |
| BitMod | 6.36 | 6.46 | 6.62 | 6.82 | 7.43 | 8.31 | 10.15 | 15.70 |
| Adana | 6.29 | 6.40 | 6.55 | 6.74 | 7.15 | 7.88 | 9.30 | 13.12 |

*Comparison with Microscaling Formats.* By using a shared scaling factor for all elements within a block, microscaling formats preserve dynamic range and numeric precision at low bit widths. In essence, they can be regarded as a form of group-wise quantization. We consider two microscaling formats with a group size of 32, MXINT/MXFP [23], and two with a group size of 16, NVINT/NVFP [1]. Adana is configured with the same respective group sizes to ensure a fair comparison. We apply these methods to quantize Llama-2-7B/13B [31] to W4/A4 and W3/A3, and evaluate their perplexity on WikiText-2. The results are summarized in Table 5. Both NV formats achieve lower perplexity than the MX formats, due to their smaller group size and FP8 group-wise scaling. The FP variants consistently outperform their INT counterparts, as they better capture nonuniformities in the data distribution. Under the W4/A4 setting, Adana attains the lowest perplexity, outperforming the strongest microscaling method, NVFP. This gain stems from the adaptability of Adana to accommodate diverse nonuniform and asymmetric data distributions. Under the more challenging W3/A3 setting, both MX formats fail to produce usable perplexity, and even NVFP incurs significant perplexity increases relative to the floating-point baseline. In contrast, Adana, with a group size of 16, increases perplexity by only 1.89 and 1.39 on the 7B and 13B models, respectively.

*Comparison with Outlier-Aware Quantization Methods.* We compare Adana with group sizes of 128 and 16 with three existing outlier-aware quantization methods, HotaQ [26], Oltron [34], and DuoQ [37], in terms of perplexity on WikiText-2. The evaluation covers W4/A4 quantized

**Table 5: Perplexity (lower is better) of Llama-2-7B/13B using microscaling formats and Adana quantization.**

| Size | FP16 | W/A | MXINT | MXFP | NVINT | NVFP | Adana§ | Adana¶ |
|---|---|---|---|---|---|---|---|---|
| 7B | 5.47 | 4 | 8.88 | 7.23 | 6.29 | 6.06 | 5.85 | 5.71 |
| | | 3 | 3622.26 | 683.12 | 38.85 | 19.71 | 8.85 | 7.36 |
| 13B | 4.88 | 4 | 7.08 | 6.37 | 5.52 | 5.35 | 5.15 | 5.06 |
| | | 3 | 962.23 | 334.87 | 19.20 | 11.48 | 7.69 | 6.27 |

§ $G = 32$, matching MXINT/MXFP ¶ $G = 16$, matching NVINT/NVFP

OPT [40] and Llama [30] models. As shown in Table 6, with a marginal memory overhead (an equivalent bit width of 4.125), Adana (lossy) outperforms state-of-the-art W4/A4 outlier-aware quantization methods. Notably, when a moderate level of group-wise quantization overhead is allowed, Adana (lossless) achieves lower perplexity, with less than a 0.2 increase compared to the floating-point baseline across all evaluated models, while still reducing memory usage by roughly 70%.

**Table 6: Perplexity (lower is better) of OPT and Llama model families using W4/A4 quantization methods.**

| Model | Size | FP16 | HotaQ | Oltron | DuoQ | Adana† | Adana‡ |
|---|---|---|---|---|---|---|---|
| OPT | 13B | 10.12 | 13.90 | 11.35 | 11.60 | 10.70 | 10.32 |
| | 30B | 9.56 | 11.02 | 10.51 | 10.44 | 9.97 | 9.75 |
| | 66B | 9.34 | 11.29 | 10.49 | 10.21 | 9.90 | 9.54 |
| Llama | 13B | 5.09 | 9.07 | 8.20 | 5.89 | 5.54 | 5.23 |
| | 30B | 4.10 | 7.39 | 6.68 | 5.14 | 4.75 | 4.28 |
| | 65B | 3.53 | 6.02 | 5.82 | 4.23 | 4.05 | 3.67 |

† lossy, $G = 128$ ‡ lossless, $G = 16$

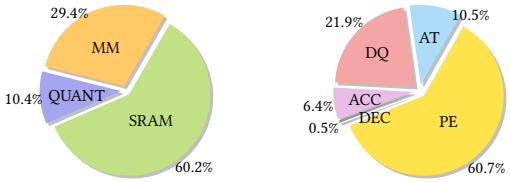
## 6.2 Hardware Evaluation

*Comparison of Online Quantization Units.* We evaluate the efficiency of Adana by comparing the area and power of quantization units with those of other adaptive numeric types. In this evaluation, all designs take a sequence of FP16 inputs and support quantization into their corresponding 4-bit and 3-bit numeric types. INT-Asym, ANT, BitMod, and Adana require two passes over the FP16 inputs, one for extracting the min/max values and the other for adaptive quantization. M-ANT requires an additional pass, due to its variance-based adaptive process. In all designs, the stages are pipelined and optimized to ensure comparable throughput. All quantization units are implemented in Verilog with Synopsys DesignWare floating-point arithmetic IPs. Synthesis is performed by Synopsys Design Compiler using a 28 nm CMOS technology at 25 °C, 0.9 V, and a 500 MHz clock. The generated netlists are then used to estimate power consumption using Synopsys PrimeTime PX. The experimental results are shown in Table 7. As observed, INT-Asym that employs the simplest rounding strategy, achieves the smallest area and power. In contrast, ANT and BitMod that minimize quantization MSE incur significantly higher hardware overhead due to the need for parallel quantization and MSE computation across four sub numeric types. Although M-ANT supports 16 sub numeric types, its variance-based adaptive mechanism simplifies the selection into a direct mapping from the variance value, resulting in relatively low hardware cost. Finally, Adana reduces the area and power of quantization units by 65.40% and 66.75%, respectively, compared to the strongest accuracy baseline BitMod.

**Table 7: Area ($\mu m^2$) and power (mW) of different quantization units.**

| Metric | INT-Asym | ANT | M-ANT | BitMod | Adana |
|---|---|---|---|---|---|
| Area | 4,171.0 | 14,328.1 | 7,828.9 | 14,931.3 | 5,165.9 |
| Power | 2.43 | 8.22 | 3.90 | 8.42 | 2.80 |

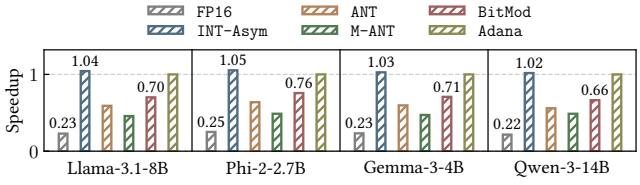
*Area Breakdown of Adana Accelerator.* We prototype an instance of the Adana accelerator, featuring 512 KB of SRAM, a QUANT engine with 64 quantization units, and an MM engine composed of four tiles (4096 PEs in total). The SRAM characteristics are modeled using CACTI [20]. The MM engine is implemented and synthesized following the same methodology used for the quantization units. Figure 5 shows the area breakdown of the components in the Adana accelerator.



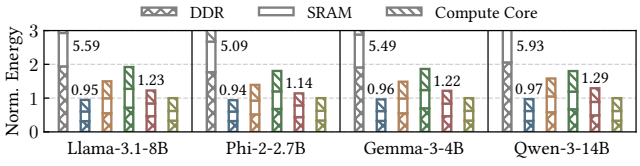

(a) Accelerator (b) MM Engine

**Figure 5: Area breakdown of components in the Adana accelerator.**

*Comparison of Accelerator Performance.* We develop a cycle-level simulator to evaluate the performance of different accelerators on four LLMs. The simulator uses SRAM parameters from CACTI and power results estimates from PrimeTime PX. During simulation, all accelerators are configured with the same on-chip SRAM and off-chip DDR. To ensure iso-area accelerators, we adjust the number of PEs in each design according to its area overhead. The speedup and normalized energy are summarized in Figure 6. On average, Adana achieves 1.68×, 2.10×, and 1.42× speedups, along with energy reductions of 32.81%, 45.85%, and 17.91%, compared to ANT, M-ANT, and BitMod, respectively. These improvements stem from Adana's efficient QUANT engine, simplified PE architecture, and reduced partial-sum bit width. Compared to the reference design INT-Asym, Adana introduces only marginal hardware overhead, leading to average latency and power increases of 3.50% and 4.79%, respectively. Compared to the FP16 accelerator without QUANT engine, Adana reduces memory footprint by about 70% while achieving a 4.32× speedup and an 81.87% reduction in energy.




(a) Comparison of speedup.




(b) Comparison of normalized energy.

**Figure 6: Speedup and normalized energy of different accelerators.**

## 7 Conclusion

To address the nonuniform and asymmetric data characteristics that arise in group-wise quantization for LLMs, this paper introduces Adana, comprising an adaptive nonuniform asymmetric numeric type, an efficient quantization framework, and an acceleration microarchitecture. With a carefully designed strategy, Adana adaptively identifies the appropriate data range and allocates higher resolution to the critical portion of the distribution, achieving lower quantization error than existing methods. Adana also delivers significantly improved online quantization efficiency, enabled by the proposed appropriate adaptive algorithm. Experimental results show that, compared to existing methods, Adana improves the accuracy of 4-bit and 3-bit weight-activation quantization, further narrowing the gap between low-bit fully quantized LLMs and their original FP16 counterparts. Meanwhile, Adana demonstrates notable speedups and energy savings across multiple LLMs, compared to state-of-the-art quantization schemes.

## References


[1] Eduardo Alvarez, Omri Almog, Eric Chung, Simon Layton, Dusan Stosic, et al. 2025. Introducing NVFP4 for Efficient and Accurate Low-Precision Inference | NVIDIA Technical Blog. *URL https://developer. nvidia. com/blog/introducing-nvfp4-for-efficient-and-accurate-low-precision-inference* (2025).

[2] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. PIQA: Reasoning about Physical Commonsense in Natural Language. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 7432–7439.

[3] Yuzong Chen, Ahmed F AbouElhamayed, Xilai Dai, Yang Wang, Marta Andronic, et al. 2025. BitMoD: Bit-serial Mixture-of-Datatype LLM Acceleration. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1082–1097.

[4] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, et al. 2018. Think You Have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv preprint arXiv:1803.05457* (2018).

[5] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. GPT3.Int8(): 8-Bit Matrix Multiplication for Transformers at Scale. *Advances in neural information processing systems* 35 (2022), 30318–30332.

[6] Xinkuang Geng, Siting Liu, Jianfei Jiang, Kai Jiang, and Honglan Jiang. 2024. Compact Powers-of-Two: An Efficient Non-Uniform Quantization for Deep Neural Networks. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.

[7] Xinkuang Geng, Siting Liu, Leibo Liu, Jie Han, and Honglan Jiang. 2024. QUQ: Quadruplet Uniform Quantization for Efficient Vision Transformer Inference. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.

[8] Xinkuang Geng, Siting Liu, Hui Wang, Jie Han, and Honglan Jiang. 2025. Lookup Table Refactoring: Towards Efficient Logarithmic Number System Addition for Large Language Models. In *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 1–7.

[9] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. 2024. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783* (2024).

[10] Cong Guo, Jiaming Tang, Weiming Hu, Jingwen Leng, Chen Zhang, et al. 2023. OliVe: Accelerating Large Language Models via Hardware-friendly Outlier-Victim Pair Quantization. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–15.

[11] Cong Guo, Chen Zhang, Jingwen Leng, Zihan Liu, Fan Yang, et al. 2022. ANT: Exploiting Adaptive Numerical Data Type for Low-bit Deep Neural Network Quantization. In *2022 55th IEEE/ACM international symposium on microarchitecture (MICRO)*. IEEE, 1414–1433.

[12] Weiming Hu, Haoyan Zhang, Cong Guo, Yu Feng, Renyang Guan, et al. 2025. M-ANT: Efficient Low-bit Group Quantization for LLMs via Mathematically Adaptive Numerical Type. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1112–1126.

[13] Mojan Javaheripi, Sébastien Bubeck, Marah Abdin, Jyoti Aneja, Sebastien Bubeck, et al. 2023. Phi-2: The Surprising Power of Small Language Models. *Microsoft Research Blog* 1, 3 (2023), 3.

[14] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023).

[15] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, et al. 2020. Scaling Laws for Neural Language Models. *arXiv preprint arXiv:2001.08361* (2020).

[16] Jahyun Koo, Dahoon Park, Sangwoo Jung, and Jaeha Kung. 2024. OPAL: Outlier-Preserved Microscaling Quantization Accelerator for Generative Large Language Models. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.

[17] Jungi Lee, Wonbeom Lee, and Jaewoong Sim. 2024. Tender: Accelerating Large Language Models via Tensor Decomposition and Runtime Requantization. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1048–1062.

[18] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, et al. 2024. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. *Proceedings of machine learning and systems* 6 (2024), 87–100.

[19] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer Sentinel Mixture Models. https://doi.org/10.48550/arXiv.1609.07843 arXiv:1609.07843

[20] Naveen Muralimanohar, Rajeev Balasubramonian, Norman P Jouppi, et al. 2009. CACTI 6.0: A Tool to Model Large Caches. *HP laboratories* 27 (2009), 28.

[21] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, et al. 2021. A White Paper on Neural Network Quantization. *arXiv preprint arXiv:2106.08295* 4 (2021).

[22] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, et al. 2016. The LAMBADA Dataset: Word Prediction Requiring a Broad Discourse Context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*. 1525–1534.

[23] Bita Darvish Rouhani, Ritchie Zhao, Ankit More, Mathew Hall, Alireza Khodamoradi, et al. 2023. Microscaling Data Formats for Deep Learning. *arXiv preprint arXiv:2310.10537* (2023).

[24] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. WinoGrande: An Adversarial Winograd Schema Challenge at Scale. *Commun. ACM* 64, 9 (2021), 99–106.

[25] Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, et al. 2023. OmniQuant: Omnidirectionally Calibrated Quantization for Large Language Models. *arXiv preprint arXiv:2308.13137* (2023).

[26] Xuan Shen, Zhaoyang Han, Lei Lu, Zhenglun Kong, Peiyan Dong, et al. 2024. HotaQ: Hardware Oriented Token Adaptive Quantization for Large Language Models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2024).

[27] Falcon-LLM Team. 2024. The Falcon 3 Family of Open Models. https://huggingface.co/blog/falcon3

[28] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, et al. 2025. Gemma 3 Technical Report. *arXiv preprint arXiv:2503.19786* (2025).

[29] Ajay Tirumala and Raymond Wong. 2024. NVIDIA Blackwell Platform: Advancing Generative AI and Accelerated Computing. In *2024 IEEE Hot Chips 36 Symposium (HCS)*. IEEE Computer Society, 1–33.

[30] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, et al. 2023. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971* (2023).

[31] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, et al. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv preprint arXiv:2307.09288* (2023).

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, et al. 2017. Attention Is All You Need. *Advances in neural information processing systems* 30 (2017).

[33] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, et al. 2023. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. In *International conference on machine learning*. PMLR, 38087–38099.

[34] Chenhao Xue, Chen Zhang, Xun Jiang, ZhuTianYa Gao, Yibo Lin, et al. 2024. Oltron: Algorithm-Hardware Co-design for Outlier-Aware Quantization of LLMs with Inter-/Intra-Layer Adaptation. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.

[35] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, et al. 2025. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388* (2025).

[36] Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, et al. 2024. Yi: Open Foundation Models by 01.AI. *arXiv preprint arXiv:2403.04652* (2024).

[37] Zhuoquan Yu, Huidong Ji, Yue Cao, Junfu Wu, Xiaoze Yan, et al. 2025. DuoQ: A DSP Utilization-aware and Outlier-free Quantization for FPGA-based LLMs Acceleration. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–7.

[38] Zhihang Yuan, Chenhao Xue, Yiqi Chen, Qiang Wu, and Guangyu Sun. 2022. PTQ4ViT: Post-training Quantization for Vision Transformers with Twin Uniform Quantization. In *European conference on computer vision*. Springer, 191–207.

[39] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a Machine Really Finish Your Sentence?. In *Proceedings of the 57th annual meeting of the association for computational linguistics*. 4791–4800.

[40] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, et al. 2022. OPT: Open Pre-trained Transformer Language Models. *arXiv preprint arXiv:2205.01068* (2022).