BiE: Bi-Exponent Block Floating-Point for Large Language Models Quantization

Lancheng Zou [1] **Wenqian Zhao** [1] **Shuo Yin** [1] **Chen Bai** [1] **Qi Sun** [2] **Bei Yu** [1]

Abstract

Nowadays, Large Language Models (LLMs) mostly possess billions of parameters, bringing significant challenges to hardware platforms. Although quantization is an efficient approach to reduce computation and memory overhead for inference optimization, we stress the challenge that mainstream low-bit quantization approaches still suffer from either various data distribution outliers or a lack of hardware efficiency. We also find that low-bit data format has further potential expressiveness to cover the atypical language data distribution. In this paper, we propose a novel numerical representation, Bi-Exponent Block Floating Point (BiE), and a new quantization flow. BiE quantization shows accuracy superiority and hardware friendliness on various models and benchmarks.

1. Introduction

Large language models (LLMs) (Brown et al., 2020; Zhang et al., 2022a) exhibit superior efficacy in a multitude of tasks within the domain of natural language processing (NLP). Furthermore, LLMs' remarkable language comprehension capabilities can be effectively extended to multi-modal tasks (Liu et al., 2023; Li et al., 2023). Nevertheless, LLMs' computational and memory requirements are significant factors that affect their sustained use and further development. For example, the GPT-3 model (Brown et al., 2020), which contains 175B parameters, requires 350GB memory to load and at least 5×A100-80G GPUs for inference.

The importance of model compression techniques for efficient large model deployment is increasing (Han et al., 2015; Hinton et al., 2015). Among these techniques, quantization (Jacob et al., 2018) has surfaced as a prevalent and fundamental methodology for model compression. Quantizing the weight and activation into a lower-bit numerical representation format can reduce the memory footprint and computational overhead.

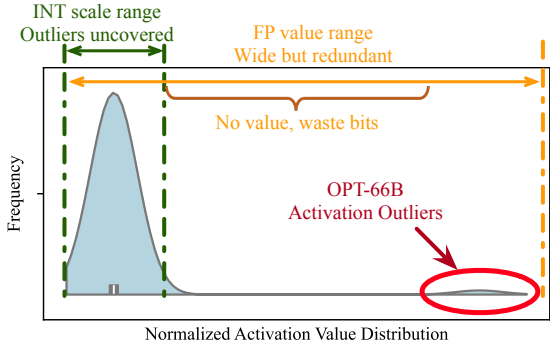


Figure 1. Normalized value distribution of input activations in one Linear Layer of OPT-66B model. Integer quantization may not cover outliers due to limited value range. Floating Point may result in redundant bit number and complex calculation.

Benefiting from GPU native support, fixed point quantization like INT8 is the mainstream for prior quantization works. Moreover, due to limited value range of INT type, some choose different scaling factors at a finer granularity than per-tensor or per-token level (Dettmers et al., 2022; Yao et al., 2022), (Zhang et al., 2023) systemically investigates block-based arithmetic quantization. Among the block-based numerical representations, the 6-bit block floating point (BFP), which is a fixed-point-variant, shows promising potential for LLMs quantization without any complex transformation or shifting operations. However, we found that BFP can not maintain accuracy in 4-bit and lower-bit LLMs quantization, which is a result of the homogeneity of its numerical representation. Inspired by these observations, we theoretically investigate the properties of BFP and claim that fixed point inherently cannot handle superposition of distributions in LLM activition scenario, as shown in Figure 1.

However, scaling up LLMs beyond 6.7B parameters will lead to the emergence of systematic *outliers* with large magnitudes in activations (Dettmers et al., 2022), which makes quantization for LLMs challenging, as visualized in Figure 1. To prevent quantization errors and performance degradation, LLM.int8() (Dettmers et al., 2022) introduces

[1]The Chinese University of Hong Kong, China [2]Zhejiang University, China. Correspondence to: Qi Sun <qisunchn@zju.edu.cn>, Bei Yu <byu@cse.cuhk.edu.hk>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

a mixed-precision approach that keeps outliers in FP16 and quantizes other activations in INT8. ZeroQuant (Yao et al., 2022) proposes a fine-grained quantization scheme that applies dynamic per-token activation quantization and group-wise weight quantization. Smoothquant (Xiao et al., 2023) smooths the magnitude across the channels using a mathematically equivalent scaling transformation between activation and weight to reduce quantization errors. Such a method can make the model quantization-friendly. Some outliers may concentrate on specific channels and are asymmetric across channels. Outlier Suppression+ (Wei et al., 2023) adopts channel-wise shifting and scaling to handle the problem. Orthogonally, some weight-only quantization approaches preserve activations in FP16 (Frantar et al., 2022; Lin et al., 2023).

In general, these approaches may still suffer from the data format restriction. We propose Bi-Exponent Block Floating Point (BiE), a novel numerical representation to achive the mutual optimality in both efficiency and accuracy. BiE contains a bi-shared exponent and is designed to be more robust to represent a larger dynamic range covering the data distributions of LLMs than BFP with tiny hardware overhead. For the vanilla BFP, when converting FP16 into BFP, the encoder will select the max exponent within the block/group as the shared exponent of this block/group. With the emergence of outliers in LLMs, once there is an outlier in this block, other normal values will be transformed into zero directly since the magnitude of the outlier value is much larger than the other values, leading to significant performance degradation. Building upon the vanilla BFP, BiE uses a bi-shared exponent, one for the normal values part and the other for the outlier values part, which will be distinguished by a pre-determined threshold value. In this way, BiE can maintain accuracy even if some outlier values are in the block.

The contributions of our paper include:

- We propose a novel numerical representation, Bi-Exponent Block Floating Point (BiE), tackling the drawbacks of current quantization methods on LLMs from the various data characteristics.
- We analyze that BiE shows advantages of data efficiency and quantization error reduction and beats SOTA baselines.
- We propose an offline thresholding optimization strategy to enhance the BiE encoding flow with Bayesian Optimization.
- We implement the BiE hardware design to validate the hardware efficiency of BiE. The simulation results demonstrate that BiE W4A4 quantization configuration can obtain $3.51\times$ computation- and $2.8\times$ memory-efficiency improvements compared with FP16.

2. Preliminaries

2.1. Quantization

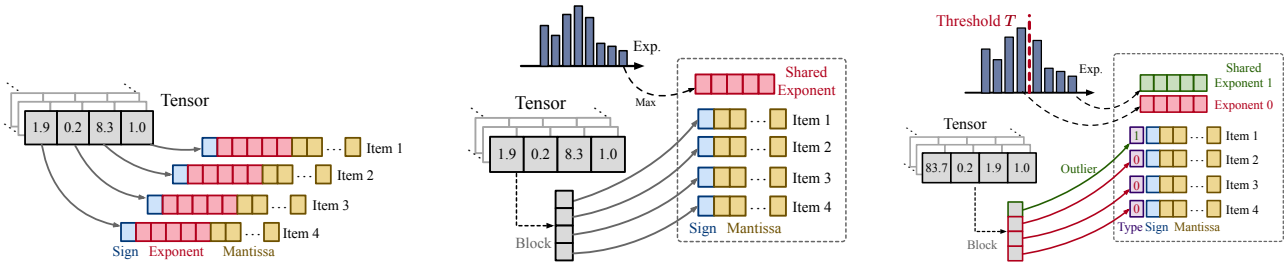
Current mainstream quantization approaches (Liu et al., 2021; Johnson, 2018; Lin et al., 2016; Nagel et al., 2020a; Kim et al., 2021) focus on lower-bit data types either in floating point such as FP16, FP8 or fixed point INT8, INT4, etc. Most popular quantization approaches in recent years revolve around uniform integer quantization (Choi et al., 2018; Nagel et al., 2020a) with the same quantization function composed of rounding and scaling operations:

$$Q(r) = \text{Round}(\frac{r}{S}), \tag{1}$$

where r is the original real number and $Q(r)$ is the quantized number. S is the scaling factor, a real number that maps r to an integer number, Round() is the rounding function. Other quantization methods include non-uniform quantization, where the value range of each projected quantized number is not equally assigned. Although non-uniform quantization may be accuracy-friendly for its flexibility with data distribution, implementing the quantization and bit assignment is not trivial and requires special design on hardware and, thus, not that common in real applications.

Apart from that, quantization flows are usually classified into two categories: Post-Training Quantization (PTQ) (Nagel et al., 2020b; Li et al., 2021; Wei et al., 2022) and Quantization-Aware Training (QAT) (Bengio et al., 2013; Gong et al., 2019; Pei et al., 2023). While QAT quantizes weight to lower-bit data format, it requires fine-tuning the model parameters to adjust the value range. On the other hand, PTQ only focuses on the data format transformation and leaves the model parameters fixed. In general, PTQ may show less flexibility on the numerical adjustment. On the other hand, fine-tuning or retraining is not a practical choice in the LLM scenario; the data expense and hardware requirements are souring as the LLM models get bulky. PTQ is the feasible solution for LLM deployment, which is also why we explore the potential of PTQ in this paper.

Hardware-aware quantization shall be discussed inevitably. Given that the main objective of quantization is to optimize the inference latency on hardware platforms, it is important to consider provided hardware resources such as memory hierarchy, bandwidth limit, specific accelerator architecture (Yao et al., 2021; Zhao et al., 2023; Hawks et al., 2021; Wang et al., 2019). Some approaches are dedicated to targeting mixed-precision quantization (Zhou et al., 2018; Wu et al., 2018) while some others target the compilation stage (Yao et al., 2021) with system-level optimization on quantization kernels such as dataflow optimization and loop order/tiling optimization during quantization. Memory access patterns also need to be considered. Several memory access patterns can seriously affect model efficiency, such



(a) IEEE FP16 (b) Block Floating-Point (c) Bi-Exponent Block Floating-Point

Figure 2. An illustration of different numerical representation formats: (a) IEEE Half-Precision Floating-Point (FP16), (b) Block Floating-Point and (c) Bi-Exponent Block Floating-Point.

as per-channel quantization for activation. Thus, when designing quantization algorithms, similar operations should be avoided to be more relevant to practical applications.

2.2. Block Floating Point

The IEEE standard Floating-Point comprises three components: sign bit s, exponent e, and mantissa m. The real value is represented with $v = (-1)^s \times 1.m \times 2^{e-e_{off}}$ where m, e denotes binarized integer and e_{off} is the exponent offset. It is well noted that floating point data types inherently represent much wider value range than integer for its exponent bits. FP32 value ranges up to 2^{127}, and FP16 ranges to 2^{16}, and they can represent decimal numbers smaller than 1. On the other hand, integer quantization is much more popular, given its efficiency for computation and hardware friendliness. The processing unit architecture design for integer operation is much simpler. Many DNN accelerator hardware, such as Tensor Core on Nvidia GPU or open-sourced Gemmini (Genc et al., 2021), are specifically designed for integer matrix multiplication and addition operations.

For a vector $\mathbf{X}$ of N elements, we can represent the elements in FP16 representations with s, e, and m as the private signs, private exponents, and private mantissas, Figure 2(a) illustrates a 4-element vector in FP16 format:

$$[(-1)^{s_0} 2^{e_0} m_0, (-1)^{s_1} 2^{e_1} m_1, ..., (-1)^{s_{N-1}} 2^{e_{N-1}} m_{N-1}]. \tag{2}$$

In addition to floating point and fixed point, BFP is also a commonly used numerical representation. BFP arithmetic has been used in different kinds of signal processing applications like recursive digital filters (Oppenheim, 1970), calculation of Fast Fourier Transform (FFT) (Oppenheim & Weinstein, 1972) and Fast Hartley Transform (FHT) (ERICkSON & Fagin, 1992). BFP representation is a special case of floating-point representation where numbers within a block share a common exponent. Due to its exponent part, it provides BFP with a high dynamic range of floating-point representation. However, the computational complexity of two BFP blocks can be reduced to the degree of integer representation, which is much more efficient than the floating-point arithmetic.

BFP format combines the precision of floating-point numbers with the efficiency of fixed-point numbers. Therefore, some research has explored the application of BFP to model quantization with specified hardware implementation to achieve a balance between precision and efficiency (Fan et al., 2019; Lian et al., 2019). In addition to optimizing inference, efficient training is also a critical topic. FAST (Zhang et al., 2022b) and FlexBlock (Noh et al., 2023) tried to use BFP format for model training to optimize memory requirements while maintaining accuracy. AFP proposes an adaptive BFP with private exponent offset, which can dynamically adjust to the characteristics of deep learning data to improve the precision (Yeh et al., 2022). BFP is also a promising numerical representation for LLMs quantization (Zhang et al., 2023).

Given a block $\mathbf{X}$ of N elements in FP16, we can convert them into BFP format $\mathbf{X}'$, including a shared exponent e_m, private signs s and private block mantissas m' shown in Figure 2(b), which is written as

$$2^{e_m}[(-1)^{s_0} m'_0, (-1)^{s_1} m'_1, ..., (-1)^{s_{N-1}} m'_{N-1}], \tag{3}$$

where

$$e_m = \max_i e_i, i \in 0, 1, ..., N-1 \tag{4}$$

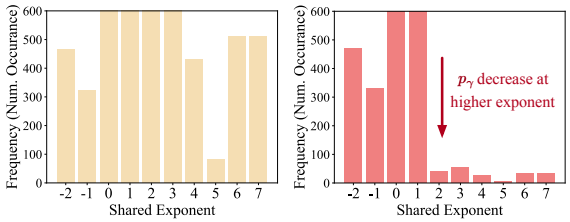
$$m'_i = m_i >> (e_m - e_i), i \in 0, 1, ..., N-1 \tag{5}$$

where $>>$ is the right shift operation.

The dot product of two vectors $\mathbf{X}_1$ and $\mathbf{X}_2$ with N elements in BFP format, the calculation will be donated as

$$2^{e_{m_1}+e_{m_2}} \sum_{i=0}^{N-1} ((-1)^{s_{1,i} \oplus s_{2,i}} m_{1,i} \cdot m_{2,i}), \tag{6}$$

where e_{m_1} and e_{m_2} are the shared exponent of $\mathbf{X}_1$ and $\mathbf{X}_2$, $\bigoplus$ is an XOR operation.



(a) BFP (b) BiE

Figure 3. p_γ distribution of BFP and BiE on the same activation in OPT-66B model. The y-axis represents the number of shared exponent occurance. p_γ decrease at higher exponents in BiE format, resulting in lower quantization error.

2.3. Data Distribution in LLMs

Outliers are the major challenge in LLMs quantization. LLMs exhibit a distinct pattern in the appearance of outliers, which are limited to a small number of channels in the activation tensor, and the variation within a channel is relatively minor (Dettmers et al., 2022; Xiao et al., 2023). Furthermore, we found that the distribution can be approximated as a superposition of two distributions as Figure 1 shows, which is the value distribution of a batched input activation tensor in OPT-66B model. One part has a distribution whose magnitude is close to 0, while the other has a distribution with a magnitude far from 0.

One way to deal with this situation is to separate the two parts for further process—for example, LLM.Int8() (Dettmers et al., 2022) introduces a mixed-precision method for LLM quantization, separating the two distribution parts based on a threshold value; the normal part is quantized into INT8 format. The outlier part, which is the more difficult part to quantize, is represented by FP16. Then, the other option is to fuse the two parts, with outliers eliminated or mitigated by the transformation between outlier activation and flat weight, channel-wise shifting, and scaling (Xiao et al., 2023; Wei et al., 2023). Inspired by the data distribution properties of LLMs and the limitation of the existing numerical format, we explore the idea of separation on the numerical representation, which is our proposed BiE. Unlike the mixed-precision approach, BiE is a unified format using the same arithmetic units, encoders, and decoders to maximize hardware efficiency.

3. Bi-Exponent Block Floating-Point Flow

3.1. BiE Data Format

3.1.1. Computing Error of Block Floating-Point

In order to optimize BFP, we must first understand the source of the BFP quantization error. The quantization error analysis was introduced by (Kalliojarvi & Astola, 1996; Song et al., 2018), for a block $\mathbf{X}$, when using rounding to nearest scheme, the quantization error has zero mean and variance σ^2 which is defined as

$$\sigma^2 = \frac{2^{-2L_m}}{12} \sum_{i=1}^{N_\gamma} p_{\gamma_i} 2^{2\gamma_i}, \tag{7}$$

where L_m is the bit length of the private block mantissa m_i', p_{γ_i} is the probability mass function (PMF) of the shared exponents γ_i $(i = 1, 2, ...N_\gamma)$. $N_\gamma = 2^{L_E}$ is the number of available shared exponent levels, where L_E is the bit length of shared exponent.

From Equation (7), it is found that the quantization errors mainly depend on L_m and p_{γ_i}. When L_m is increasing, the quantization error will be reduced. Moreover, if the probabilities of taking a larger exponent as the shared exponent are smaller, then the quantization errors of BFP will be reduced. Since the bit length is fixed in the typical quantization flow, we can only decrease the probabilities of larger shared exponents for quantization error reduction.

AFP (Yeh et al., 2022) introduces a 3-bit private exponent offset for each component within the block. For example, if the shared exponent is 3, the value 2 would have an offset of 2. In this way, the influence of the large shared exponent on the small-exponent value will be weakened. For low-bit quantization, however, this format is not well suited. First, AFP needs extra bits to store the offsets, reducing the memory efficiency. Second, in LLM quantization, the outliers in activations will require more bits for offsets when the mantissa bits get lower to maintain the precision. Third, the AutoFocus features in AFP need a fully dynamic encoder and decoder, which will cause significant hardware overhead. An alternative that balances precision and efficiency deserves discussion.

3.1.2. Design Details

To overcome the previously mentioned problems and challenges, we propose Bi-Exponent BFP (BiE), which is designed to be effective and efficient for Deep Neural Network (DNN) models, especially LLMs. Different from the vanilla BFP, the significant modification is that the BiE format has a bi-shared exponent for each block, e_o for the outlier part and e_n for the normal part, and private 1-bit type t_i that indicates this component belongs to outlier part or normal part. That means the normal part of the block will use e_n as their shared exponent, and the outlier part will use e_o as the corresponding shared exponent. The data format of BiE is illustrated in Figure 2(c).

Given a vector $\mathbf{X}$ with N elements in FP16, we can obtain its BiE representation $\mathbf{X}''$ as

$$2^{e_n|e_o}[(-1)^{s_0} m_0'', (-1)^{s_1} m_1'', ..., (-1)^{s_{N-1}} m_{N-1}''], \tag{8}$$

$$e_n = \max\{e_i \mid |x_i| \leq T\}, \tag{9}$$

$$e_o = \max\{e_i \mid |x_i| > T\}, \tag{10}$$

$$t_i = \begin{cases} 0 & |x_i| \leq T, \\ 1 & |x_i| > T, \end{cases} \tag{11}$$

$$m_i'' = m_i >> \begin{cases} e_n - e_i & t_i = 0, \\ e_o - e_i & t_i = 1, \end{cases} \tag{12}$$

where T is the threshold value in FP16 for distinguishing the normal part and the outlier part, which is always larger than 0. Hence, we use the magnitude of x_i to determine whether it is an outlier or not. m_i'' denotes the private mantissa of BiE. $e_n|e_o$ means that if $t_i = 0$, shared exponent of x_i'' is e_n, otherwise it is e_o. Although T is an FP16 number, the memory overhead is negligible because, in our quantization configuration, each tensor will have only one threshold T, i.e., a threshold per tensor.

The BiE arithmetic is similar to the vanilla BFP. The mantissa part is the same as BFP, which is fixed point multiplication. For the exponent part, the corresponding exponents are then added as the result exponent.

The rationale for the bi-exponent is also to address the effect of a large shared exponent on other small-exponent values in the block to reduce the quantization error. In this way, the p_{γ_i} for relative large γ_i will be decreased, and the p_{γ_i} for relative small γ_i will be increased, but the overall quantization error will be reduced. For example, as Figure 3(a) shows, the shared exponents are concentrated in the middle part of the distribution (0-3) and have quite a few large exponent values as shared exponents (4-7) originally. By introducing an additional shared exponent for outliers within the block, the frequency of higher exponents (4-7) shrinks tremendously, and even the frequency of some middle parts is significantly reduced (2-3) while migrating to some exponents in the even smaller values (0-1) illustrated in Figure 3(b). In this case, the quantization error of BiE decreases, which can be derived from Equation (7). BiE uses such a separate way to adapt to the data distribution of LLMs.

For a uniform representation, the weights will also be divided into two parts. Although the weights will not have outliers of particularly large magnitude, there will be relative outliers or relatively large values in a block, so it is possible to split the values in the block into relatively small (normal) and relatively large (outlier) parts, thus conforming to the BiE format. Hence, the BiE format can also be extended and implemented for any other DNN models with precision improvements compared with BFP.

3.1.3. Numerical Efficiency

Quantization Flow: For fixed-point quantization, FP16 number will be mapped to integer by using Equation (1).

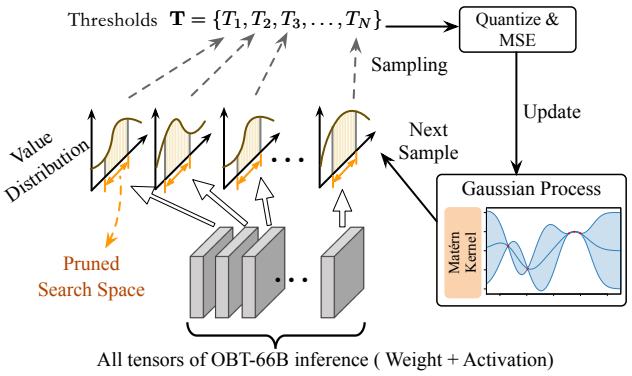


Figure 4. Threshold searching with Bayesian optimization with pruned search space.

Different from fixed-point quantization, BiE quantization requires a comparative operation to find the bi-shared exponent and shifting and rounding operations to obtain the quantized private block mantissa. These are all efficient operations for hardware compared with scaling operation, which is floating-point multiplication in fixed-point quantization. In post-training quantization, fixed-point quantization is usually about finding an appropriate scaling factor, whereas in BiE quantization, we are looking for thresholds, which are both at a certain granularity, e.g., per-tensor or per-channel.

Data Efficiency: BiE can also be seen as a special case of fixed-point. When the bit length of mantissa equals the bits' number of fixed points, the computational efficiency of both is roughly the same. However, BiE has a bi-exponent, which improves the dynamic range but sacrifices some memory efficiency. As a result, BiE can achieve lower-bit quantization. Thus, in a lower-bit configuration, BiE can achieve precision comparable to INT8 with higher computation and memory efficiency, which is the superiority of BiE.

3.2. Offline Threshold Searching Strategy

As discussed in Section 3.1.2, threshold determination is rather critical before the BiE quantization flow. In order to select the optimal threshold value for each tensor during inference, we build an efficient offline threshold searching strategy based on Bayesian Optimization (BO), which is visualized in Figure 2(c).

Firstly, we define the threshold value **search space** for all tensors of LLM inference stage (including weights and activations) here:

$$\mathbf{\Omega} = \{T_1, T_2, ..., T_N\},\ N = N_a + N_w, \tag{13}$$

where N_a and N_w are the number of activations and weights in the models, where each variable T_i denotes the threshold

candidate (in real number) for i-th tensor. Now, this problem is a multi-variable design space exploration (DSE) problem.

Before we dig into our specific design, We need to discuss the limitations and how we narrow down our offline solution to this problem. For latency consideration, we give up online thresholding: adaptively splitting the block into two groups during FP-to-BiE conversion while minimizing quantization error. This is impractical because dynamic encoding is not only very time-consuming but also requires additional hardware design overhead. We also give up using the gradient-descent method because the partial derivatives with respect to the thresholds do not exist, which is non-differentiable and hard to approximate.

Now that we have the search space, we need a surrogate model to guide the sampling by learning the quantization performance distribution among the search space. As shown in Figure 4, we adopt the Gaussian Process (GP) as the surrogate model with matérn kernel function (details in Appendix C) for its robust ability to capture uncertainty quantification during modeling. For the quantization performance indicator, we simply use the mean square error (MSE) between the output of the full-precision model and the quantized model:

$$\min_{\mathbf{T}} \frac{1}{N_s} \sum \|f(\mathbf{W}, \mathbf{X}) - f(Q_{\mathbf{T}}(\mathbf{W}), Q_{\mathbf{T}}(\mathbf{X}))\|_2^2, \quad (14)$$

where f is the LLM model inference, $Q_{\mathbf{T}}(\cdot)$ denotes BiE quantization operation with thresholds $\mathbf{T}$ for each weight and activation tensor, N_s represents the number of calibration samples. $\mathbf{W}$ is the weights of the model and $\mathbf{X}$ is the activations of the model. Using the MSE of the final output as an indicator can take the dependencies between layers into account, thus allowing the quantized model to better fit the output of the original model.

We find the proposed search space in Equation (14) is still too large for the enormous number of tensors in LLM. We need to prune the search space, trimming off unnecessary subspace before the sampling to improve searching efficiency further. Given the design of our BiE format, we need to divide a block of data into normal and outlier parts. We empirically find that the threshold is around the median of value distribution so that the superiority of BiE can be fully exploited. We choose to use the percentile method for calibration, which will set the range to a percentile of the distribution of absolute values seen during calibration. Hence, the search space is bounded as:

$$\begin{aligned} &\mathbf{\Omega} = \{T_1, T_2, ..., T_N\}, \ N = N_a + N_w; \\ &T_i \in [P_{lo}(\mathbf{X}), P_{hi}(\mathbf{X})]. \end{aligned} \quad (15)$$

P_{lo} and P_{hi} are the lower-bound and higher-bound of the percentile. $\mathbf{X}$ is the activations or weights in calibrations. In our implementation, P_{lo} and P_{hi} are set as 75% and 95%.

Table 1. Hardware efficiency of the same size PE array with different numerical representations including the ratio of throughput and memory efficiency compared with FP16. We **highlight** the hardware efficiency of BiE.

Method	Config	Throughput	Mem. Effi.
FP16	-	1.00×	1.0×
INT	W8A8	1.46×	2.0×
INT	W6A6	1.62×	2.7×
BFP	W4A4	3.51×	3.7×
BFP	W3A3	3.96×	4.8×
BiE	W4A4	**3.51**×	**2.8**×
BiE	W3A3	**3.96**×	**3.5**×

Once the surrogate model is established, we search within the pruned parameter space by iteratively sampling $\mathbf{T} = \{T_1, T_2, T_3...\}$ candidates with given performance modeling GP. The intuition is to select the next point $\mathbf{T}^*$ with highest expected improvement (EI):

$$\mathbf{T}^* = \text{argmax}_{\mathbf{T}} EI_{\mathbf{\Omega}}(\mathcal{P}, GP(\mathbf{T})), \quad (16)$$

where $\mathcal{P}$ is the optimal Pareto MSE reached by the sampled threshold points. $GP(\mathbf{T})$ is the performance distribution modeled from the currently sampled $\mathbf{T}$s. The newly sampled points will also be used to update the GP by updating the Matérn kernel function. Here the $EI(\cdot)$ is the optimal improvement from $\mathcal{P}$. The details of the derivation of $EI(\cdot)$ is in Appendix C.

3.3. Hardware-Level Analysis

In this section, we validate the hardware-level efficiency of our proposed BiE quantization. We implement the customized DNN accelerator for this data format and evaluate the hardware resources and throughput advantage. Not only is the performance competitive, but the hardware design itself is simple, such that it can easily adapt to general-purpose hardware accelerator (Jouppi et al., 2017; Genc et al., 2021) with small effort.

The customized accelerator has a systolic array architecture that supports the BiE arithmetic. The BiE systolic array consists of three parts: Floating-Point to BiE converter (Encoder), processing elements (PE) for BiE, and decoder.

Given two blocks $\mathbf{X}_1$ and $\mathbf{X}_2$ with N elements in BiE format, the dot product of them can be formulated as

$$e_m = e_{o_1} + e_{o_2} \quad (17)$$

$$2^{e_m} \sum_{0}^{N-1} (-1)^{s_{1,i} \oplus s_{2,i}} (m_{1,i} \cdot m_{2,i} >> (e_m - e_{1,i} - e_{2,i})) \quad (18)$$

where e_m is the max exponent combination of the two blocks which is determined by the summation of the two

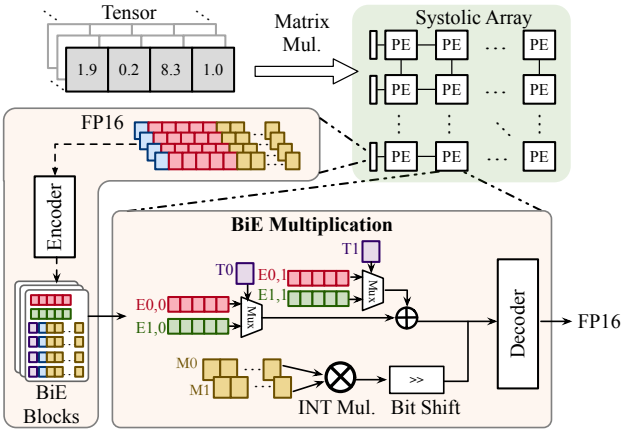


Figure 5. Our proposed BiE-based accelerator. Inside each Pe Unit, the BiE multiplication is decomposed into integer multiplication and bit shift operations. Two multiplexers are integrated to handle outlier exponents.

outlier shared exponents e_{o_1} and e_{o_2}, $e_{1,i}$ and $e_{2,i}$ represent the shared exponent used for the ith element of $\mathbf{X}_1$ and $\mathbf{X}_2$.

According to Equation (18), one of the most significant advantages of BiE is its computational efficiency. In BiE arithmetic, it only involves **INT multiplication**, **INT addition** and **shifting**, which are both efficient for hardware. For the arithmetic of BiE, as shown in Figure 5, we illustrate how to reorganize the original PE to support BiE multiplication. The shared exponents of two blocks are added in parallel with the INT multiplication. Thus, it avoids introducing extra delay. A multiplexer detects the Type bit to select the corresponding shared exponent. As shown in Equation (18), a bit shifter is required for the mantissa product alignment. The right shift amount is $e_m - e_{1,i} - e_{2,i}$.

Compared with fixed point quantization, the encoder and decoder do not involve costly floating-point multiplication. The encoder for BiE is required to handle the operations related to the threshold, such as obtaining the Type bit t_i of each element through two comparator trees and determining the maximum exponents of the numbers above and below the threshold in the input block as the shared exponents e_o and e_n. If no outliers exist in the block, the default value for e_o equals e_n. It can reduce the necessity for the comparator to determine e_m. The dot product of BiE will obtain a partial sum of aligned mantissa products, and the max shared exponents summation e_m; thus, the decoder is the same as BFP. The decoder has a leading zero detector to obtain the normalization factor and a shifter to achieve normalization. Then, the decoder will convert the dot product result into FP16 format for further inter-block summation, the same as BFP.

Table 2. Comparison with different methods and different quantization configurations for OPT-models on Wikitext2 (**Perplexity**↓). We **highlight** our 4-bit BiE results which are comparable with SmoothQuant W8A8.

Method	Config	6.7B	13B	30B	66B
FP16	/	10.64	9.91	9.33	9.12
SmoothQuant	W8A8	11.33	12.79	9.35	9.62
SmoothQuant	W6A6	13.16	13.75	82.54	3383.21
BFP	W4A4	11.22	11.15	9.90	14.16
BiE (Ours)	W4A4	**10.93**	**10.39**	**9.37**	**9.82**
BFP	W3A3	14.61	13.85	13.83	137.72
BiE (Ours)	W3A3	12.10	11.13	10.01	32.41

Unlike the BFP hardware design, the hardware for BiE requires an extra comparator tree for the additional shared exponent extraction. The difference between the PE design is that the products of mantissa need to be aligned by a shift operation in order to compute the partial sum of the products correctly. In addition, the shift amount depends on the difference between the sum of the outlier shared exponents and the sum of the shared exponents of the current elements. The hardware implementations of the decoder for BFP and BiE are identical, and no additional hardware design is required for BiE. Concerning the inter-block FP addition, the hardware overhead is identical for the BiE and BFP. Thus, the hardware implementation of BiE will not introduce much hardware overhead.

The BiE PE array is implemented by Xilinx Vitis High-Level Synthesis Tool (HLS) 2020.1 targeting Field Programmable Gate Arrays (FPGA), which is Xilinx Zynq UltraScale+ ZCU104 Evaluation Board.

For a fair comparison, we also implemented a PE array with the same size for Floating-Point (FP16), Fixed-Point (INT8 and INT6), and Block Floating-Point (BFP4 and BFP3), respectively.

From Table 1, the simulation results indicate that the BiE can achieve the same performance with tiny memory overhead in the same settings compared with BFP, as expected. BiE4 increases the throughput by $3.51\times$, $2.40\times$ and $2.17\times$, and memory efficiency by $2.80\times$, $1.40\times$ and $1.04\times$ compared with FP16, INT8, INT6, respectively.

4. Experiments

4.1. Settings

Baselines. We compared with two baselines in different post-training quantization configurations. For fixed-point quantiation, we choose Smoothquant (Xiao et al., 2023) with INT8 and INT6 settings. For naive BFP quantization for LLM, we compare with (Zhang et al., 2023) with 4-bit and 3-bit configurations.

Table 3. BiE's performance for OPT-30B and OPT-66B on other zero-shot tasks including multiple choice, commonsense reasoning, etc.. We **highlight** our 4-bit BiE results that can achieve nearly lossless quantization performance.

Model	Method	Config	LAMBADA	Arc_easy	PIQA	COPA	QNLI	SST2	**Average↑**
OPT-30B	FP16	/	71.45%	70.03%	77.64%	82.00%	51.78%	66.51%	69.90%
	SmoothQuant	W8A8	71.71%	69.61%	77.75%	84.00%	52.52%	66.63%	70.37%
	SmoothQuant	W6A6	0.54%	41.29%	57.94%	66.00%	51.44%	58.26%	45.91%
	BFP	W4A4	61.25%	68.18%	76.28%	85.00%	54.09%	66.51%	68.55%
	BiE (Ours)	W4A4	**70.11**%	**69.15**%	**77.26**%	**85.00**%	**52.15**%	**67.66**%	**70.22**%
	BFP	W3A3	7.45%	56.40%	66.43%	77.00%	50.92%	56.42%	52.44%
	BiE (Ours)	W3A3	65.11%	68.14%	75.84%	81.00%	52.19%	61.12%	67.23%
OPT-66B	FP16	/	73.96%	71.12%	78.73%	86.00%	52.19%	68.58%	71.76%
	SmoothQuant	W8A8	73.26%	71.21%	78.35%	86.00%	51.84%	63.19%	70.64%
	SmoothQuant	W6A6	0.00%	25.29%	53.32%	55.00%	50.67%	51.26%	39.26%
	BFP	W4A4	63.83%	68.86%	76.66%	86.00%	52.10%	64.68%	68.69%
	BiE (Ours)	W4A4	**72.50**%	**70.58**%	**77.26**%	**85.00**%	**51.95**%	**70.07**%	**71.23**%
	BFP	W3A3	3.22%	36.49%	55.44%	61.00%	51.09%	52.98%	43.37%
	BiE (Ours)	W3A3	11.97%	38.34%	56.04%	60.00%	49.68%	53.33%	44.89%

Models and Datasets We evaluate BiE with two representative families of LLMs: OPT (Zhang et al., 2022a) ranging from 6.7B to 66B and LLaMA-2 (Touvron et al., 2023) including 7B, 13B and 70B. For threshold searching, we use Pile datasets (Gao et al., 2020) with random 128 samples as the calibration dataset to get the statistical characteristics of activations and weights. Seven zero-shot NLP tasks including multiple choice, commonsense reasoning, language modeling, etc.: LAMBADA (Paperno et al., 2016), Arc_easy (Clark et al., 2018), PIQA (Bisk et al., 2020), COPA (Roemmele et al., 2011), QNLI (Wang et al., 2018) SST2 (Socher et al., 2013), WikiText2 (Merity et al., 2016) are utilized for evaluated on OPT and LLaMA-2 models. The evaluation code is based on `lm-evaluation-harness` (Gao et al., 2023).

Implementations. To simulate the behavior of BiE in hardware, we use the Pytorch (Paszke et al., 2019) and huggingface (Wolf et al., 2019) libraries to quantize the models on four A100-80G GPUs. The quantization configurations are W4A4 and W3A3 for BiE and BFP, with 4-bit mantissa and 3-bit mantissa, respectively; both of them have 5-bit for one shared exponent. We use 16 elements as a block for both activations and weights, which is a slice along the matrix row. Each tensor will use a threshold to distinguish between normal and outlier parts. We quantized all the matrix multiplications in the Transformer Decoder Layer, including all the Linear Layer, `bmm` in OPT models and `matmul` in LLaMA-2 models, and left other parts in FP16, e.g., Softmax and LayerNorm.

4.2. Results of PTQ for LLMs

Results on OPT models. We apply BiE PTQ for different scales OPT ranging from 6.7B to 66B compared with SmoothQuant (Xiao et al., 2023) and vanilla BFP (Zhang et al., 2023) on various tasks. From Table 2, BiE with 4-bit achieves comparable performances with FP16 and SmoothQuant with 8-bit and outperforms the vanilla BFP on language modeling task WikiText2. With 6-bit SmoothQuant and 3-bit BFP, performances degrade considerably, especially on the OPT-30B and 66B models. This is because, compared to the other two models, the OPT-30B and 66B models have more severe and larger outliers. However, even in the 3-bit case, BiE is able to maintain a stable performance without significant precision degradation. For other tasks, including multiple-choice, commonsense reasoning, etc., the performances of BiE are shown in Table 3. Experimental results demonstrate the comprehensive superiority of BiE. To show the performance of BiE in models that are non-outliers or whose maximum magnitude is relatively low, we applied BiE to small-scale OPT models from 125m to 2.7B. From Table 4, the performance of BiE is similar to its results for OPT (>6B). BiE4 enables almost lossless quantization, while BiE3 better preserves precision compared to BFP3. This demonstrates BiE's ability to push the boundaries of low-bit block-based quantization and maintain its good properties even in the non-outliers distribution. More detailed experimental results about BiE

Table 4. Block-based quantization for OPT-models (<6.7B) on Wikitext2 (**Perplexity**↓). We **highlight** our BiE results which outperform BFP under the same quantization configurations.

Method	Config	125m	350m	1.3B	2.7B
FP16	/	27.50	21.76	14.38	12.18
BFP	W4A4	30.17	23.82	16.38	13.06
BiE	W4A4	**28.20**	**22.34**	**15.28**	**12.54**
BFP	W3A3	52.76	47.58	55.67	27.83
BiE	W3A3	**34.54**	**27.10**	**19.85**	**15.39**

Table 5. PTQ performances using different methods on LLaMA-2 models for various tasks. **Average**↑ is the average accuracy among various tasks. **Perplexity**↓ is for WikiText2. SQ represents SmoothQuant. We **highlight** our 4-bit BiE results.

	Config	Average↑			Perplexity↓		
		7B	13B	70B	7B	13B	70B
FP16	/	76.16	77.50	81.62	6.73	5.95	4.51
SQ	W8A8	75.12	77.47	80.35	6.93	5.94	4.56
SQ	W6A6	66.07	67.51	76.96	10.38	8.28	5.73
BFP	W4A4	68.65	71.20	80.24	7.69	6.74	4.69
BiE	W4A4	**72.83**	**75.90**	**80.21**	**7.00**	**6.16**	**4.61**
BFP	W3A3	42.28	43.52	73.05	20.86	14.70	5.78
BiE	W3A3	50.00	63.25	77.24	8.92	7.86	5.21

on OPT models are shown in Appendix A.1.

Results on LLaMA-2 models. BiE also works well for LLaMA-2 family shown in Table 5. 4-bit BiE will be the first choice that can balance precision and efficiency. For Wikitext2 tasks in particular, the stability of BiE is very impressive, and even at 3bit, the performance gap is not particularly large. It should be noted that 6bit SmoothQuant also gives quite good results on the LLaMA-2 model. However, block-based methods, BFP and BiE, do not require any transformations between activations and weights or complex operations on the model architectures but only focus on the data itself. The experimental results indicate that the performance degradation of the low-bit quantization method is lesser than that of the OPT models, which can be due to two reasons: firstly, the model architectures of the two are different, and secondly, the magnitude of the outliers for the activations is not as high in the LLaMA-2 models as in the OPT models. More detailed results can be found in Appendix A.2.

5. Conclusion

In this paper, we discuss the the limitations of the existing quantization data format for LLMs and propose a novel numerical representation, BiE, Bi-Exponent Block Floating-point combined with a new quantization flow. BiE can balance precision and hardware efficiency. BiE can be naturally adapted to the numerical distribution characteristics of the LLMs and achieve negligible loss in 4-bit activations and weights quantization. We also implement a hardware accelerator to validate the hardware-level computational efficiency of BiE quantization.

Impact Statement

This paper contributes to the advancement of Machine Learning. While there are numerous potential societal impacts stemming from our research, we do not believe that any specific issues need to be highlighted in this context.

References

Bengio, Y., Léonard, N., and Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. PIQA: Reasoning about Physical Commonsense in Natural Language. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 34, pp. 7432–7439, 2020.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 33:1877–1901, 2020.

Choi, J., Wang, Z., Venkataramani, S., Chuang, P. I.-J., Srinivasan, V., and Gopalakrishnan, K. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

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

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

ERICkSON, A. C. and Fagin, B. S. Calculating the FHT in hardware. *IEEE Transactions on Signal Processing (TSP)*, 40(6):1341–1353, 1992.

Fan, H., Wang, G., Ferianc, M., Niu, X., and Luk, W. Static block floating-point quantization for convolutional neural networks on FPGA. In *IEEE International Conference on Field-Programmable Technology (FPT)*, pp. 28–35. IEEE, 2019.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., et al. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou,

A. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836.

Genc, H., Kim, S., Amid, A., Haj-Ali, A., Iyer, V., Prakash, P., Zhao, J., Grubb, D., Liew, H., Mao, H., et al. Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration. In *ACM/IEEE Design Automation Conference (DAC)*, pp. 769–774. IEEE, 2021.

Gong, R., Liu, X., Jiang, S., Li, T., Hu, P., Lin, J., Yu, F., and Yan, J. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4852–4861, 2019.

Han, S., Mao, H., and Dally, W. J. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv preprint arXiv:1510.00149*, 2015.

Hawks, B., Duarte, J., Fraser, N. J., Pappalardo, A., Tran, N., and Umuroglu, Y. Ps and qs: Quantization-aware pruning for efficient low latency neural network inference. *Frontiers in Artificial Intelligence*, 4:676564, 2021.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2704–2713, 2018.

Johnson, J. Rethinking floating point for deep learning. *arXiv preprint arXiv:1811.01721*, 2018.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, pp. 1–12, 2017.

Kalliojarvi, K. and Astola, J. Roundoff errors in block-floating-point systems. *IEEE Transactions on Signal Processing (TSP)*, 44(4):783–790, 1996.

Kim, S., Gholami, A., Yao, Z., Mahoney, M. W., and Keutzer, K. I-bert: Integer-only bert quantization. In *International conference on machine learning*, pp. 5506–5518. PMLR, 2021.

Li, J., Li, D., Savarese, S., and Hoi, S. BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models. *arXiv preprint arXiv:2301.12597*, 2023.

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

Lian, X., Liu, Z., Song, Z., Dai, J., Zhou, W., and Ji, X. High-performance FPGA-based CNN accelerator with block-floating-point arithmetic. *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, 27(8):1874–1885, 2019.

Lin, D., Talathi, S., and Annapureddy, S. Fixed point quantization of deep convolutional networks. In *International conference on machine learning*, pp. 2849–2858. PMLR, 2016.

Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., and Han, S. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

Liu, F., Zhao, W., He, Z., Wang, Y., Wang, Z., Dai, C., Liang, X., and Jiang, L. Improving neural network efficiency via post-training quantization with adaptive floating-point. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5281–5290, 2021.

Liu, H., Li, C., Wu, Q., and Lee, Y. J. Visual instruction tuning. *arXiv preprint arXiv:2304.08485*, 2023.

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

Nagel, M., Amjad, R. A., Van Baalen, M., Louizos, C., and Blankevoort, T. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning*, pp. 7197–7206. PMLR, 2020a.

Nagel, M., Amjad, R. A., Van Baalen, M., Louizos, C., and Blankevoort, T. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning*, pp. 7197–7206. PMLR, 2020b.

Noh, S.-H., Koo, J., Lee, S., Park, J., and Kung, J. FlexBlock: A flexible DNN training accelerator with multi-mode block floating point support. *IEEE Transactions on Computers*, 2023.

Oppenheim, A. Realization of digital filters using block-floating-point arithmetic. *IEEE Transactions on Audio and Electroacoustics*, 18(2):130–136, 1970.

Oppenheim, A. V. and Weinstein, C. J. Effects of finite register length in digital filtering and the fast fourier transform. *Proceedings of the IEEE*, 60(8):957–976, 1972.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, Q. N., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The LAMBADA dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 32, 2019.

Pei, Z., Yao, X., Zhao, W., and Yu, B. Quantization via distillation and contrastive learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

Roemmele, M., Bejan, C. A., and Gordon, A. S. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*, 2011.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1631–1642, 2013.

Song, Z., Liu, Z., and Wang, D. Computation error analysis of block floating point arithmetic oriented convolution neural network accelerator design. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 32, 2018.

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

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Wang, K., Liu, Z., Lin, Y., Lin, J., and Han, S. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8612–8620, 2019.

Wei, X., Gong, R., Li, Y., Liu, X., and Yu, F. Qdrop: Randomly dropping quantization for extremely low-bit post-training quantization. *arXiv preprint arXiv:2203.05740*, 2022.

Wei, X., Zhang, Y., Li, Y., Zhang, X., Gong, R., Guo, J., and Liu, X. Outlier suppression+: Accurate quantization of large language models by equivalent and optimal shifting and scaling. *arXiv preprint arXiv:2304.09145*, 2023.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *arXiv preprint arXiv:1910.03771*, 2019.

Wu, B., Wang, Y., Zhang, P., Tian, Y., Vajda, P., and Keutzer, K. Mixed precision quantization of convnets via differentiable neural architecture search. *arXiv preprint arXiv:1812.00090*, 2018.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning (ICML)*, pp. 38087–38099. PMLR, 2023.

Yao, Z., Dong, Z., Zheng, Z., Gholami, A., Yu, J., Tan, E., Wang, L., Huang, Q., Wang, Y., Mahoney, M., et al. Hawq-v3: Dyadic neural network quantization. In *International Conference on Machine Learning*, pp. 11875–11886. PMLR, 2021.

Yao, Z., Yazdani Aminabadi, R., Zhang, M., Wu, X., Li, C., and He, Y. ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 35:27168–27183, 2022.

Yeh, T., Sterner, M., Lai, Z., Chuang, B., and Ihler, A. Be Like Water: Adaptive Floating Point for Machine Learning. In *International Conference on Machine Learning (ICML)*, pp. 25490–25500. PMLR, 2022.

Zhang, C., Cheng, J., Shumailov, I., Constantinides, G., and Zhao, Y. Revisiting Block-based Quantisation: What is Important for Sub-8-bit LLM Inference? In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 9988–10006, 2023.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. OPT: Open Pre-trained Transformer Language Models. *arXiv preprint arXiv:2205.01068*, 2022a.

Zhang, S. Q., McDanel, B., and Kung, H. FAST: DNN Training Under Variable Precision Block Floating Point with Stochastic Rounding. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 846–860. IEEE, 2022b.

Zhao, W., Bai, Y., Sun, Q., Li, W., Zheng, H., Jiang, N., Lu, J., Yu, B., and Wong, M. D. A high-performance accelerator for super-resolution processing on embedded gpu. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

Zhou, Y., Moosavi-Dezfooli, S.-M., Cheung, N.-M., and Frossard, P. Adaptive quantization for deep neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

A. Detailed Experiments

A.1. Results on OPT

The Table 6 demonstrates the complete experimental results on OPT (>6B) models. The results show the robustness of BiE on various scales of OPT models with outliers. To illustrate the generalizability of BiE, we also conducted the same experiments on the small OPT model. From Table 7, we compare it to a vanilla BFP with the same quantization configuration, and we can see that BiE still outperforms the BFP when the outlier is less pronounced, or there is no distribution of outliers. According to the design of BiE, no matter what kind of data distribution is, if the data distribution can be divided into two parts by the threshold (relatively normal and relatively outlier), then its performance will be better than that of BFP.

Table 6. Detailed comparison with different methods on OPT family

Model	Method	Config	LAMBADA	Arc_easy	PIQA	COPA	QNLI	SST2	**Average↑**	**WikiText↓**
OPT-6.7B	FP16	/	67.65%	65.61%	76.28%	81.00%	50.87%	76.49%	69.65%	10.64
	SmoothQuant	W8A8	68.62%	64.81%	76.50%	80.00%	50.67%	68.23%	68.14%	11.33
	SmoothQuant	W6A6	64.16%	59.55%	71.93%	82.00%	51.38%	57.22%	64.37%	13.16
	BFP	W4A4	57.11%	63.85%	75.19%	81.00%	54.07%	69.61%	66.81%	11.22
	BiE	W4A4	66.18%	64.65%	76.17%	83.00%	50.70%	73.39%	69.02%	10.93
	BFP	W3A3	30.43%	55.09%	68.72%	79.00%	50.47%	61.24%	57.49%	14.61
	BiE	W3A3	56.41%	60.56%	71.55%	79.00%	53.41%	62.27%	63.87%	12.10
OPT-13B	FP16	/	68.64%	67.09%	75.90%	86.00%	57.06%	59.86%	69.09%	9.91
	SmoothQuant	W8A8	68.41%	66.54%	75.30%	84.00%	56.14%	75.69%	71.01%	12.79
	SmoothQuant	W6A6	46.19%	54.00%	68.28%	63.00%	49.81%	60.09%	56.90%	13.75
	BFP	W4A4	59.11%	65.99%	74.54%	83.00%	54.18%	69.04%	67.64%	11.15
	BiE	W4A4	67.53%	66.54%	75.52%	83.00%	56.75%	55.96%	67.55%	10.39
	BFP	W3A3	6.75%	55.30%	67.03%	79.00%	51.88%	65.48%	54.24%	13.85
	BiE	W3A3	59.48%	65.19%	73.94%	81.00%	53.94%	67.89%	66.91%	11.13
OPT-30B	FP16	/	71.45%	70.03%	77.64%	82.00%	51.78%	66.51%	69.90%	9.33
	SmoothQuant	W8A8	71.71%	69.61%	77.75%	84.00%	52.52%	66.63%	70.37%	9.35
	SmoothQuant	W6A6	0.54%	41.29%	57.94%	66.00%	51.44%	58.26%	45.91%	82.54
	BFP	W4A4	61.25%	68.18%	76.28%	85.00%	54.09%	66.51%	68.55%	9.90
	BiE	W4A4	70.11%	69.15%	77.26%	85.00%	52.15%	67.66%	70.22%	9.37
	BFP	W3A3	7.45%	56.40%	66.43%	77.00%	50.92%	56.42%	52.44%	13.83
	BiE	W3A3	65.11%	68.14%	75.84%	81.00%	52.19%	61.12%	67.23%	10.01
OPT-66B	FP16	/	73.96%	71.12%	78.73%	86.00%	52.19%	68.58%	71.76%	9.12
	SmoothQuant	W8A8	73.26%	71.21%	78.35%	86.00%	51.84%	63.19%	70.64%	9.62
	SmoothQuant	W6A6	0.00%	25.29%	53.32%	55.00%	50.67%	51.26%	39.26%	3383.21
	BFP	W4A4	63.83%	68.86%	76.66%	86.00%	52.10%	64.68%	68.69%	14.16
	BiE	W4A4	72.50%	70.58%	77.26%	85.00%	51.95%	70.07%	71.23%	9.82
	BFP	W3A3	3.22%	36.49%	55.44%	61.00%	51.09%	52.98%	43.37%	137.72
	BiE	W3A3	11.97%	38.34%	56.04%	60.00%	49.68%	53.33%	44.89%	32.41

Table 7. Detailed comparison with blocked-based methods on OPT family (<6.7B)

Model	Method	Config	LAMBADA	Arc_easy	PIQA	COPA	QNLI	SST2	**Average↑**	**WikiText↓**
OPT-125m	FP16	/	37.84%	43.43%	63.06%	66.00%	49.44%	53.21%	52.16%	27.50
	BFP	W4A4	33.59%	41.08%	60.07%	71.00%	49.44%	50.80%	51.00%	30.17
	BiE	W4A4	37.98%	42.05%	61.53%	68.00%	49.44%	50.23%	51.54%	28.20
	BFP	W3A3	12.71%	34.81%	54.46%	64.00%	49.39%	50.46%	44.31%	52.76
	BiE	W3A3	30.25%	39.10%	58.27%	67.00%	49.35%	50.34%	49.05%	34.54
OPT-350m	FP16	/	45.16%	44.11%	64.47%	72.00%	49.53%	61.81%	56.18%	21.76
	BFP	W4A4	41.24%	42.13%	62.02%	69.00%	49.53%	60.55%	54.08%	23.82
	BiE	W4A4	45.20%	43.77%	64.04%	69.00%	49.51%	57.11%	54.77%	22.34
	BFP	W3A3	17.70%	35.27%	56.15%	67.00%	49.70%	49.66%	45.91%	47.58
	BiE	W3A3	36.74%	39.77%	62.08%	63.00%	49.22%	56.54%	51.23%	27.10
OPT-1.3B	FP16	/	57.93%	57.15%	71.65%	79.00%	51.57%	81.88%	66.53%	14.38
	BFP	W4A4	43.86%	53.11%	68.66%	75.00%	51.64%	65.25%	59.59%	16.38
	BiE	W4A4	50.88%	56.69%	70.35%	77.00%	51.75%	75.11%	63.63%	15.28
	BFP	W3A3	3.90%	40.78%	58.71%	66.00%	49.99%	50.57%	44.99%	55.67
	BiE	W3A3	33.48%	50.42%	66.10%	74.00%	49.64%	62.39%	56.00%	19.85
OPT-2.7B	FP16	/	63.63%	60.73%	73.83%	77.00%	51.20%	51.72%	63.02%	12.18
	BFP	W4A4	54.05%	57.95%	71.71%	78.00%	50.96%	51.72%	60.73%	13.06
	BiE	W4A4	61.61%	59.43%	72.58%	74.00%	50.17%	51.95%	61.62%	12.54
	BFP	W3A3	6.64%	44.61%	57.56%	69.00%	49.97%	53.78%	46.93%	27.83
	BiE	W3A3	39.26%	54.63%	68.82%	74.00%	49.70%	56.08%	57.08%	15.39

A.2. Results on LLaMA-2

Different from OPT models, the outliers are not severe in LLaMA-2 models. From Table 8, BiE W4A4 enables lossless quantization and achieves comparable performance with SmoothQuant (Xiao et al., 2023) W8A8.

Table 8. Detailed comparison with different methods on LLaMA-2 family

Size	Method	Config	LAMBADA	Arc_easy	PIQA	COPA	QNLI	SST2	**Average↑**	**WikiText↓**
7B	FP16	/	70.95%	73.86%	76.39%	89.00%	58.01%	88.76%	76.16%	6.73
	SmoothQuant	W8A8	70.83%	73.61%	75.90%	90.00%	55.15%	85.21%	75.12%	6.93
	SmoothQuant	W6A6	56.61%	66.58%	72.52%	80.00%	52.11%	68.58%	66.07%	10.38
	BFP	W4A4	51.93%	68.39%	71.06%	83.00%	54.05%	83.49%	68.65%	7.69
	BiE	W4A4	64.51%	71.30%	73.29%	88.00%	58.91%	80.96%	72.83%	7.00
	BFP	W3A3	2.43%	33.80%	55.22%	64.00%	49.94%	48.28%	42.28%	20.86
	BiE	W3A3	23.09%	52.82%	65.23%	72.00%	51.05%	65.83%	55.00%	8.92
13B	FP16	/	73.03%	77.57%	77.80%	90.00%	54.38%	92.20%	77.50%	5.95
	SmoothQuant	W8A8	72.89%	77.95%	77.42%	90.00%	54.68%	91.86%	77.47%	5.94
	SmoothQuant	W6A6	60.86%	70.79%	74.48%	83.00%	51.13%	64.79%	67.51%	8.28
	BFP	W4A4	54.73%	73.27%	74.65%	89.00%	51.24%	84.29%	71.20%	6.74
	BiE	W4A4	70.10%	76.81%	75.63%	88.00%	52.88%	91.95%	75.90%	6.16
	BFP	W3A3	3.43%	35.35%	56.26%	66.00%	50.67%	49.43%	43.52%	14.70
	BiE	W3A3	42.85%	66.12%	68.55%	82.00%	51.88%	68.12%	63.25%	7.86
70B	FP16	/	75.43%	80.30%	80.63%	89.00%	72.49%	91.86%	81.62%	4.51
	SmoothQuant	W8A8	75.22%	80.60%	80.36%	88.00%	66.19%	91.74%	80.35%	4.56
	SmoothQuant	W6A6	69.16%	77.23%	78.84%	86.00%	63.96%	86.58%	76.96%	5.73
	BFP	W4A4	73.14%	79.38%	79.54%	89.00%	69.54%	90.83%	80.24%	4.69
	BiE	W4A4	74.50%	80.64%	79.43%	90.00%	65.84%	90.83%	80.21%	4.61
	BFP	W3A3	59.11%	74.87%	74.54%	88.00%	54.05%	87.73%	73.05%	5.78
	BiE	W3A3	68.43%	76.01%	77.15%	92.00%	59.34%	90.48%	77.24%	5.21

B. Hardware Implementation

We have demonstrated the detailed hardware implementation of BiE in a PE array in Section 3.3. Here, we will give an analysis of the hardware simulation results. From Table 9, we list the major metrics and resource utility reported by the simulation tool. For the resource utility, we report the number of Digital Signal Processor Slices (DSPs), Flip-Flops (FFs), and Look-Up Tables (LUTs).

DSP usually refers to specialized hardware modules for high-speed digital signal processing in FPGAs. FF is a digital storage element that stores one bit of binary data in a digital circuit. The LUT is an FPGA's basic programmable logic unit and is used to implement logic functions. It is found that the usage of LUT in BiE is slightly more than that in BFP. This result aligns with our expectations, as BiE requires additional logic operations to handle the threshold compared to BFP and an additional shared exponent via the Type bit.

The throughput is the same for both, as it depends mainly on the multiplication calculation, which is the same for both. Although the sum operation of the exponent is different, it is operated in parallel with tail multiplication and will require fewer cycles than multiplication, so it does not affect throughput. Although BiE introduces an extra 5-bit shared exponent, its memory efficiency is averaged out by the size of the block, so the actual impact is negligible. Here, memory efficiency is smaller than BFP because each element needs a 1-bit type bit to distinguish whether it is a normal or outlier part. However, BiE's memory efficiency is still better than INT6. From a comprehensive perspective, BiE can combine accuracy and hardware efficiency.

Table 9. Hardware Efficiency of the same size PE array with different numerical representations

Method	Config	#DSPs	#FFs	#LUTs	Throughput	Memory Efficiency
FP16	-	2	182	414	1.00×	1.0×
INT	W8A8	0	86	442	1.46×	2.0×
INT	W6A6	0	90	402	1.62×	2.7×
BFP	W4A4	0	71	839	3.51×	3.7×
BFP	W3A3	0	71	775	3.96×	4.8×
BiE	W4A4	0	71	849	3.51×	2.8×
BiE	W3A3	0	71	785	3.96×	3.5×

C. Bayesian Optimization

In a GP model we place a prior distribution over $g(\boldsymbol{x})$ indexed by $\boldsymbol{x}$, in our threshold searching task, $\boldsymbol{x}$ represents threshold tenser $\mathbf{T}$:

$$g(\boldsymbol{x})|\theta \sim \mathcal{GP}\left(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'|\theta)\right), \tag{19}$$

with mean and covariance functions:

$$m_0(\boldsymbol{x}) = \mathbb{E}[g(\boldsymbol{x})], \tag{20}$$

$$k(\boldsymbol{x}, \boldsymbol{x}'|\boldsymbol{\theta}) = \mathbb{E}[(g(\boldsymbol{x}) - m_0(\boldsymbol{x}))(g(\boldsymbol{x}') - m_0(\boldsymbol{x}'))]. \tag{21}$$

Here $\mathbb{E}[\cdot]$ denotes the expectation, and the hyperparameters $\boldsymbol{\theta}$ determine the kernel function. The mean function can be assumed to be zero, for example $m_0(\boldsymbol{x}) \equiv 0$, by virtue of centering the data. Alternative choices are possible, e.g., a linear function of $\boldsymbol{x}$, but rarely adopted unless information on the form of the function is *a-priori* available. The covariance function can take many forms, the kernel function we use for threshold searching is matérn kernel, as shown in Equation (22),

$$k(\boldsymbol{x}, \boldsymbol{x}'|\boldsymbol{\theta}) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}|\boldsymbol{x} - \boldsymbol{x}'|}{\boldsymbol{\theta}} \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}|\boldsymbol{x} - \boldsymbol{x}'|}{\boldsymbol{\theta}} \right). \tag{22}$$

From this point on we omit the explicit notation of the dependence of $k(\boldsymbol{x}, \boldsymbol{x}')$ on $\boldsymbol{\theta}$. The hyperparameters $\theta_1, \ldots, \theta_l$ in this case are called the length-scales, ν is a positive parameter and K_ν is a modified Bessel function. The ν parameter effectively controls the level of smoothness of the function. As $\nu \to \infty$, we recover exactly the RBF kernel. For lower values of ν, we obtain rougher functions. When ν is a half-integer, the kernel has an especially nice form as the product of an exponential function and a polynomial function. Thus, for our threshold searching task, we set $\nu = \frac{3}{2}$. The kernel function is Equation (23),

$$k(\boldsymbol{x}, \boldsymbol{x}'|\boldsymbol{\theta}) = \left(1 + \frac{\sqrt{3}|\boldsymbol{x} - \boldsymbol{x}'|}{\boldsymbol{\theta}} \right) \exp\left(-\frac{\sqrt{3}|\boldsymbol{x} - \boldsymbol{x}'|}{\boldsymbol{\theta}} \right). \tag{23}$$

For any fixed $\boldsymbol{x}$, $g(\boldsymbol{x})$ is a random variable. A collection of values $g(\boldsymbol{x}_i)$, $i = 1, \ldots, N$, on the other hand, is a partial realization of the GP. Realizations of the GP are deterministic functions of $\boldsymbol{x}$. The main property of GPs is that the joint distribution of $g(\boldsymbol{x}_i)$, $i = 1, \ldots, N$, is multivariate Gaussian. Assuming the model inadequacy $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ is Gaussian, with the prior (19) and available data $\boldsymbol{y} = (y_1, \ldots, y_N)^\top$, we can derivative the model likelihood

$$\begin{aligned} \mathcal{L} \triangleq p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) &= \int (g(\boldsymbol{x}) + \varepsilon) dg = \mathcal{N}(\boldsymbol{y}|0, \mathbf{K} + \sigma^2 \boldsymbol{I}) \\ &= -\frac{1}{2} \boldsymbol{y}^T (\mathbf{K} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{y} - \frac{1}{2} \ln |\mathbf{K} + \sigma^2 \boldsymbol{I}| - \frac{N}{2} \log(2\pi), \end{aligned} \tag{24}$$

where the covariance matrix $K = [K_{ij}]$, in which $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $i, j = 1, \ldots, N$. The hyperparameters $\boldsymbol{\theta}$ are normally obtained from point estimates by maximum likelihood estimate (MLE) of Equation (24) w.r.t. θ.

The joint distribution of $\boldsymbol{y}$ and $g(\boldsymbol{x})$ also forms a joint Gaussian distribution. Conditioning on $\boldsymbol{y}$ provides the conditional predictive distribution $\hat{g}(\boldsymbol{x}) \sim \mathcal{N}\left(\mu(\boldsymbol{x}), v(\boldsymbol{x})\right)$, where

$$\begin{aligned} \mu(\boldsymbol{x}) &= k(\boldsymbol{x})^\top \left(\mathbf{K} + \sigma^2 \boldsymbol{I} \right)^{-1} \boldsymbol{y}, \\ v(\boldsymbol{x}) &= \sigma^2 + k(\boldsymbol{x}, \boldsymbol{x}) - k^\top(\boldsymbol{x}) \left(\mathbf{K} + \sigma^2 \boldsymbol{I} \right)^{-1} k(\boldsymbol{x}). \end{aligned} \tag{25}$$

Based on the posterior in Equation (25), we can simply calculate the improvements for a new input $\boldsymbol{x}$ as $I(\boldsymbol{x}) = \max(\hat{g}(\boldsymbol{x}) - \boldsymbol{y}^\dagger, 0)$, where $\boldsymbol{y}^\dagger$ is the current optimal and $\hat{g}(\boldsymbol{x})$ is the predictive posterior in Equation (25). The expected improvement (EI) over the probabilistic space is

$$\begin{aligned} EI(\boldsymbol{x}) &= E_{\hat{g}(\boldsymbol{x})}[\max(\hat{g}(\boldsymbol{x}) - \boldsymbol{y}^\dagger, 0)] \\ &= (\mu(\boldsymbol{x}) - \boldsymbol{y}^\dagger) \psi\left(u(\boldsymbol{x})\right) + v(\boldsymbol{x}) \phi\left(u(\boldsymbol{x})\right), \end{aligned} \tag{26}$$

where $u(\boldsymbol{x}) = (\mu(\boldsymbol{x}) - \boldsymbol{y}^\dagger)/v(\boldsymbol{x})$, $\psi(\cdot)$ and $\phi(\cdot)$ are the probabilistic density function (PDF) and cumulative density function (CDF) of a standard normal distribution, respectively.