# SBQ: Exploiting Significant Bits for Efficient and Accurate Post-Training DNN Quantization

Jiayao Ling[1,†], Gang Li[2,*], Qinghao Hu[2,3,†], Xiaolong Lin[1], Cheng Gu[1], Jian Cheng[2,3], Xiaoyao Liang[1]
[1]Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
[2]The Key Laboratory of Cognition and Decision Intelligence for Complex Systems, CASIA, Beijing, China
[3]AiRiA, Nanjing, China
arthur.ling0088@sjtu.edu.cn, gang.li@ia.ac.cn

***Abstract*—Post-Training Quantization is an effective technique for deep neural network acceleration. However, as the bit-width decreases to 4 bits and below, PTQ faces significant challenges in preserving accuracy, especially for attention-based models like LLMs. The main issue lies in considerable clipping and rounding errors induced by the limited number of quantization levels and narrow data range in conventional low-precision quantization.**

**In this paper, we present an efficient and accurate PTQ method that targets 4 bits and below through algorithm and architecture co-design. Our key idea is to dynamically extract a small portion of significant bit terms from high-precision operands to perform low-precision multiplications under the given computational budget. Specifically, we propose Significant-Bit Quantization (SBQ). It exploits a product-aware method to dynamically identify significant terms and an error-compensated computation scheme to minimize compute errors. We present a dedicated inference engine to unleash the power of SBQ. Experiments on CNNs, ViTs, and LLMs reveal that SBQ consistently outperforms prior PTQ methods under 2∼4-bit quantization. We also compare the proposed inference engine with state-of-the-art bit-operation-based quantization architectures TQ and Sibia. Results show that SBQ can achieve the highest area and energy efficiency.**

## I. Introduction

PTQ has been demonstrated as an effective technique for accelerating DNN inference. Compared to Quantization-Aware Training (QAT), PTQ offers two notable advantages. First, It requires minimal to no labeled data, which is crucial for maintaining data privacy. Second, PTQ operates offline without necessitating a complex training process, significantly reducing computational demands. This is particularly advantageous for LLMs [18, 23]. Therefore, PTQ has garnered increasing attention from both the academic and industrial communities.

While PTQ performs well in high-precision scenarios, it is increasingly challenging to preserve accuracy as the bit-width decreases to 4 bits and below. The main issue lies in the significant quantization errors caused by the limited number of quantization levels and narrow data range in conventional uniform quantization. On the one hand, the narrow data range can lead to significant clipping errors for outliers with large magnitudes, which has been demonstrated very harmful to model accuracy, particularly for attention-based models [4, 21]. On the other hand, the restricted number of quantization levels poses a challenge in accurately representing diverse data distributions at fine granularities, thereby introducing significant rounding errors. Moreover, unlike QAT, PTQ cannot mitigate the above quantization errors through network training. Consequently, to maintain high model accuracy, the low-precision values in the desired PTQ method should possess broad data ranges and fine-grained representation abilities to minimize clipping and rounding errors simultaneously.

In this paper, we rethink quantization from a bit-operation perspective and propose a novel PTQ method through algorithm and hardware co-design. Our fundamental idea is to dynamically extract a small portion of significant bit terms from high-precision weights and activations to perform low-precision multiplications while adhearing to specific computational budget. This mechanism has two distinct merits. First, the extracted significant bit terms inherently align with the same data range and quantization levels as the high-precision models, thus holding greater potential to minimize quantization errors compared to conventional methods. Second, starting from high-precision models allows to inherit the memory efficiency of these base models (such as W4A8 model) and further reduce computational complexity, which is crucial in memory-bound scenarios like LLM inference. Accordingly, we propose Significant-Bit Quantization (SBQ). It encodes the high-precision weights and activations into collections of 2-bit nonzero bit terms. During inference, SBQ exploits a product-aware method to dynamically identify and calculate a subset of term products that contribute significantly to each high-precision product within a specific computational budget. Meanwhile, SBQ retains a small number of less significant terms at runtime, which are used for error compensation whenever the computational budget cannot be fully utilized due to term sparsity. By doing these, SBQ can utilize as many significant terms as possible to achieve flexible low-precision quantization with the same data range and quantization levels as high-precision quantization.

While promising, SBQ is challenging to witness performance gains on general processors. First, modern GPUs cannot efficiently handle fine-grained bit operations, particularly those below 4 bits. Second, SBQ introduces data dependency and irregular sparsity in the error-compensated computation, significantly heightening the challenges for parallel processing on GPUs. Therefore, we also propose an efficient inference engine to unleash the potential of SBQ. To summarize, this paper

[†]Equal contributions
[*]Corresponding author

978-3-9826741-0-0/DATE25/© 2025 EDAA

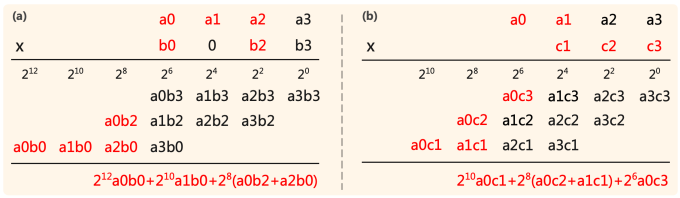
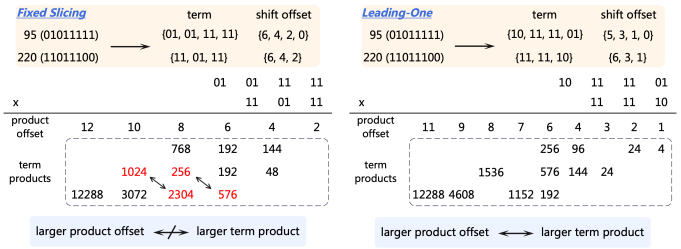



Fig. 1: Our proposed product-aware term selection method. Significant terms are selected according to the magnitude of product offsets to minimize product errors. The computational budget is four term products per multiplication in this example.




(a) (b)

Fig. 2: An illustration of term encoding methods: (a) fixed slicing, (b) proposed leading-one. Fixed slicing can result in a magnitude mismatch between product offset and term product, while our leading-one encoding can ensure a precise alignment, enabling an unbiased product-aware term selection.

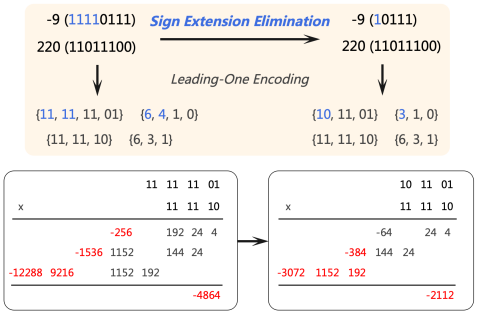



Fig. 3: An example of the optimized leading-one term encoding with sign extension elimination. The computational budget is four term products. By removing sign extension bits, our method can produce more compact terms and effectively reduce compute errors.

makes the following contributions:

- We propose Significant-Bit Quantization (SBQ), a novel bit-operation-based PTQ algorithm. It builds upon high-precision quantized models to inherit broad data ranges and rich quantization levels, as well as memory efficiency. During inference, SBQ exploits a product-aware method to dynamically identify significant bit terms and utilizes an error-compensated computation to minimize errors.
- We propose a dedicated inference engine for SBQ. It exploits intersection-based term generators to efficiently identify significant and less significant terms in product-aware term selection. It also employs 2-bit multipliers and low-cost term queues to enable flexible error-compensated computations.
- Extensive experiments on CNNs, ViTs, and LLMs reveal that SBQ consistently outperforms state-of-the-art PTQ methods under 2∼4-bit quantization. We also compare the proposed inference engine with state-of-the-art bit-operation-based quantization architectures TQ [9] and Sibia [8]. Results show that our SBQ inference engine can achieve the highest area and energy efficiency.

## II. Significant-Bit Quantization

### A. Product-Aware Term Selection

In SBQ, we denote a bit term as a 2-bit nonzero slice with corresponding shift offset. To represent an 8-bit input operand, we start with a simple case where the 8-bit integer is sliced into terms with the fixed shift offsets of 0, 2, 4, and 6. For example, 110 (01101110) can be represented as the term collection of $\{01 * 2^6, 10 * 2^4, 11 * 2^2, 10\}$. Under 2-bit granularity, an 8-bit integer contains at most four terms, and the number of terms can be reduced for an increased bit sparsity. In SBQ, we employ a product-aware method to select significant terms. Specifically, given a computational budget of $B$ term products ($4B$ BitOps) per multiplication, our goal is to respectively select $m$ and $n$ terms from weight and activation to calculate $k$ term products with the maximum magnitudes while satisfying the constraint $k \leq B$. To achieve this, we utilize the shift offsets of term products to guide the term selection, as products with larger shift offsets tend to have larger magnitudes, and the product offsets can be easily obtained by simple additions between term offsets. Figure 1 illustrates the mechanism of product-aware term selection. It can be seen that, for the same input operand $a0a1a2a3$, the product-aware method can adaptively select different terms for different multiplications to reduce product errors. For a given computational budget, such as 16 BitOps, our method can actually achieve a dynamic mixed-precision quantization, such as 2/8-bit and 4/4-bit, at the granularity of individual multiplications, which is very flexible. Note that in the example (b) in Figure 1, $a0c3$, $a1c2$, and $a2c1$ share the same product offset. Since their exact magnitudes are unknown before calculation, in this case we simply select terms from top to bottom in the product table to reduce complexity.

### B. Leading-One Term Encoding

In the product-aware term selection method, a basic assumption is that term products with larger shift offsets have larger magnitudes. This holds true when the term granularity is 1 bit. However, we find that in the case of 2-bit granularity, the naive fixed slicing in Section II-A does not guarantee the validity of this assumption. Figure 2a shows an example of term encoding with fixed slicing. Notice that between adjacent columns, the term product with a smaller shift offset (located on the right) may have a larger magnitude. In this scenario, selecting terms based on the magnitude of product offsets is misleading and cannot guarantee minimizing product errors. Through detailed analysis, we observe that the fundamental cause for the mismatch is the excessively large dynamic range

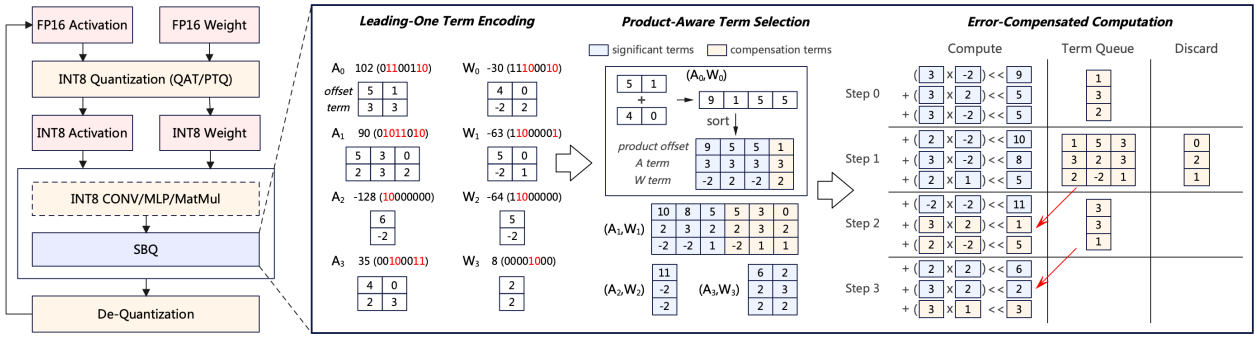



Fig. 4: An example of the proposed SBQ algorithm for inner product calculation ($A_0W_0 + A_1W_1 + A_2W_2 + A_3W_3$) between two INT8 vectors. The computational budget per multiplication ($B$), the maximum number of compensation terms per multiplication ($C$), and the depth of term queue ($D$) is set as (3,3,3), which is equavalent to 3/4-bit quantization (12 BitOps per multiplication). SBQ can be applied by simply replacing the original high-precision integer operations at runtime *without* additional efforts.

of 2-bit terms. Taking the unsigned integer as an example, in fixed slicing, each 2-bit term has a value range of $[1*2^n, 3*2^n]$. Let's say we have two term products located in the adjacent columns, denoted as $a_0b_0 * 2^n$ and $a_1b_1 * 2^{2n}$, it is possible that when $a_0$ and $b_0$ are larger while $a_1$ and $b_1$ are smaller, we may encounter a situation where $a_0b_0 > a_1b_1 * 2^2$, such as $a_0 = 3$, $b_0 = 3$, $a_1 = 1$, and $b_1 = 1$. To address this issue, we employ a leading-one term encoding in SBQ. For a given high-precision integer, we progressively extract several 2-bit terms starting from the most significant nonzero bit, as shown in Figure 2b. In this case, the data range of each term is restricted to $[2*2^n, 3*2^n]$, which can ensure a strict match between shift offset and magnitude of a term product, effectively eliminating biases in the process of product-aware term selection. It is worth noting that under leading-one encoding, the data range of terms with a shift offset of 0 remains $[1, 3]$. However, it is easy to prove that the product with a offset of 0 always has the minimum magnitude.

For negative values, we observe that encoding directly from the most significant nonzero bit may result in large product errors. The main reason lies in the sign extension bits. Due to the relatively large shift offsets, these sign extension bits have a high probability of being selected as terms, leading to an increased bias, especially for negative values with small magnitudes. As seen from Figure 3, when the sign extension bits are encoded as terms, the product is changed from -1980 to -4864, leading to significant errors. To address this issue, we dynamically remove the sign extension bits during term encoding in SBQ. Another advantage of sign extension elimination is the ability to represent values with fewer terms. For example, with the optimized encoding, the average number of terms in INT8 values can be reduced from 3 (with fixed slicing) to 2.61. This aids in achieving more precise computations for a specific computational budget.

### C. Error-Compensated Computation

Given a computational budget of $B$ term products per multiplication, the low-precision computation between weight and activation can be achieved by calculating $k$ ($k \leq B$) term products with the largest magnitudes. Assume that each operand contains $m$ and $n$ terms, respectively, two situations may occur: 1) $mn < B$. In this case, the computation introducts no errors, but the budget cannot be fully utilized. For example, when performing 4-bit inference ($B = 4$) between 01000000 and 00110000, the utilization of the computational budget is only 25%; 2) $mn \geq B$. The budget can be fully utilized in this case, but the remaining $mn - B$ term products with smaller magnitudes cannot be calculated. To achieve more precise computations, a desired approach is to utilize the surplus budget of a multiplication (situation 1) to calculate the less significant terms of the others (situation 2). To this end, we propose an error-compensated computation in SBQ. Specifically, we introduce a small term queue during inference. When the total number of term pairs in a multiplication exceeds the computational budget, i.e., $mn > B$, we cache a maximum of $min(C, mn - B)$ term pairs in the term queue instead of directly discarding them, where $C$ denotes the maximum number of less significant terms allowed for error compensation in each multiplication. Once the budget is not fully utilized during inference, these cached terms can be immediately retrieved for computation. It is clear that this mechanism can utilize as many terms as possible within a given budget to reduce quantization error *without* introducing additional computational resources. This is especially advantageous for models in which outliers have a notable impact on accuracy, such as attention-based models [1–3, 14, 17, 19], as outliers tend to produce more terms after encoding.

Intuitively, the deeper the term queue, the more terms are likely to be calculated, enabling reduced quantization errors. However, a deep term queue will incur non-negligible overhead. Therefore, we exploit low-cost FIFOs with a depth of $D$ to implement the term queue. Once the queue is full during inference, the incoming less significant terms are directly discarded. Surprisingly, we find that this simple method can effectively maintain model accuracy.

### D. SBQ

SBQ builds upon high-precision quantized models which are referred to as base models that can be obtained from any uniform PTQ or QAT methods. During inference, all

we need is to replace the original high-precision integer CONV/MLP/MatMul operations with SBQ to achieve low-precision computations *without* additional offline calibration.

Taking the inner product calculation of two vectors as an example, the standard SBQ process is as follows. **1) Term Encoding:** For each pair of high-precision input activation and weight, we dynamically convert them into two term collections at runtime using the proposed leading-one encoding with sign extension elimination (detailed in Section II-B). Each term collection is composed of 2-bit terms with corresponding shift offsets; **2) Term Selection:** Calculating shift offsets of all term products between weight and activation, and sorting them in a descending order to identify significance of all terms for the given multiplication (detailed in Section II-A); **3) Term Computation:** For the given computational budget $B$, we pick $B$ pairs of terms with the largest product offsets and perform "multiply-and-shift" based computations. Once a multiplication's workload (total number of term pairs) is less than or exceeds the computational budget, we perform error-compensated computation (detailed in Section II-C) between successive multiplications. An example of SBQ for low-precision vector inner product calculation is shown in Figure 4.

## III. SBQ Inference Engine

To achieve efficient SBQ inference, in this work we design custom hardware components tailored to the three primary steps (term encoding, term selection, and term computation) of the algorithm. Specifically, we design inner product units (IPUs) to perform product-aware term selection and error-compensated computation for the last two steps. For step 1, since terms can be encoded in one shot and utilized in multiple inner products, we design standalone term encoders to perform leading-one encoding with sign extension elimination.

### A. Term Encoder

The role of a term encoder is to convert integer values taken from the data buffers into term sequences and associated metadata that are used as input to the IPUs. Given an input integer, the term encoder successively executes sign extension elimination and leading-one encoding to produce outputs. Taking the INT8 input as an example, the output term sequence contains up to four 2-bit terms with the corresponding 3-bit shift offsets ranging from 0 to 6. To indicate the sign for each term, the encoder generates a 4-bit mask where "1" refers to a negative term. Since the number of terms is not fixed for different inputs, we also utilize another 4-bit mask to indicate whether each term is valid in the output sequence.

### B. Inner Product Unit

The core functionalities of an IPU are twofold: 1) For each pair of input term sequences, IPU dynamically identifies significant and less significant term pairs by calculating and sorting the shift offsets of all term products; 2) Performing error-compensated computation based on two types of terms under the given computational budget. Figure 5(a) illustrates the micoarchitecture of an IPU. It consists of 16 parallel compute lanes. For a vector inner product of length $L$, each lane is responsible for calculating $L/16$ low-precision multiplications. Once the computations of all lanes are complete, the partial results in each local output register are aggregated in a systolic manner. To overlap the aggregation process with computation, output registers in each compute lane are implemented in a ping-pong manner.

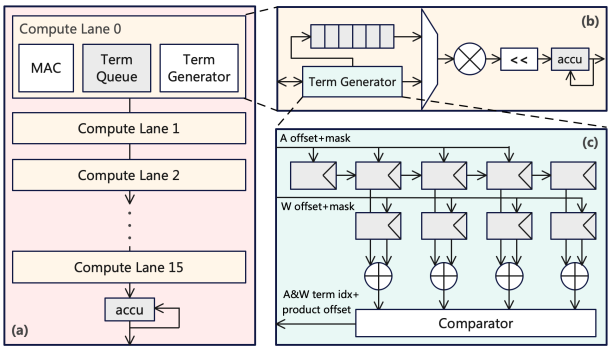



Fig. 5: Microarchitecture of (a) inner product unit, (b) compute lane (b), and (c) term generator.

Figure 5(b) shows the microarchitecture of a compute lane. The term generator is used to receive term sequences from term encoders and dynamically determines significant and less significant term pairs for subsequent processing. To flexibly support quantization of different precision, a 2-bit multiplier is employed in each compute lane to execute low-precision multiplication over multiple cycles. For example, we can achieve 4-bit quantization (16 BitOps) with a maximum of four cycles. In order to retain less significant terms for error compensation, each compute lane also contains a FIFO-based term queue with a depth of $D$. When processing starts, the term generator reads a pair of term sequences to calculate shift offsets of all term products and perform sorting on them. In each cycle, the term generator retrieves a pair of terms with the largest product offset for computation. It simultaneously extracts and stores a pair of less significant terms with corresponding product offset in the term queue. If the term queue is full, the incoming less significant terms will be discarded directly. Once a multiplication with fewer workloads fails to fully utilize the computational budget, a pair of terms in the term queue can be fetched for computation. If the term queue is empty, the multiplier will stay idle until the next workload arrives. From the above process, we can notice two unique characteristics: 1) When constraining $C \leq B$, the storage of less significant terms can be fully overlapped with computation; 2) The computation of both significant and less significant terms shares the same multiplier, without incurring additional MAC overhead.

### C. Term Generator

For a given computational budget, the term generator needs to distinguish significant and less significant terms based on the magnitudes of product offsets. An intuitive approach is to calculate all product offsets and then perform sorting using an algorithm such as bubble sort. However, we observe that classic sorting algorithms exhibit significant area overhead and

are challenging to achieve sorting in one cycle, even under a moderate working frequency. To address this issue, we develop an efficient intersection-based term generation method, which is motivated by two key observations: 1) Since only one term pair is calculated in each cycle, the term generator only needs to identify the pair with the largest product offset among the remaining term pairs in each cycle; 2) Given a computational budget $B$, each multiplication between weight and activation consumes a maximum of $B+C$ term pairs, indicating that only $B+C$ largest offsets are required to be sorted. Taking INT8 input operands as an example, when $B = C = 4$, we only need to determine the order of the 8 largest product offsets rather than that of a maximum of 16 offsets. We observe that this functionality can be efficiently achieved through a systolic-based intersection. Figure 5(c) shows the microarchitecture of the proposed intersection-based term generator. Initially, the term offsets and valid mask encoded from weight and activation are arranged in two streams in the opposite order. In each cycle, the upper stream moves a step to the right to create an overlap between the two streams. By calculating the sum of paired term offsets in the intersection region, several candidate product offsets can be obtained in parallel. Through a low-cost comparator, we can finally locate the term pair with the largest product offset. Since the largest product offsets always appear first during intersection, an exact sorting can be achieved and early terminated as necessary.

Although the intersection-based term generation can achieve efficient sorting, it presents a challenge: in each cycle, only one pair of significant terms can be identified for calculation, with no capability to simultaneously identify another pair of less significant terms for caching. Additionally, it is difficult for pipeline execution since it is unable to process the next pair of input operands while the current processing is incomplete. To address this issue, we divided all term pairs into two groups in advance, with one group only containing terms that are certain to be computed. This way, the term generation operates solely on potentially computed terms rather than all, enabling parallel execution of computation and term generation. For example, given two term sequences $\{a_0, a_1, a_2\}$ and $\{b_0, b_1, b_2\}$ where $a_0$ and $b_0$ have the largest shift offsets, we experimentally find that $a_0b_0$, $a_0b_1$, and $a_1b_0$ are definite to be computed when $B > 2$. In this case, we directly omit the generation of these determined terms. With this mechanism, we can achieve more efficient term generation and pipeline execution, avoiding the need to implement ping-pong term generators in each compute lane that would incur non-negligible area overheads.

## IV. Evaluation

### A. Methodology

We conduct experiments on CNNs (ResNet-18/50 [6], MobileNetV2 [15]), ViTs (DeiT-S [17], Swin-S [14]), and LLMs (OPT-2.7B/6.7B/13B [23], LLaMA2-7B [18]) to evaluation the effectiveness and scalability of SBQ algorithm. The base quantized models for SBQ are obtained from QDrop [20], I-ViT [12], and OmniQuant [16] in CNNs, ViTs, and LLMs

TABLE I: Accuracy comparison of SBQ with state-of-the-art PTQ algorithms on CNNs.

| Method | Prec. | ResNet-18 | ResNet-50 | MobileNetV2 |
|---|---|---|---|---|
| | fp32 | 71.06 | 76.63 | 72.62 |
| AdaQuant [7] | 4/4 | 69.60 | 75.90 | 47.16 |
| BRECQ [10] | 4/4 | 69.60 | 75.05 | 66.57 |
| QDrop [20] | 4/4 | 69.16 | 74.92 | 67.68 |
| ANT [5] | 4/4 | 69.22 | 75.68 | 66.14 |
| AQuant [11] | 4/4 | 70.03 | 75.58 | 69.15 |
| TQ [9] | 4/4 | 69.32 | 75.33 | 62.86 |
| SBQ | B=4 | **70.85** | **76.51** | **72.05** |
| AdaQuant [7] | 2/4 | 0.11 | 0.12 | 0.15 |
| BRECQ [10] | 2/4 | 64.80 | 70.29 | 53.34 |
| QDrop [20] | 2/4 | 63.36 | 67.49 | 54.31 |
| ANT [5] | 2/4 | 6.98 | 0.42 | 0.91 |
| AQuant [11] | 2/4 | 66.63 | 71.11 | 56.74 |
| TQ [9] | 2/4 | 66.75 | 73.32 | 20.17 |
| SBQ | B=2 | **69.18** | **74.92** | **67.72** |

TABLE II: Accuracy comparison of SBQ with state-of-the-art PTQ algorithms on ViTs.

| Method | Prec. | DeiT-S | Swin-S |
|---|---|---|---|
| | fp32 | 79.85 | 83.23 |
| PTQ4ViT [22] | 4/4 | 37.44 | 76.73 |
| RepQ-ViT [13] | 4/4 | 69.33 | 79.58 |
| ANT [5] | 4/4 | 75.66 | 80.93 |
| TQ [9] | 4/4 | 53.42 | 45.30 |
| SBQ | B=4 | **79.11** | **81.94** |
| PTQ4ViT [22] | 2/4 | 0.30 | 4.32 |
| RepQ-ViT [13] | 2/4 | 0.25 | 0.16 |
| ANT [5] | 2/4 | 2.27 | 0.90 |
| TQ [9] | 2/4 | 54.05 | 36.75 |
| SBQ | B=2 | **54.99** | **73.51** |

experiments, respectively. We find that W4A8 base models has the best stability across a variety of computational budgets.

For hardware evaluation, we compare the proposed SBQ inference engine with state-of-the-art TQ [9] and Sibia [8]. TQ is also a bit-operation based method for PTQ. Unlike SBQ, TQ exploits operand-aware methods to extract bit terms for calculation and directly discards the unselected terms during inference. Sibia is a 4-bit architecture for DNN inference. It also starts with high-precision weights and activations and converts each signed operand into several signed 4-bit operands at runtime. We implement SBQ inference engine and baseline architectures in Verilog, and synthesize the RTL using Synopsys Design Compiler in a TSMC 28nm technology node at 250MHz to get the area and power numbers. We also use CACTI to model the area and power consumption of on-chip data buffers.

### B. Accuracy Results

We compare SBQ with state-of-the-art PTQ algorithms on three CNN models for 4/4-bit and 2/4-bit quantization. Results

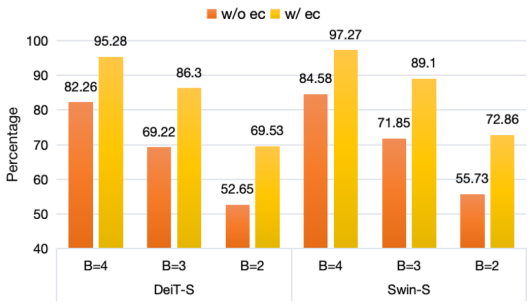



Fig. 6: The percentage of terms calculated among outlier values.

TABLE III: Perplexity (↓) of LLMs on WikiText-2 dataset.

| Method | Prec. | OPT-2.7B | OPT-6.7B | OPT-13B | LLaMA2-7B |
|---|---|---|---|---|---|
| | fp16 | 12.47 | 10.86 | 10.13 | 5.47 |
| SmoothQuant [21] | 4/4 | 2.4e4 | 1.8e4 | 7.4e3 | 83.12 |
| OliVe [4] | 4/4 | 46.08 | 45.37 | 40.09 | 206.94 |
| OmniQuant [16] | 4/4 | 15.11 | 12.24 | 11.65 | 14.26 |
| SBQ | B=4 | **12.73** | **10.90** | **10.24** | **6.23** |
| SmoothQuant [21] | 3/4 | 7.3e3 | 8.7e3 | 1.6e4 | 2.2e6 |
| Olive [4] | 3/4 | 156.18 | 275.82 | 271.91 | 400.16 |
| OmniQuant [16] | 3/4 | 19.58 | 14.75 | 13.83 | - |
| SBQ | B=3 | **13.05** | **11.29** | **10.56** | **6.35** |

TABLE IV: Area and power of an IPU and term encoder.

| Components | | Area (um$^2$) | Power (mW) |
|---|---|---|---|
| Compute Lane x1 | 2-bit MAC | 90.454 (37.2%) | 0.015 (28.3%) |
| | Term Genarator | 87.220 (35.9%) | 0.021 (39.6%) |
| | Term Queue (D=4) | 65.366 (26.9%) | 0.017 (32.1%) |
| IPU (Compute Lane x16) | | 4110.292 | 0.892 |
| Term Encoder x1 | | 145.432 | 0.062 |

are shown in Table I. It can be seen that our SBQ can consistently outperform state-of-the-art methods. Particularly, significant accuracy improvements can be achieved for MobileNetV2, where the accuracy loss of 4/4-bit quantization is less than 1%, and at least 10.98% accuracy improvement can be achieved at 2/4-bit.

We also conduct experiments on ViTs that are much more challenging in maintaining accuracy at low precision. Results are shown in Table II. It is clear that state-of-the-art PTQ methods can incur severe accuracy degradation for extremely low-precision quantization such as 2/4-bit. However, thanks to the error-compensated computation, our SBQ can preserve accuracy to a large extent. The main reason is that our method can significantly reduce the quantization error on outlier values. Figure 6 shows the percentage of terms calculated among multiplications with at least one outlier. It can be seen that with error compensation, 12.86%, 17.17%, and 17.01% more terms in outlier multiplications can be calculated when $B$ is equal to 4, 3, and 2, respectively.

To demonstrate the scalability of SBQ, we further conduct experiments on four LLMs. Results are shown in Table III. It can be seen that our SBQ can consistently achieve the lowest perplexities under 4/4-bit and 3/4-bit quantization. Surprisingly, our 3/4-bit SBQ is even more accurate than 4/4-bit baselines across all four LLMs.

TABLE V: Area and power comparison of sorting 16 4-bit product offsets using various implementations.

| | Area (um$^2$) | Power (mW) |
|---|---|---|
| Bubble Sorter | 2126.599 | 0.263 |
| Bitonic Sorting Networks | 1685.612 | 0.568 |
| Intersection-Based Sorter (this work) | **87.220** | **0.021** |

TABLE VI: Area and power comparison. Our SBQ inference engine can achieve the highest area and energy efficiency as all three architectures have the same throughput.

| | TQ [9] | Sibia [8] | SBQ |
|---|---|---|---|
| Technology | 28nm | 28nm | 28nm |
| Frequency | 250MHz | 250MHz | 250MHz |
| Number of MACs | 1536 | 1536 | 1536 |
| Buffer Size | 80KB | 80KB | 80KB |
| Area (mm$^2$) | 1.247 | 1.069 | **0.687** |
| Power (mW) | 177.4 | 100.7 | **93.3** |

### C. Hardware Evaluation

*1) **Area and Power Analysis**:* Table IV shows the area and power breakdown of an inner product unit (IPU) and a term encoder. The depth of the term queue is set to 4 (32 bits in total), which we find is sufficient to preserve model accuracy. Each entry of the term queue holds an 8-bit value consisting of a pair of 2-bit terms from weight and activation as well as a 4-bit product offset. The term generator accounts for 35∼40% area and power consumption of an IPU. However, we find it is far more efficient than other implementations to perform term sorting in SBQ. Table V shows the comparison of our intersection-based sorter with bubble sorter and bitonic sorting networks. Our method can significantly achieve 19∼25x area and power saving compared to the baselines.

*2) **Comparison with TQ and Sibia**:* Similar to SBQ, TQ and Sibia store high-precision quantized models and perform low-precision computations at runtime. In this experiment, we constrain all three architectures to have the same number of MACs and on-chip buffer size for a fair comparison. Additionally, we set the computational budget per multiplication to 3 for TQ and SBQ. Under this configuration, all three architectures can achieve nearly the same throughput for a given model. Table VI shows the results of area and power consumption. SBQ achieves a 35.7% reduction in area compared to Sibia. This is because Sibia employs 4-bit multipliers for computation and introduces additional sparse processing units to reduce computational complexity. However, SBQ utilizes area-efficient 2-bit multipliers and can achieve acceleration through flexible adjustment of the computational budget without incurring additional resource overhead. Compared to TQ, SBQ demonstrates significant advantages in both area and power consumption. TQ employs 1-bit multipliers in each PE, which, despite having a smaller multiplier area and enabling more fine-grained bit selection, significantly increase the complexity of bit indexing and result in a large amount of combinational logic needed to implement PEs.

## V. Conclusion

In this paper, we present a bit-operation based PTQ method that targets 4 bits and below through algorithm and architecture co-design. We propose Significant-Bit Quantization (SBQ), which exploits a product-aware term selection method and an error-compensated computation scheme to minimize quantization errors. We also design an inference engine tailored for SBQ. Extensive experiments on CNNs, ViTs, and LLMs demonstrate that SBQ consistently outperforms state-of-the-art PTQ methods under 2∼4-bit quantization and can achieve higher area and energy efficiency than state-of-the-art bit-operation-based PTQ architectures TQ and Sibia.

## Acknowledgment

This work was supported in part by the National Key R&D Program of China (2022ZD0116406), the Natural Science Foundation of Jiangsu Province (BK20243051), and the National Natural Science Foundation of China (No.62106267).

## REFERENCES


[1] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European conference on computer vision*. Springer, 2020, pp. 213–229.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[4] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.

[5] 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 *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1414–1433.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[7] I. Hubara, Y. Nahshan, Y. Hanani, R. Banner, and D. Soudry, "Accurate post training quantization with small calibration sets," in *International Conference on Machine Learning*. PMLR, 2021, pp. 4466–4475.

[8] D. Im, G. Park, Z. Li, J. Ryu, and H.-J. Yoo, "Sibia: Signed bit-slice architecture for dense dnn acceleration with slice-level sparsity exploitation," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 69–80.

[9] H.-T. Kung, B. McDanel, and S. Q. Zhang, "Term quantization: furthering quantization at run time," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–16.

[10] 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.

[11] Z. Li, C. Guo, Z. Zhu, Y. Zhou, Y. Qiu, X. Gao, J. Leng, and M. Guo, "Efficient adaptive activation rounding for post-training quantization," 2023.

[12] Z. Li and Q. Gu, "I-vit: Integer-only quantization for efficient vision transformer inference," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 17 065–17 075.

[13] Z. Li, J. Xiao, L. Yang, and Q. Gu, "Repq-vit: Scale reparameterization for post-training quantization of vision transformers," 2023.

[14] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.

[15] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[16] 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," 2023.

[17] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *International conference on machine learning*. PMLR, 2021, pp. 10 347–10 357.

[18] 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.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

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

[21] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 38 087–38 099.

[22] Z. Yuan, C. Xue, Y. Chen, Q. Wu, and G. Sun, "Ptq4vit: Post-training quantization for vision transformers with twin uniform quantization," in *European Conference on Computer Vision*. Springer, 2022, pp. 191–207.

[23] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.