PeLog: Rotation-Assisted Logarithmic Encoding for Low-Bit Large Language Model Quantization

Xiangfei Hu[1], Lancheng Zou[2], Xilong Kang[1], Yuchen Liu[1], Jiajie Xu[1], Tairan Cheng[2], Yuyang Ye[2], Chang Meng[3†], Hao Yan[1], Bei Yu[2]
[1]Southeast University [2]Chinese University of Hong Kong [3]Eindhoven University of Technology

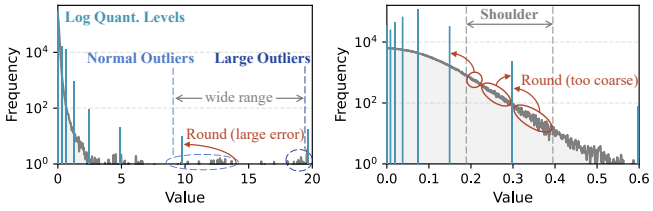
Abstract

Logarithmic quantization shows strong promise for efficient LLM deployment, but remains limited in low-bit settings due to its inability to handle wide-range outliers. While rotation mitigates outliers, its over-smoothing of the data distribution induces misalignment with logarithmic quantization. To address this, we propose PeLog, an algorithm-hardware co-design framework based on hierarchical quantization, integrating block-level local rotation with sub-block-level logarithmic encoding. By restricting rotation to blocks spanning only a subset of channels, PeLog compresses the outlier range without over-smoothing the data distribution. At a finer sub-block granularity, group-wise logarithmic encoding is employed for efficient quantization. Moreover, the homogenizing effect of rotation enables PeLog to dynamically allocate quantization levels within a narrow range of logarithmic bases, further improving accuracy and hardware efficiency. Implemented on an architecture with low-bit logarithmic processing elements and real-time quantization units, PeLog preserves state-of-the-art accuracy under the W4A4KV4 configuration, while achieving 1.52× performance improvement and 29% energy savings over the previous LLM accelerator.

1 Introduction

Large Language Models (LLMs) [1–3] have achieved remarkable performance in diverse natural language processing tasks, such as real-time conversation and translation. Despite their impressive capabilities, the massive parameter count and computational intensity incur substantial memory and energy overhead during inference [4]. This hinders their practical use in scenarios with limited resources, particularly for on-device applications. In this context, low-bit quantization [4–17] offers a promising solution. It maps high-precision values to low-bit representations while preserving acceptable model accuracy. Consequently, the storage and computation overhead for LLM deployment can be significantly reduced.

Among various methods, linear quantization is widely favored for its simplicity. However, its uniform quantization levels struggle with the long-tailed distributions commonly observed in LLMs [17, 18]. Although this can be mitigated by asymmetry and group-wise sharing of scaling factors and zero points [10, 14], frequent floating-point operations are introduced. In contrast, logarithmic quantization provides a finer resolution for small values and a wider coverage for large values [17–22], naturally aligning with the long-tailed distribution. Moreover, multiplication can be simplified into lightweight additions in the logarithmic domain and shifts. This implies that matrix multiplication, the primary computation in LLM inference, can be implemented in a more hardware-friendly manner.



(a) before rotation (b) after rotation

Figure 1: Logarithmic quantization for input activations of the first FFN down projection in Llama3.1-8B. The grey curve denotes the positive data distribution before quantization.

Nevertheless, in low-bit settings, logarithmic quantization is inherently restricted by the absence of explicit outlier awareness. In LLMs, outliers are not isolated anomalies but span a relatively wide range of values [8, 23]. As presented in Fig. 1(a), since the step sizes of logarithmic quantization increase exponentially, handling large outliers not only force small values to underflow to zero but also rounds normal outliers to coarse quantization levels. This incurs substantial quantization errors. Therefore, *effective outlier mitigation is essential to make logarithmic encoding viable for low-bit LLM quantization.*

Rotation [4–9] is an elegant way to address the outlier problem. It performs orthogonal transformations on pre-quantization tensors to redistribute outliers and reduce anisotropy, thereby reducing quantization difficulty without accuracy degradation. However, direct integration of logarithmic quantization and rotation introduces a certain misalignment. As illustrated in Fig. 1(b), by its mathematical nature, rotation typically reshapes the data into a near-normal distribution, where the shoulders exhibit appreciable data density [7]. In contrast, logarithmic quantization provides exponentially increasing step sizes. This leaves shoulders of the reshaped distribution with low resolution, thus increasing quantization errors.

Furthermore, optimizing logarithmic quantization itself for higher precision, such as layer- or tensor-wise base selection [19–22], incurs additional hardware overhead due to base switching and alignment. This further limits its practical use in efficient LLM quantization.

Based on the preceding analysis, two crucial questions should be answered: (1) *How can rotation be leveraged to fully unlock the potential of logarithmic encoding in low-bit LLM quantization?* (2) *How can logarithmic encoding be further refined to achieve a favorable balance between accuracy and hardware overhead?*

To this end, we propose PeLog, a novel framework for low-bit LLM quantization that elegantly integrates multi-base logarithmic encoding with lightweight rotation. By introducing local rotations with

This paper is supported by the National Natural Science Foundation of China (No. 62474038), the key R&D Program of Jiangsu Province (BE2023003-2), the Natural Science Foundation of Jiangsu Province (BK20232005), and the Fundamental Research Funds for the Central Universities. †Corresponding author: Chang Meng.

ICCAD '26, San Jose, CA, USA

ACM ISBN 979-8-4007-2873-0/2026/11
https://doi.org/10.1145/3831252.3834117

tunable scopes instead of global rotation over the entire tensor, the range of outliers is effectively compressed, while preserving a certain degree of long-tailed characteristic amenable to logarithmic quantization. Meanwhile, leveraging the homogenizing effect of rotation on different distributions, we dynamically allocate quantization levels across a narrow range of bases. This preserves the high precision of multi-base logarithmic encoding while significantly reducing the overhead of its hardware implementation. Our main contributions are summarized as follows:

- We propose a hierarchical quantization framework with block-level rotation and sub-block-level logarithmic encoding. The rotation scope is strictly restricted to blocks spanning a subset of channels, while group-wise logarithmic encoding is applied at a finer sub-block level for better adaptability. This effectively mitigates misalignment between logarithmic encoding and the data distribution after rotation.
- To balance quantization quality across value ranges, we dynamically allocate quantization levels to different logarithmic bases and smooth step size transitions at allocation boundaries using offsets. Benefiting from the homogenizing effect of rotation, these bases can be confined to a narrow range, which further improves hardware efficiency.
- We develop a sequential search strategy based on the learning space partitions algorithm. It determines optimal rotation scopes and base allocations, thus enhancing quantization accuracy.
- We design a systolic array-based architecture for LLM inference at 4-bit precision. It uses logarithmic processing elements to achieve multiplier-free matrix multiplication, and integrates real-time quantization units for configurable rotation and encoding.

Experimental results demonstrate that at W4A4KV4 precision, PeLog delivers higher accuracy across diverse downstream tasks than representative rotation-based methods [6, 8], while preserving favorable language modeling capability. Furthermore, compared to existing accelerators LightRot [7] and Tender [24], the PeLog accelerator achieves a performance improvement of 1.20-1.52× and reduces energy consumption by 11-29%.

2 Background

2.1 Linear and Logarithmic Quantization

Mainstream quantization methods typically map high-precision value (e.g., in IEEE FP16 format) to low-bit value in either linear or non-linear domains. This reduces the memory and computation overhead for LLM deployment. In linear quantization, the quantization levels are uniformly spaced in the linear domain, leading to a constant step size between adjacent dequantized values. For example, a typical linear quantization function can be formulated as follows:

$$q = \text{clamp}\left(\left\lfloor \frac{x}{S} \right\rceil, -2^{N-1}, 2^{N-1} - 1\right), \tag{1}$$

where x and q are the original and quantized values, respectively. S is the scaling factor, N is the number of bits in quantized data format, and $\lfloor \cdot \rceil$ is the rounding operation. The function clamp$(\cdot)$ restricts q to the representable range of the quantized data format.

Unlike linear quantization, logarithmic quantization uses levels uniformly spaced in the logarithmic domain. Therefore, the dequantized values exhibit progressively increasing step sizes, providing a finer resolution for small values and wider coverage for large values.

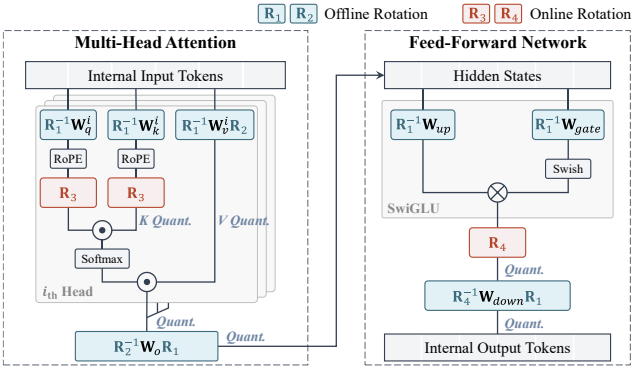


Figure 2: Schematic of applying offline and online rotations to a specific LLM layer. Matrices ($R_1 \sim R_4$) involve in this process.

A base-2 quantization function can be formulated as follows:

$$q = \text{clamp}\left(\left\lfloor -\log_2\left(\frac{|x|}{S}\right)\right\rceil, 0, 2^{N-1} - 1\right). \tag{2}$$

In this case, the dequantization function can be expressed simply as $\widehat{x} = S \cdot \text{Sgn}(x) \cdot 2^{-q}$, where Sgn$(\cdot)$ is the sign function. It is worth noting that logarithmic representations enable multiplier-free matrix multiplication using lightweight adders and shifters [17].

To further improve effectiveness, the aforementioned quantization methods can be applied in a group-wise manner [10–16]. By partitioning tensors into fine-grained groups and sharing parameters such as the scaling factor S within each group, a favorable balance between model accuracy and hardware efficiency can be achieved.

2.2 Quantization with Rotations

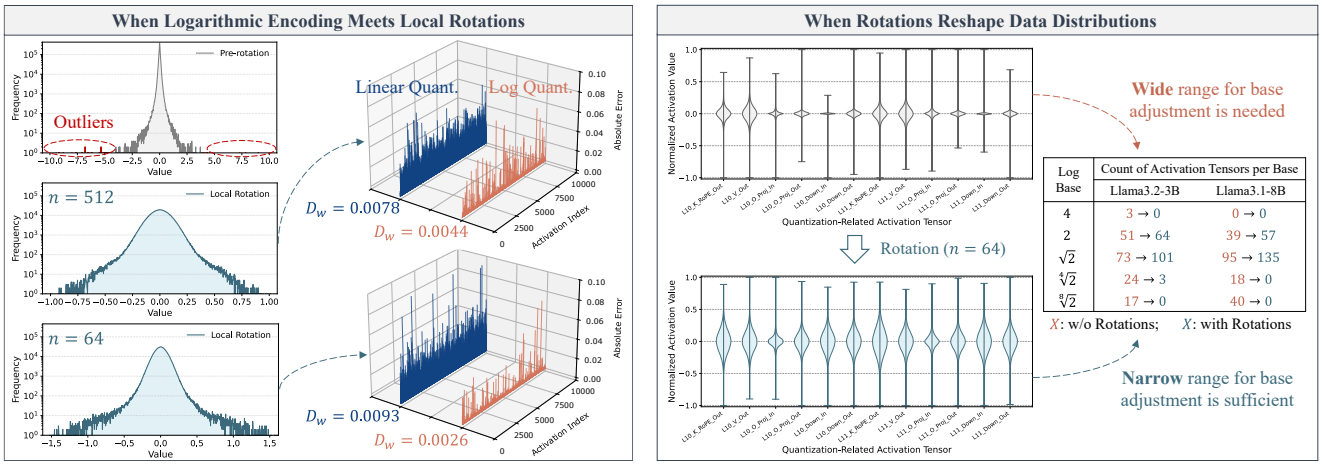
Outliers in LLMs degrade the resolution of quantization for normal values. Although Eq. (2) provides a finer resolution near zero, the outlier-induced increase in scaling factor S still forces many normal values to be rounded to zero, thus introducing significant quantization errors. Since outliers are crucial for maintaining the functional integrity of LLMs, directly clipping them can lead to severe accuracy degradation [13]. Moreover, isolating these outliers in groups with higher precision [25–28] incurs substantial overhead for grouping and mixed-precision processing.

Rotation elegantly addresses the outlier problem by reshaping the data distribution through matrix rotation [5, 6]. As shown in Fig. 2, it inserts a pair of orthogonal matrices $\mathbf{R}$ and $\mathbf{R}^{-1}$ into the linear layer, where $\mathbf{R}$ rotates the activations and $\mathbf{R}^{-1}$ is absorbed into the adjacent weight matrix. This operation preserves model equivalence while mitigating outliers, thus facilitating robust quantization.

Rotations can be categorized into offline and online modes for practical deployment. As illustrated in Fig. 2, for offline rotations, the constructed matrix pair (e.g., $\mathbf{R}_2$ and $\mathbf{R}_2^{-1}$) can be fully absorbed into the two adjacent weight tensors before inference. For online rotations, at least one matrix must be executed during inference due to prior operations like RoPE, incurring additional runtime overhead. Although online rotations can be simplified to a series of additions by applying Walsh-Hadamard matrices with recursive construction:

$$\mathbf{H}_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad \mathbf{H}_{2^e} = \mathbf{H}_2 \otimes \mathbf{H}_{2^{e-1}}, \tag{3}$$

and fast Hadamard transform (FHT) [5, 6, 9, 29], heavy data dependencies and excessive addition operations still pose challenges for hardware efficiency [7].



Log Base	Count of Activation Tensors per Base	
	Llama3.2-3B	Llama3.1-8B
4	3 → 0	0 → 0
2	51 → 64	39 → 57
$\sqrt{2}$	73 → 101	95 → 135
$\sqrt[4]{2}$	24 → 3	18 → 0
$\sqrt[8]{2}$	17 → 0	40 → 0

(a) (b)

Figure 3: Two observations on complementary benefits of logarithmic quantization and rotations. (a) Left: the smoothing effect of local rotations on input activations of the 26$^{\text{th}}$ FFN down projection in LLAMA3.1-8B. Right: corresponding errors under group-wise linear/logarithmic quantization. (b) Left: Activation distributions in two LLAMA3.1-8B layers with and without rotations. Right: allocation breakdown of logarithmic bases across related activation tensors in LLAMA3.2-3B and 3.1-8B.

3 Motivation

As discussed in Section 1, logarithmic quantization is promising but limited in low bit widths due to the wide range of outliers. While rotation mitigates outliers, directly applying it to assist logarithmic quantization introduces misalignment. Therefore, starting from potential modifications to both methods, we explore ways to eliminate misalignment and further improve quantization efficiency.

Observation 1: Logarithmic quantization fits well with the data distribution induced by local rotations. While global rotation amortizes the influence of outliers across the entire tensor for better quantizability, it incurs non-negligible online overhead. In fact, *local rotation* within a certain scope (e.g., a group of channels) can also effectively mitigate outliers, consistent with observations in [4, 7]. This implies that the tensor can be smoothed by a set of small rotation matrices, greatly reducing computational complexity.

Notably, the smoothing capability of local rotation varies with its scope [4]. The left part of Fig. 3(a) illustrates the smoothing effect of local rotation on data distributions, with the rotation scope set to n activation channels. Local rotations also compress the range of outliers, while the resulting data distribution exhibits more evident long-tailed characteristics as the scope n decreases (e.g., from 512 to 64). In this case, linear quantization once again suffers from underflow, forcing many small values to be rounded to zero. Even with group quantization, the resulting errors remain non-negligible, as shown in the right part of Fig. 3(a). In contrast, logarithmic quantization fits well with this long-tailed distribution devoid of large outliers, effectively reducing the quantization errors. Taking the case of $n = 64$ in Fig. 3(a) as an example, it further reduces the Wasserstein distance [30] D_w by 72% compared to linear quantization.

Based on the above observation, it is beneficial to *integrate the logarithmic quantization with local rotations*, thereby balancing the cost of rotation and the accuracy of logarithmic quantization. This motivates our hierarchical quantization framework in Section 4.

Observation 2: Adjusting the logarithmic base within a narrow range is sufficient for quantization with rotations. Adaptive adjustment of the logarithmic base by empirical rules or systematic search improves quantization performance [18–22]. In practice, the adjustable range of base is limited to 2^{2^γ} ($\gamma \in \mathbb{Z}$), such as $\{\sqrt[4]{2}, \sqrt{2}, 2, 4\}$. This enables base conversion through simple exponent shift.

Among the base options, a larger base provides a wider dynamic range and better hardware efficiency, at the cost of coarser granularity. In contrast, a smaller base provides finer granularity, but increases hardware overhead due to additional fraction handling from exponent shift. In conventional quantization scenarios, since data distributions can exhibit noticeable variation across quantization-related activation tensors (i.e., the activations to be quantized at different stages in Fig. 2), a wide range of base adjustment is needed to preserve model accuracy [19, 20, 22]. Consequently, the hardware necessitates more complex base switching and multi-bit fraction handling.

Rotations reshape diverse data distributions into more regular forms, thereby reducing distributional variation across quantization-related activation tensors [6]. As illustrated in the left part of Fig. 3(b), rotations transform activation distributions into more comparable forms in terms of zero-centered symmetry and probability density spread. This homogenizing effect makes certain bases designed for specific distributions unnecessary. For example, the right part of Fig. 3(b) compares base allocations across activation tensors with and without rotations. These bases are determined during post-training quantization (PTQ) using the fast progressive combining search in AdaLog [20]. The results indicate that the range for base adjustment can be narrowed to primarily 2 and $\sqrt{2}$ after rotations.

Based on the above observation, it is beneficial to *refine existing base allocation strategy to capitalize on the narrowed adjustment range*, thereby reducing both the search complexity and hardware implementation overhead. This motivates the dynamic allocation of quantization levels across logarithmic bases in Section 4.2.

4 Rotation-Assisted Logarithmic Encoding

Motivated by observation 1 in Section 3, we propose a hierarchical quantization framework comprising **block-level rotation** and **sub-block-level logarithmic encoding**. Given a tensor to be quantized, it is partitioned into blocks along the channel dimension based on the

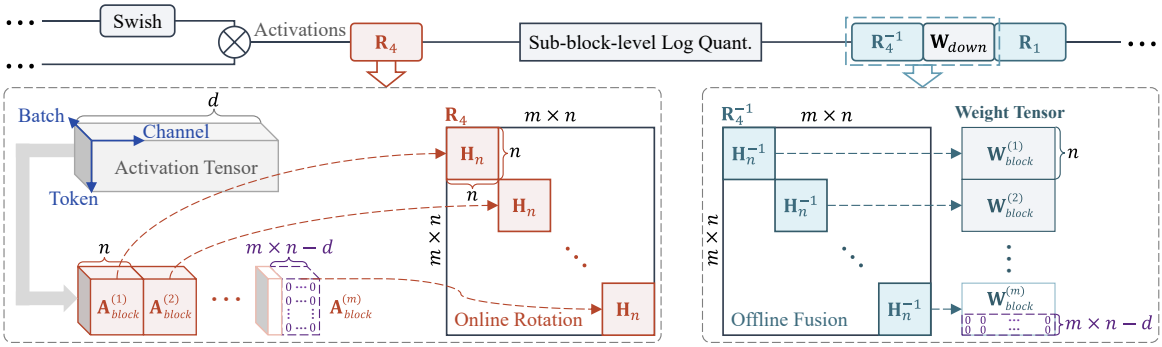


Figure 4: Block-level rotation on the input activation tensor of FFN down projection.

rotation scope, with each block rotated individually for outlier range compression (Section 4.1). Each block is further partitioned into sub-blocks of fixed size, where values undergo logarithmic encoding with a shared scaling factor. Inspired by observation 2, encoding efficiency is enhanced by the dynamic allocation of quantization levels across a narrow range of bases (Section 4.2). To maximize quantization performance, an offline sequential search based on learning space partitions is developed to determine the optimal scope for each type of rotation and quantization level allocation for each tensor (Section 4.3).

4.1 Block-Level Rotation

As discussed in Section 3, local rotations within a restricted scope n assists logarithmic encoding in reducing quantization errors. To enable these local rotations, the tensor should be properly partitioned prior to quantization. Considering the channel-specific nature of outliers, we partition the tensor into blocks along the *channel dimension*, where each block contains n channels (i.e., the scope of local rotation). The rotation is then applied locally within each block. For better performance of subsequent quantization, we set n as a tunable parameter, which can be determined during the calibration stage of PTQ. Meanwhile, to enable efficient rotations based on the Walsh-Hadamard matrix in Eq. (3), n is restricted to powers of two. If the number of channels is not an integer multiple of n, zero padding is applied for alignment.

Taking the activation tensor in Fig. 4 as an example, it has d channels, where d is not divisible by n. To fully partition the tensor into $m \in \mathbb{Z}$ blocks, zero padding is applied to extend it with $m \times n - d$ additional channels. After padding and partitioning, each block is independently transformed using an n-dimensional Hadamard matrix $\mathbf{H}_n$ for rotation. To preserve equivalence, during the offline fusion of inverse rotation matrices, the associated weight tensor is padded with zeros in the corresponding channels. The padded weight tensor is then partitioned into m blocks along the channel dimension, with each block fused with the inverse Hadamard matrix $\mathbf{H}_n^{-1}$.

Block-level rotation offers two benefits: (1) It redistributes outliers within only n channels, compressing the range of outliers without over-smoothing the long-tailed data distribution. (2) By applying a set of small Hadamard matrices with fewer total elements, it reduces substantial cross-channel arithmetic operations, thus effectively improving computational efficiency.

4.2 Sub-Block-Level Logarithmic Encoding

After block-level rotation, the outlier range within each block is compressed, while the data distribution retains a certain long-tailed characteristic suitable for logarithmic quantization. Since quantization with finer group sizes (e.g., 16 and 32) often outperforms that with coarser ones [12, 13, 16], each block is further partitioned into smaller sub-blocks of 16 contiguous values along the channel dimension. In each sub-block, all values share a scaling factor stored solely as an exponent, with the scaled values individually mapped to low-bit integers in the logarithmic domain. Consequently, the quantized representation of a value x_i consists of three parts: (1) a shared exponent E_s, which can be decoded as a scaling factor that scales all values strictly into [0,1); (2) the sign $\mathrm{Sgn}(x_i)$; and (3) a logarithmic code q_i. As shown in the left part of Fig. 5, the shared exponent is stored once per sub-block, whereas the sign and logarithmic code are maintained per value using N-bits.

Typically, the value before quantization is represented in floating-point format as $x_i = \mathrm{Sgn}(x_i) \cdot 2^{E_i} \cdot M_i$, where E_i is the exponent and M_i is the normalized mantissa with implicit leading 1. Therefore, the shared exponent E_s is obtained as one plus the maximum exponent $E_{\max}$ across all values within the sub-block, i.e., $E_s = E_{\max} + 1$, with an extra bit allocated to prevent overflow. Meanwhile, the scaled value $\tilde{x}_i \in [0, 1)$ can be easily calculated as:

$$\tilde{x}_i = \frac{|x_i|}{2^{E_s}} = M_i >> (E_{\max} + 1 - E_i), \tag{4}$$

where $>>$ is the right shift operation.

To map $\tilde{x}_i$ to the code q_i, the multi-base logarithmic encoding is adopted. As indicated by observation 2 in Section 3, adjusting the logarithmic base within a narrow range is sufficient after rotations. Therefore, we replace conventional layer-wise or tensor-wise base selection with a *dynamic allocation of quantization levels across multiple bases*. For hardware simplicity, we restrict the bases to 2 and $\sqrt{2}$, with their efficacy also supported in Fig. 3(b). A tunable integer L is introduced to govern the allocation of levels. As presented in the right part of Fig. 5, among the 2^{N-1} available levels, levels in $[0, L]$ are assigned to base $\sqrt{2}$, while those in $(L, 2^{N-1})$ are assigned to base 2. To avoid abrupt increase in encoding step size at the boundary between the two bases, an offset of $\frac{L}{2}$ is applied to base 2 conditions for smooth connection. Accordingly, the proposed logarithmic encoding function for $\tilde{x}_i \in [0, 1)$ can be formulated as follows:

$$q_i = \begin{cases} \mathrm{clamp}\Big(\lfloor -\log_{\sqrt{2}}(\tilde{x}_i) \rceil,\ 0,\ L\Big), & \tilde{x}_i \geq T \\ \mathrm{clamp}\Big(\lfloor \frac{L}{2} - \log_2(\tilde{x}_i) \rceil,\ L+1,\ 2^{N-1} - 1\Big), & \tilde{x}_i < T \end{cases} \tag{5}$$

where T determines which base to use for encoding $\tilde{x}_i$. To facilitate nearest-level mapping, it is defined as the arithmetic mean of the decoded values for level L (base $\sqrt{2}$) and $L+1$ (base 2), i.e., $T =$

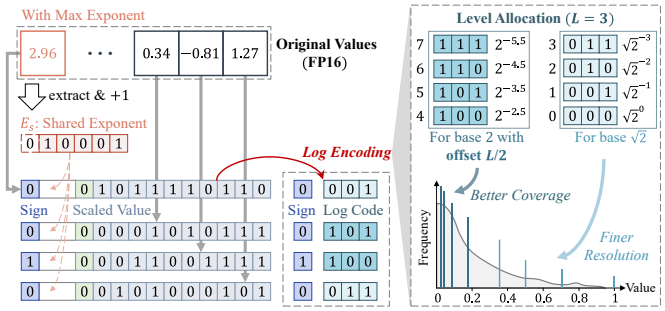


Figure 5: Sub-block-level logarithmic encoding.

$0.75 \cdot 2^{-L/2}$. For $\tilde{x}_i = 0$, q_i is clamped to level $2^{N-1} - 1$. Finally, the function for x_i reconstruction can be expressed as follows:

$$\widehat{x}_i = \begin{cases} \mathrm{Sgn}(x_i) \cdot 2^{E_s} \cdot \sqrt{2}^{-q_i}, & 0 \le q_i \le L \\ \mathrm{Sgn}(x_i) \cdot 2^{E_s} \cdot 2^{\frac{L}{2}-q_i}, & L < q_i < 2^{N-1} \end{cases} \tag{6}$$

With a tunable L, the above method can adaptively balance the resolution of small values and the granularity for larger values. Moreover, it reduces the alignment overhead for adjusting base within a wide range, while obviating the need for codebook generation and floating-point operations [18, 20] that are inefficient for online activation quantization.

4.3 Offline Search Strategy

Our proposed hierarchical quantization framework introduces two tunable parameters: (1) the rotation scope n, which is restricted to powers of 2; (2) the integer L for quantization level allocation. As discussed in Section 2.2, four rotation matrices $\{\mathbf{R}_1, \mathbf{R}_2, \mathbf{R}_3, \mathbf{R}_4\}$ participate in the computation process of a *transformer layer*. Therefore, corresponding rotation scopes $\{n_1, n_2, n_3, n_4\}$ are prepared for them. To balance the compression efficacy of outlier range and the computational overhead, we further restrict these rotation scopes to $2^e, e \in \{5, 6, 7, 8, 9, 10\}$. Notably, $\mathbf{R}_1$ is shared by the entire model and $\mathbf{R}_2 \sim \mathbf{R}_4$ are layer-specific [6]. In this case, the scope n_1 is a global parameter for the model, while $n_2 \sim n_4$ can be independently determined for each transformer layer. Moreover, the integer $L \in [0, 2^{N-1})$ is determined for each quantization-related tensor, where N is the bit width of quantized values.

Jointly optimizing these parameters across the entire model induces an intractably large search space. Advanced PTQ methods address this by optimizing one transformer layer at a time, where the optimally quantized output of each layer is propagated to the next [31, 32]. This sequential search over transformer layers mitigates the accumulation of quantization errors within each layer, while keeping our joint search over layer-wise $\{n_2, n_3, n_4\}$ and tensor-wise L tractable. For the global parameter n_1, which cannot be optimized in a layer-wise manner, we repeat the aforementioned sequential search over its 6 candidate values to identify the optimal configuration.

To improve search efficiency within a transformer layer l, we develop a search flow based on learning space partitions [33]. For better search stability, we reparameterize $\{n_2, n_3, n_4\}$ as their exponent $\{e_2, e_3, e_4\}$ during optimization. As a result, the search flow operates in search space $\Omega^{(l)} = \{L_1, ..., L_t, e_2, e_3, e_4\}$, where t is the number of quantization-related tensors in l. Over N_s calibration data, its objective is to minimize the mean square error (MSE) between the full precision layer output $\mathbf{Y}$ and the quantized layer output $\widehat{\mathbf{Y}}_\omega$ under the configuration $\omega \in \Omega^{(l)}$, i.e., $MSE(\omega) = \frac{1}{N_s}||\mathbf{Y} - \widehat{\mathbf{Y}}_\omega||_2^2$. Learning

Algorithm 1: Search with Learning Space Partitions

Input: Set of initial sampled configurations $\mathcal{P}_0$, iterarion limit $I_{\max}$.
Output: Optimal configuration ω_{best}.

1 **for** $i \le I_{max}$ **do**
  /* Step 1: Recursive Space Partitioning */
2 Reset tree $\mathcal{T}$ with root $\Omega_{\text{root}} \leftarrow \Omega^{(l)}$, queue $\mathcal{Q}$.**insert**(Ω_{root});
3 **while** $\mathcal{Q} \neq \emptyset$ **do**
4 $\Omega_j \leftarrow \mathcal{Q}$.**pop**(), $\mathcal{P}_{ij} \leftarrow \mathcal{P}_i \cap \Omega_j$;
5 **if** $\mathcal{P}_{ij}$ has sufficient configurations **then**
6 $\tau \leftarrow$ **median**($\{MSE(\omega) | \omega \in \mathcal{P}_{ij}\}$);
7 $\mathcal{P}^+_{ij} \leftarrow \{\omega \mid \omega \in \mathcal{P}_{ij},\ MSE(\omega) \le \tau\}$, $\mathcal{P}^-_{ij} \leftarrow \mathcal{P}_{ij} \setminus \mathcal{P}^+_{ij}$;
8 Train SVM with $\mathcal{P}^+_{ij}$ and $\mathcal{P}^-_{ij}$ to learn the boundary;
9 Expand $\mathcal{T}$ with Ω_L, Ω_R partitioned from Ω_j via SVM;
10 $\mathcal{Q}$.**insert**(Ω_L, Ω_R);
  /* Step 2: Subspace Selection */
11 $\Omega_c \leftarrow \Omega_{\text{root}}$;
12 **while** Ω_c is not a leaf of $\mathcal{T}$ **do**
13 $\Omega_1, \Omega_2 \leftarrow$ get the children of Ω_c;
14 $\Omega_c \leftarrow \arg\max_{\Omega \in \{\Omega_1, \Omega_2\}}$ **UCB**(Ω); // In Eq. (7)
  /* Step 3: Search within the Subspace */
15 $\mathcal{P}_{\text{new}} \leftarrow$ **DiffEvo**($MSE(\omega), \Omega_c$), $\mathcal{P}_{i+1} \leftarrow \mathcal{P}_i \cup \mathcal{P}_{\text{new}}$;
16 $\omega_{\text{best}} \leftarrow \arg\min_{\omega \in \mathcal{P}_{i+1}} MSE(\omega)$
17 **return** ω_{best}

space partitions aims to *identify the most promising subspace and concentrate the search within it*, thus effectively reducing the sampling cost for MSE minimization.

The search flow is detailed in Algorithm 1. It is initialized with a set of configurations $\mathcal{P}_0$ randomly sampled in $\Omega^{(l)}$, followed by a three-step iterative process: (1) recursive space partitioning with support vector machine (SVM), (2) subspace selection based on upper confidence bound (UCB), and (3) search within the selected subspace using discrete differential evolution [34]. Inspired by the Monte Carlo tree search framework in [33], we build a tree $\mathcal{T}$ that effectively assists in space partitioning and selection, where *each node is a subspace of* $\Omega^{(l)}$. In step 1 (Line 2-10), $\mathcal{T}$ is recursively grown from the root node representing $\Omega^{(l)}$. Specifically, for a given node Ω_j, we extract the set $\mathcal{P}_{ij}$ of configurations sampled within it. If $\mathcal{P}_{ij}$ contains sufficient configurations, we divide it into two subsets, $\mathcal{P}^+_{ij}$ and $\mathcal{P}^-_{ij}$ based on the median MSE τ. Both subsets are used to train the SVM classifier. At this point, the SVM can partition Ω_j into promising and non-promising subspaces, which naturally expands $\mathcal{T}$ with two leaf nodes, denoted Ω_L and Ω_R, respectively.

To select the smallest subspace (i.e., a leaf node) worthy of further search, step 2 (Line 11-14) traverses $\mathcal{T}$ from the root by sequentially choosing the child node with better UCB:

$$\mathrm{UCB}(\Omega) = -\frac{1}{N_\omega} \sum_{\omega \in \Omega} MSE(\omega) + C\sqrt{(\ln N^p_\omega)/N_\omega}, \tag{7}$$

where N_ω and N^p_ω are the numbers of configurations in node Ω and its parent node, respectively. C denotes the exploration parameter.

In step 3 (Line 15-16), discrete differential evolution is employed for efficient search within the selected subspace Ω_c. Firstly, Latin hypercube sampling [35] expands sampled configurations in Ω_c to enhance coverage. The search is then driven by mutation. It generates offspring ω_{new} from a parent ω_m through DE/current-to-pbest: $\omega_{\text{new}} = \mathrm{round}(\omega_m + F_m \cdot (\omega_{\text{pbest}} - \omega_m) + F_m \cdot (\omega_{r1} - \omega_{r2}))$, where ω_{pbest} is selected from the top 10% configurations with the lowest MSE, ω_{r1},

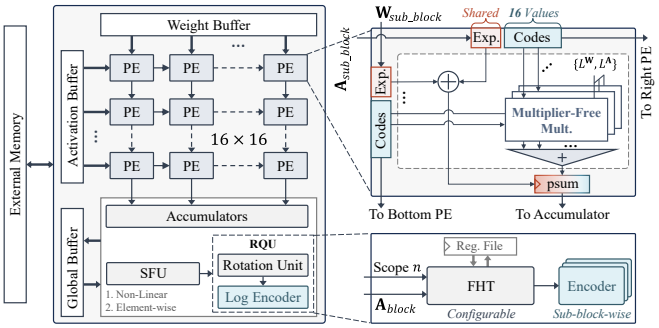


Figure 6: Architecture of PeLog accelerator.

ω_{r2} are exclusive random configurations, and F_m is the differential weight. Subsequently, binomial crossover recombines the parameters from offspring ω_{new} and parent ω_m with probability CR_m. Both F_m and CR_m are self-adaptive following [36]. Notably, all out-of-bound offspring are strictly clipped to the nearest integer boundaries in Ω_c.

5 PeLog Architecture

To leverage the hardware efficiency of the PeLog algorithm, we design an accelerator based on a systolic array architecture [13], and carefully modify it to align with our quantization flow. It integrates multiplier-free PE (Section 5.1), along with real-time quantization unit for logarithmic encoding and configurable rotation (Section 5.2), effectively supporting LLM inference under 4-bit quantization.

Fig. 6 illustrates the architecture overview of the PeLog accelerator. It employs a 16×16 systolic PE array operating in an output-stationary manner, where each PE computes a 16-way dot product without multipliers. Quantized tensors are stored in weight and activation buffers. Tensor-wise integers L for quantization level allocation are stored in a global register and broadcast to all PEs. Weight tensors are rotated and quantized offline, while activation tensors are processed by the real-time quantization unit (RQU). In RQU, a configurable rotation unit performs the block-level fast Hadamard transform with rotation scope n, while logarithmic encoders conduct 4-bit group quantization at a finer sub-block-level. Each PE receives weight and activation sub-blocks together with L to perform the 16-way dot product with integrated dequantization. The resulting partial sums are collected by the accumulator for post-processing. The special function unit (SFU) performs non-linear and element-wise operations.

5.1 PeLog Processing Element

The PeLog PE, as shown in Fig. 7, performs a multiplier-free 16-way dot product through three steps: ① logarithmic-domain addition, ② log-to-linear conversion, and ③ linear-domain summation. The first two steps are executed in a 16-way parallel manner, and the last step is carried out by an adder tree. According to Eqs. (5) and (6), our encoding employs bases $\sqrt{2}$ and 2, with an offset of $\frac{L}{2}$ introduced for base 2 encoding. Therefore, step ① requires base alignment and offset compensation before performing the addition.

In step ①, for each weight-activation pair, their sign bits Sgn_i^a and Sgn_i^w are fed into an XOR gate to determine the sign of product Sgn_i^P. Meanwhile, the logarithmic code q of activation/weight and the tensor-wise L are passed to a preprocessing unit (P-unit). The P-unit employs a comparator to determine the encoding base of the code and processes it accordingly. If $q \le L$, the code is encoded with base $\sqrt{2}$. To align it with the base 2 domain, the P-unit right-shifts it by one bit through fixed wiring. Otherwise, the code corresponds to base 2,

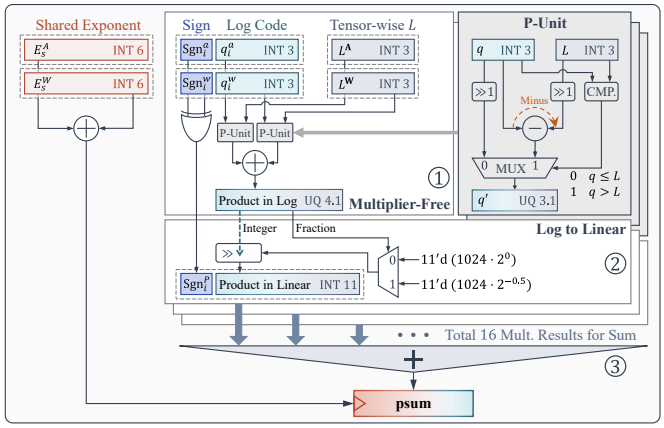


Figure 7: The multiplier-free PE for 16-way dot product.

and the offset $\frac{L}{2}$ is subtracted from it. With an input logarithmic code of 3-bit, the P-unit outputs a fixed-point number with a 3-bit integer part and a 1-bit fractional part (i.e., UQ3.1 format). As the processed logarithmic codes of the activation and weight share the same fixed-point format, they are summed by an integer adder without fractional alignment, thus yielding the product in the logarithmic domain.

In step ②, each product P in the logarithmic domain is converted back into the linear domain via 2^{-P}. Since P is unified into the base 2 domain, the conversion is achieved by right-shifting the exponential of its fraction part P_{frac} by the integer part P_{int}, i.e., $2^{-P_{\text{frac}}} >> P_{\text{int}}$. Benefiting from our encoding, P_{frac} contains a single bit, allowing a 2-to-1 MUX to directly select between 2^0 and $2^{-0.5}$. To facilitate subsequent intra-PE summation and conversion to FP16 during accumulation, these two constants are pre-scaled into 11-bit integers (1024 and 724). The selected integer is then right-shifted by P_{int} and truncated to 11-bit, thus obtaining the linear-domain product.

In step ③, 16 linear-domain products are summed by a 4-level adder tree, which progressively halves the operands by pairwise addition to obtain the partial sum. Notably, the partial sum is associated with a scaling factor stored in exponent form, which can be easily calculated by summing the shared exponents of the weight and activation sub-blocks. This exponent-based representation simplifies subsequent normalization, alignment and bias correction in the accumulator.

5.2 Real-time Quantization Unit

The real-time quantization unit consists of a configurable rotation unit and logarithmic encoders. The rotation unit supports rotations with scope $n = 2^e, e \in \{5, 6, 7, 8, 9, 10\}$, while each logarithmic encoder performs group quantization over 16 values per sub-block.

The rotation unit is designed based on the fast Hadamard transform (FHT). We achieve configurability by reusing a single 5-stage FHT core. The core follows the design in [7] and integrates 32 add/subtract units per stage, enabling the 5-stage pipeline to compute a 32-way FHT defined by $\mathbf{H}_{32}$ in one pass. Leveraging Kronecker decomposition $\mathbf{H}_{2^e} = \mathbf{H}_{2^{e-5}} \otimes \mathbf{H}_{32}$, the full FHT is realized through two-pass reuse of the core. In the first pass, the core activates 5 stages to compute the FHT defined by $\mathbf{H}_{32}$ and stores the results in the register file. In the second pass, the results are read back with a stride of 32. Partial gating activates the first $e - 5$ stages of the core, each with 2^{e-5} add/subtract units, to complete the FHT defined by $\mathbf{H}_{2^{e-5}}$.

As shown in Fig. 8, the logarithmic encoder follows the encoding flow in Section 4.2, including shared exponent extraction, value scaling, and mapping. First, the exponents of FP16 activations within

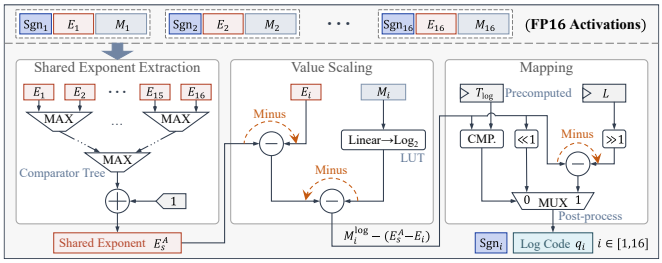


Figure 8: The logarithmic encoder of PeLog accelerator.

each sub-block are processed by a comparator tree to identify their maximum. The shared exponent E_s^A for value scaling is obtained by adding 1 to the maximum. To avoid costly barrel shifters, the shift-based scaling in Eq. (4) is converted to subtraction in the logarithmic domain. Specifically, the mantissa M_i of each activation is converted to a fixed-point logarithmic value $M_i^{\log} = \log_2(M_i)$ by Mitchell approximation with simple LUT-based correction [37]. In this case, the scaled value can be obtained with lightweight subtractors, i.e., $M_i^{\log} - (E_s^A - E_i)$. Accordingly, the threshold T in Eq. (5) is precomputed as a logarithmic value $T_{\log}$, matching the format of the scaled value. The MUX then determines the mapping result: if the scaled value is not smaller than $T_{\log}$, it is left-shifted by one bit to the base $\sqrt{2}$ domain; otherwise, an offset $\frac{L}{2}$ is subtracted from it. Finally, the sign of the mapping result is discarded, followed by adding 0.5 and truncating to an integer for logarithmic code generation.

6 Experiments

Evaluation Scenarios. We evaluate PeLog on four representative open-source LLMs of different scales from two families, including LLAMA2-7B, LLAMA2-13B [2], LLAMA3.2-3B, and LLAMA3.1-8B [3]. The performance of quantized models are assessed on downstream tasks and language modeling. For task-level evaluation, we conduct zero-short inference on Arc-easy/challenge [38], BoolQ [39], PIQA [40], and WinoGrande[41], which reflect the model's performance on tasks such as commonsense reasoning and causal inference. To evaluate the language modeling capability, we measure the perplexity (PPL) of models on WikiText-2 dataset [42]. These evaluations are conducted on two NVIDIA RTX 6000 Ada GPUs with a total 96GB VRAM.

Quantization Method. At the algorithmic level, we compare PeLog with three rotation-based linear quantization methods: (1) SpinQuant [6], (2) Duquant [8], and (3) LightRot [7]. SpinQuant integrates global rotation with asymmetric group quantization and improves quality through trainable rotations. DuQuant employs a two-stage local rotation with an intermediate permutation to enhance smoothing capability. LightRot combines local rotation featuring a fixed scope of 128 with asymmetric group quantization. Given the lack of prior low-bit logarithmic quantization methods for LLMs, we adapt AdaLog [20] (a multi-base method originally for ViTs) as a baseline and enhance it with group-wise scaling. We also include a variant of PeLog *PeLog-G* that uses greedy search to optimize quantization configurations.

As most of the aforementioned methods tune the quantization configurations during PTQ, we randomly select 128 samples from the Pile dataset [43] for calibration. Since sampling affects the searched configurations, all evaluation results are averaged over 10 runs. The offline search in Section 4.3 is initialized with 50 configurations and runs up to 60 iterations if not converged. The SVM for space partitioning employs a polynomial kernel with degree 4, UCB constant C for subspace selection is $0.15 \times (MSE_{\max} - MSE_{\min})$, and the population size of differential evolution for subspace search is 10.

Table 1: Zero-shot accuracy (↑) of different quantization methods on downstream tasks under the W4A4KV4 setting.

Model	Method	Arc-e.	Arc-c.	BoolQ	PIQA	WinoG.
LLAMA2-7B (q: 4-4-4)	FP16	75.23	46.41	77.18	79.06	69.53
	SpinQuant	72.18	41.70	73.85	77.19	66.02
	DuQuant	66.01	39.32	71.44	76.36	64.79
	LightRot	72.73	43.69	75.60	78.18	67.72
	AdaLog	59.28	31.91	68.55	65.03	55.24
	PeLog-G	71.90	42.96	75.05	77.52	68.38
	PeLog	**73.35**	**44.01**	**76.13**	**78.20**	**69.01**
LLAMA2-13B (q: 4-4-4)	FP16	77.96	51.09	80.02	80.33	72.12
	SpinQuant	76.32	47.53	79.44	78.17	67.99
	DuQuant	71.83	46.12	76.30	78.05	68.18
	LightRot	76.47	**48.72**	77.68	**79.33**	70.96
	AdaLog	61.80	39.34	70.31	68.28	59.00
	PeLog-G	76.35	47.61	77.90	78.33	70.91
	PeLog	**77.03**	47.95	**78.24**	79.19	**71.74**
LLAMA3.2-3B (q: 4-4-4)	FP16	68.92	47.55	79.10	76.12	66.80
	SpinQuant	64.04	41.98	75.38	74.76	63.59
	DuQuant	61.82	42.07	71.15	74.22	62.06
	LightRot	66.09	43.17	75.72	**75.10**	64.92
	AdaLog	49.75	29.90	66.83	58.03	49.11
	PeLog-G	65.69	42.63	76.09	74.78	65.30
	PeLog	**66.83**	**43.80**	**77.33**	74.92	**66.09**
LLAMA3.1-8B (q: 4-4-4)	FP16	77.71	54.26	83.11	80.72	73.30
	SpinQuant	75.02	47.29	76.90	79.00	68.18
	DuQuant	71.00	44.62	76.31	77.35	67.92
	LightRot	**78.49**	49.40	79.66	80.25	70.09
	AdaLog	63.31	40.83	70.19	66.10	59.61
	PeLog-G	77.09	49.16	78.28	78.13	69.44
	PeLog	77.63	**51.24**	**80.57**	**80.30**	**71.67**

Hardware Implementation. The PeLog PEs, the real-time quantization unit, and associated components in Section 5 are implemented in Verilog. They are synthesized by Synopsys Design Compiler [44] using industrial 28nm technology to obtain power and area statistics. The latency and power of the on-chip buffers are estimated using CACTI [45]. Additionally, following the evaluation methodology in [46], we develop a cycle-accurate simulator to evaluate the end-to-end performance of the PeLog architecture and related baselines.

At the hardware level, we compare PeLog with: (1) Tender [24] and (2) LightRot [7]. As a representative LLM accelerator, Tender isolates outliers by grouping channels, and uses shifts for inter-group scaling to mitigate floating-point overheads. LightRot is effective due to local rotation and asymmetric group quantization, but relies on floating-point operations for scaling and zero point compensation. For a fair comparison, all designs are synthesized using 28nm technology at 1GHz with a 0.9V supply voltage, and evaluated under iso-compute-area constraints, consistent with the setup of Tender [24].

6.1 Accuracy Analysis

Task-level Evaluation. We compare the zero-shot accuracy of different methods on downstream tasks under the W4A4KV4 precision, with the results summarized in Table 1. Meanwhile, the quantization settings of these methods, such as the precision of scaling factor (SF) and zero point (ZP), are detailed in the left part of Table 2.

According to the results in Table 1, our PeLog incurs negligible accuracy loss across different models compared to the full-precision baseline. In most cases, PeLog achieves higher accuracy than the leading rotation-based method LightRot, particularly on BoolQ and WinoGrande tasks. This demonstrates the effectiveness of logarithmic encoding on data after local rotation. Although AdaLog [20] without rotation provides fine-grained adjustment of the logarithmic base,

Table 2: Comparison of perplexity (↓) on WikiText-2 dataset and quantization settings across different methods. The gray numbers denote the perplexity variation caused by the random sampling of calibration data for configuration search.

Method	Precision Settings W-A-KV	Scaling Factor	Zero Point	Rotation	LLAMA2-7B	LLAMA2-13B	LLAMA3.2-3B	LLAMA3.1-8B
FP16	16-16-16	–	–	–	5.51 ±0.00	4.90 ±0.00	10.72 ±0.00	6.13 ±0.00
Tender	8-8-16	FP16	–	–	5.79 ±0.03	5.06 ±0.03	11.31 ±0.08	6.86 ±0.02
AdaLog	6-6-6	FP16	–	–	9.03 ±0.04	6.61 ±0.01	18.08 ±0.10	9.58 ±0.02
Tender	4-4-16	FP16	–	–	36.44 ±0.09	53.90 ±0.31	401.96 ±6.20	31.73 ±0.11
SpinQuant	4-4-4	FP16	FP16	✓	5.95 ±0.03	5.40 ±0.02	11.39 ±0.04	7.40 ±0.02
DuQuant	4-4-4	FP16	FP16	✓	6.10 ±0.02	5.38 ±0.02	12.05 ±0.05	7.91 ±0.04
LightRot	4-4-4	FP16	FP16	✓	5.71 ±0.02	5.09 ±0.02	11.14 ±0.03	6.96 ±0.02
AdaLog	4-4-4	FP16	–	–	12.72 ±0.05	7.58 ±0.05	35.39 ±0.12	14.46 ±0.04
AdaLog-R	4-4-4	FP16	–	✓	5.90 ±0.05	5.36 ±0.03	11.47 ±0.07	7.37 ±0.03
PeLog-G	4-4-4	INT6	–	✓	5.71 ±0.03	5.28 ±0.04	11.24 ±0.07	7.22 ±0.04
PeLog	4-4-4	INT6	–	✓	5.66 ±0.02	5.11 ±0.01	10.97 ±0.04	6.92 ±0.01

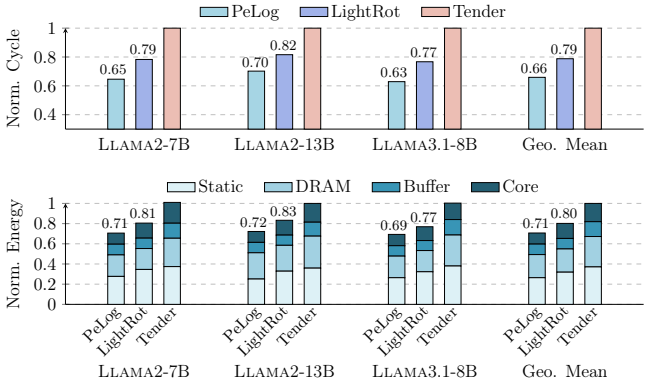


Figure 9: Comparison of normalized latency (measured by cycle count) and energy consumption in different accelerators.

it exhibits the worst accuracy across all tasks. This indicates that rotation is essential for logarithmic encoding to mitigate outliers spanning a wide range. Moreover, PeLog outperforms its greedy search variant PeLog-G, highlighting the advantage of our offline sequential search with learning space partitions.

Evaluation of Language Modeling Capability. Table 2 presents the PPL comparison of different methods on WikiText-2 dataset. We also list the PPL variation across 10 runs to show the stability of the configuration search. To broaden the evaluation, we further introduce Tender [24] at 8-bit and 4-bit precisions, along with *AdaLog-R*, an AdaLog variant integrating our tunable local rotation. According to the results, our PeLog maintains low PPL across all cases, especially on LLAMA2-7B and LLAMA3.2-3B, where it outperforms 8-bit Tender. Compared to LightRot, PeLog achieves better or comparable PPL while eliminating the need for floating point SF and ZP.

While AdaLog struggles at 4 and 6 bits, AdaLog-R achieves a substantial reduction in PPL thanks to our tunable local rotation. By dynamically allocating quantization levels to logarithmic bases, PeLog still outperforms AdaLog-R without complex base adjustment. In addition, PeLog exhibits minimal PPL variation, which further reflects the stability of our offline sequential search.

6.2 Performance, Power and Area Evaluation

We evaluate the performance and energy consumption of PeLog accelerator against LightRot and Tender, with all designs operating at 4-bit precision. Fig. 9 shows the results normalized to Tender. To isolate outliers, Tender employs serialized sub-tensor execution and indirect memory indexing, which disrupts computational continuity

Table 3: Area and power breakdown of PeLog accelerator.

Component	Description	Area (mm^2)	Power (mW)
PE Array	16×16 PEs & psum accum.	0.36 (17.1%)	382.53 (66.5%)
Rotation Unit	Configurable FHT	0.05 (2.4%)	9.06 (1.6%)
Log Encoder	1×16 group encoding	0.02 (1.0%)	11.83 (2.1%)
On-chip Buffer	W/A: 64+8KB, global: 256KB	1.40 (66.7%)	157.29 (27.4%)
Others	SFU, controller, etc.	0.27 (12.9%)	14.07 (2.4%)

and incurs the highest latency. Benefiting from elegant outlier mitigation via rotation, both PeLog and LightRot significantly reduce latency. By eliminating multipliers and floating-point dequantization, PeLog packs more PEs under the iso-compute-area constraint to enhance performance. Consequently, PeLog achieves 1.52× and 1.20× faster inference over Tender and LightRot, respectively. In addition, PeLog achieves energy savings of 29% over Tender and 11% over LightRot. The computational simplicity of PeLog enables a streamlined pipeline and reduced hardware complexity, effectively lowering static and core energy. While PeLog avoids high-precision SF and ZP, its fine-grained blocking incurs higher DRAM and buffer energy than LightRot. This is a worthy trade-off for the better accuracy.

The area and power distribution of PeLog accelerator is summarized in Table 3. The PEs and accumulators for matrix multiplication dominate the power consumption with a share of 66.5%. Rotation unit and 16 logarithmic encoders for real-time quantization incurs only 3.4% area and 3.7% power overhead. The on-chip buffers occupy the majority of area at 66.7%, comprising a 256KB global buffer and weight/activation buffers, each with 64KB for values and 8KB for shared exponents. The remaining area and power are consumed by common components like special function unit and controller.

7 Conclusion

In this work, we propose PeLog, an algorithm-hardware co-design framework that unlocks low-bit logarithmic quantization for LLM inference via rotation. By integrating tunable local rotations with group logarithmic quantization at a finer granularity, PeLog mitigates outliers and avoids distribution mismatch caused by over-smoothing. Leveraging the homogenizing effect of rotation, Pelog dynamically allocates quantization levels within a narrow range of logarithmic bases, further improving accuracy and hardware efficiency. In addition, we design a tailored offline search strategy and accelerator architecture to facilitate the practical use of PeLog. Experimental results show that PeLog incurs negligible quantization loss for 4-bit LLM inference, while achieving 1.52× performance improvement and 29% energy savings over the baseline accelerator.

References

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," in *Proc. NIPS*, 2020.

[2] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[3] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[4] S. Kim, Y. Chou, B. Kim, J. Oh, and H.-J. Yoo, "GyRot: Leveraging hidden synergy between rotation and fine-grained group quantization for low-bit LLM inference," in *Proc. HPCA*, 2026.

[5] S. Ashkboos, A. Mohtashami, M. L. Croci, B. Li, P. Cameron, M. Jaggi, D. Alistarh, T. Hoefler, and J. Hensman, "QuaRot: Outlier-free 4-bit inference in rotated LLMs," in *Proc. NIPS*, 2024.

[6] Z. Liu, C. Zhao, I. Fedorov, B. Soran, D. Choudhary, R. Krishnamoorthi, V. Chandra, Y. Tian, and T. Blankevoort, "SpinQuant: LLM quantization with learned rotations," *arXiv preprint arXiv:2405.16406*, 2024.

[7] S. Kim, Y. Choi, J. Oh, B. Kim, and H.-J. Yoo, "LightRot: A light-weighted rotation scheme and architecture for accurate low-bit large language model inference," *IEEE JETCAS*, vol. 15, no. 2, pp. 231–243, 2025.

[8] H. Lin, H. Xu, Y. Wu, J. Cui, Y. Zhang, L. Mou, L. Song, Z. Sun, and Y. Wei, "DuQuant: Distributing outliers via dual transformation makes stronger quantized LLMs," in *Proc. NIPS*, 2024.

[9] Z. Su, Z. Chen, W. Shen, H. Wei, L. Li, H. Yu, and K. Yuan, "RotateKV: Accurate and robust 2-bit KV cache quantization for LLMs via outlier-aware adaptive rotations," *arXiv preprint arXiv:2501.16383*, 2025.

[10] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022.

[11] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration," in *Proc. MLSys*, 2024.

[12] L. Zou, W. Zhao, S. Yin, C. Bai, Q. Sun, and B. Yu, "BiE: Bi-exponent block floating-point for large language models quantization," in *Proc. ICML*, 2024.

[13] L. Zou, S. Yin, M. Li, M. Wang, C. Bai, W. Zhao, and B. Yu, "Oiso: Outlier-isolated data format for low-bit large language model quantization," *IEEE TCAD*, vol. 45, no. 2, pp. 929–942, 2025.

[14] Y. Li, R. Yin, D. Lee, S. Xiao, and P. Panda, "GPTAQ: Efficient finetuning-free quantization for asymmetric calibration," *arXiv preprint arXiv:2504.02692*, 2025.

[15] B. Chmiel, M. Fishman, R. Banner, and D. Soudry, "FP4 all the way: Fully quantized training of LLMs," *arXiv preprint arXiv:2505.19115*, 2025.

[16] W. Hu, H. Zhang, C. Guo, Y. Feng, R. Guan, Z. Hua, Z. Liu, Y. Guan, M. Guo, and J. Leng, "M-ANT: Efficient low-bit group quantization for LLMs via mathematically adaptive numerical type," in *Proc. HPCA*, 2025.

[17] X. Geng, Y. Lu, H. Wang, and H. Jiang, "Segment-wise accumulation: Low-error logarithmic domain computing for efficient large language model inference," in *Proc. DATE*, 2025.

[18] J. Xu, Y. Zheng, C. Sun, T. Zhou, Z. Zhang, J. Li, L. Zheng, and Z. Zou, "LogART: Pushing the limit of efficient logarithmic post-training quantization," in *Proc. ICLR*, 2026.

[19] J. Zhao, S. Dai, R. Venkatesan, B. Zimmer, M. Ali, M.-Y. Liu, B. Khailany, W. J. Dally, and A. Anandkumar, "LNS-Madam: Low-precision training in logarithmic number system using multiplicative weight update," *IEEE TC*, vol. 71, no. 12, pp. 3179–3190, 2022.

[20] Z. Wu, J. Chen, H. Zhong, D. Huang, and Y. Wang, "AdaLog: Post-training quantization for vision transformers with adaptive logarithm quantizer," in *Proc. ECCV*, 2024.

[21] B. Li, P. Jiang, J. Wang, H. Zhou, and Y. Ge, "Self-learning logarithm with dual shift-ln-log2: An activation quantization for large language models," in *Proc. AANN*, 2025.

[22] A. Ramachandran, Z. Wan, G. Jeong, J. Gustafson, and T. Krishna, "Algorithm-hardware co-design of distribution-aware logarithmic-posit encodings for efficient DNN inference," in *Proc. DAC*, 2024.

[23] Y. An, X. Zhao, T. Yu, M. Tang, and J. Wang, "Systematic outliers in large language models," *arXiv preprint arXiv:2502.06415*, 2025.

[24] J. Lee, W. Lee, and J. Sim, "Tender: Accelerating large language models via tensor decomposition and runtime requantization," in *Proc. ISCA*, 2024.

[25] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale," in *Proc. NIPS*, 2022.

[26] J. H. Heo, J. Kim, B. Kwon, B. Kim, S. J. Kwon, and D. Lee, "Rethinking channel dimensions to isolate outliers for low-bit weight quantization of large language models," *arXiv preprint arXiv:2309.15531*, 2023.

[27] Y. Zhao, C.-Y. Lin, K. Zhu, Z. Ye, L. Chen, S. Zheng, L. Ceze, A. Krishnamurthy, T. Chen, and B. Kasikci, "Atom: Low-bit quantization for efficient and accurate LLM serving," in *Proc. MLSys*, 2024.

[28] C. Lee, J. Jin, T. Kim, H. Kim, and E. Park, "OWQ: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models," in *Proc. AAAI*, 2024.

[29] K. Agarwal, R. Astra, A. Hoque, M. Srivatsa, R. Ganti, L. Wright, and S. Chen, "Hadacore: Tensor core accelerated hadamard transform kernel," *arXiv preprint arXiv:2412.08832*, 2024.

[30] V. M. Panaretos and Y. Zemel, "Statistical aspects of wasserstein distances," *Annu. Rev. Stat. Appl.*, vol. 6, no. 1, pp. 405–431, 2019.

[31] Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, and S. Gu, "BRECQ: Pushing the limit of post-training quantization by block reconstruction," *arXiv preprint arXiv:2102.05426*, 2021.

[32] W. Shao, M. Chen, Z. Zhang, P. Xu, L. Zhao, Z. Li, K. Zhang, P. Gao, Y. Qiao, and P. Luo, "OmniQuant: Omnidirectionally calibrated quantization for large language models," *arXiv preprint arXiv:2308.13137*, 2023.

[33] L. Wang, R. Fonseca, and Y. Tian, "Learning search space partition for black-box optimization using monte carlo tree search," in *Proc. NIPS*, 2020.

[34] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Trans. Evol. Comput.*, vol. 15, no. 1, pp. 4–31, 2010.

[35] B. G. Husslage, G. Rennen, E. R. Van Dam, and D. Den Hertog, "Space-filling latin hypercube designs for computer experiments," *Optim. Eng.*, vol. 12, no. 4, pp. 611–630, 2011.

[36] J. Zhang and A. C. Sanderson, "JADE: adaptive differential evolution with optional external archive," *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 945–958, 2009.

[37] Y. Wang, D. Deng, L. Liu, S. Wei, and S. Yin, "PL-NPU: An energy-efficient edge-device DNN training processor with posit-based logarithm-domain computing," *IEEE TCAS I*, vol. 69, no. 10, pp. 4042–4055, 2022.

[38] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have solved question answering? try arc, the ai2 reasoning challenge," *arXiv preprint arXiv:1803.05457*, 2018.

[39] C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova, "BoolQ: Exploring the surprising difficulty of natural yes/no questions," in *Proc. NAACL*, 2019.

[40] Y. Bisk, R. Zellers, J. Gao, Y. Choi *et al.*, "PIQA: Reasoning about physical commonsense in natural language," in *Proc. AAAI*, 2020.

[41] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, "WinoGrande: An adversarial winograd schema challenge at scale," *Commun. ACM*, vol. 64, no. 9, pp. 99–106, 2021.

[42] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[43] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima *et al.*, "The Pile: An 800gb dataset of diverse text for language modeling," *arXiv preprint arXiv:2101.00027*, 2020.

[44] Synopsys, "Design compiler user guide," https://www.synopsys.com/zh-cn/implementation-and-signoff/rtl-synthesis-test/design-compiler-graphical.html, 2023.

[45] N. Muralimanohar, R. Balasubramonian, N. P. Jouppi *et al.*, "CACTI 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.

[46] C. Guo, C. Zhang, J. Leng, Z. Liu, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "ANT: Exploiting adaptive numerical data type for low-bit deep neural network quantization," in *Proc. MICRO*, 2022.