QFlash: Bridging Quantization and Memory Efficiency in Vision Transformer Attention

Sehyeon Oh[1,2] , **Yongin Kwon**[3] , **Jemin Lee**[4†]
[1]University of Science and Technology, Daejeon, Republic of Korea
[2]Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea
[3]Pusan National University, Busan, Republic of Korea
[4]Jeonbuk National University, Jeonju-si, Republic of Korea
somae0604@gmail.com, yongin@pusan.ac.kr, jemin.lee@jbnu.ac.kr

Abstract

FlashAttention improves efficiency through tiling, but its online softmax still relies on floating-point arithmetic for numerical stability, making full quantization difficult. We identify three main obstacles to integer-only FlashAttention: (1) scale explosion during tile-wise accumulation, (2) inefficient shift-based exponential operations on GPUs, and (3) quantization granularity constraints requiring uniform scales for integer comparison. To address these challenges, we propose *QFlash*, an end-to-end integer FlashAttention design that performs softmax entirely in the integer domain and runs as a single Triton kernel. On seven attention workloads from ViT, DeiT, and Swin models, QFlash achieves up to 6.73× speedup over I-ViT and up to 8.53× speedup on Swin, while reducing energy consumption by 18.8% compared to FP16 FlashAttention, without sacrificing Top-1 accuracy on ViT/DeiT and remaining competitive on Swin under per-tensor quantization. Our code is publicly available at https://github.com/EfficientCompLab/qflash.

1 Introduction

Transformer self-attention provides strong representational power, but its computation and memory cost grow as $O(N^2)$ with sequence length. This growth incurs a memory bottleneck in Vision Transformers (ViTs), where intermediate tensors must be transferred between on-chip and off-chip memory. FlashAttention [Dao *et al.*, 2022] was proposed to reduce this problem by splitting the sequence into tiles. Each tile is computed on-chip, and only the final outputs are written to off-chip memory.

However, FlashAttention and later works [Dao *et al.*, 2022; Dao, 2023; Shah *et al.*, 2024] cannot compute the softmax in one step because of the tile-based design. They update the row-wise maximum for each tile, rescale the past results, and then add the new tile results. This process needs numerical stability and therefore depends on floating-point operations.

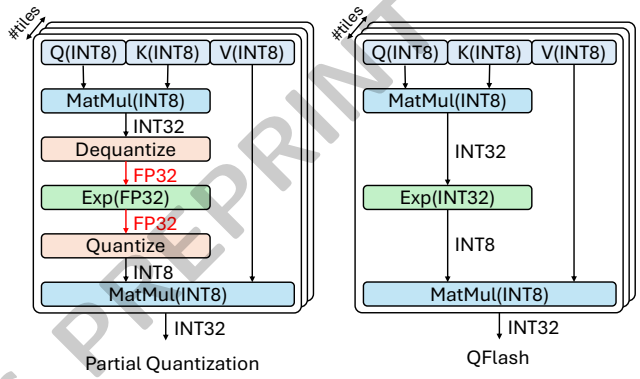

Figure 1: Comparison of partial quantization and the proposed QFlash method.

Other works on quantizing attention include I-BERT [Kim *et al.*, 2021], I-ViT [Li and Gu, 2023], QAttn [Kluska *et al.*, 2024], and INT-FlashAttention [Chen *et al.*, 2024]. Although I-BERT and I-ViT quantize the Softmax operation, they do not consider a tile-based fused attention design like FlashAttention [Dao *et al.*, 2022], leaving memory bottlenecks unresolved. QAttn and INT-FlashAttention quantize only the matrix multiplications, while the softmax still runs in floating point. As a result, there has been no work on a fully integer-only fused attention that reduces memory cost and removes floating-point operations.

Through our analysis of integer-only FlashAttention, we identify three key challenges. *(C1) Scale Explosion*: In tile-wise accumulation, approximation errors and scale changes from integer exponential operations propagate across tiles, causing numerical instability. *(C2) GPU Inefficiency*: The shift-based exponential approximation requires integer division to separate integer and fractional parts, which is significantly slower than multiplication or shift operations on GPUs. *(C3) Quantization Granularity*: Fused attention requires uniform scales across tiles for direct integer comparison. per-token quantization introduces scale mismatches that necessitate costly dequantization.

In this paper we propose QFlash, an integer-only fused attention on tile-based computation that addresses these challenges. As shown in Figure 1, QFlash keeps the tile-based

†Corresponding author
*Appendix available at: https://arxiv.org/abs/2604.25306

design but quantizes all operators including the softmax, thus the full kernel runs in the integer domain. *First*, we implement online softmax [Milakov and Gimelshein, 2018] with shift-based exponential approximation and row-wise max updates, addressing C1 by carefully managing scale propagation across tiles. *Second*, we optimize the shift-based exponential computation to minimize GPU inefficiency (C2) through efficient integer arithmetic. *Third*, we adopt per-tensor quantization granularity to maintain uniform scales across the entire attention computation, enabling direct integer comparison without dequantization (C3). This full kernel fusion reduces off-chip memory use, improves scheduling, and raises GPU utilization. QFlash therefore keeps the benefits of fused attention while providing a fully integer path that improves both latency and energy efficiency.

To validate QFlash, we implement it as an integer-only Triton kernel and evaluate it on an NVIDIA RTX 5090 across seven attention workloads (A1–A7) from ViT, DeiT, and Swin. QFlash achieves up to 6.73× speedup over I-ViT on ViT/DeiT and 8.53× on Swin, reduces energy by 18.8% over FP16 FlashAttention-2, and maintains Top-1 accuracy close to FP32 on ViT/DeiT. QFlash also improves SQNR by up to 6.7dB over I-ViT, indicating reduced quantization noise. We also observe that QFlash's IMMA utilization is 8.0%, lower than FlashAttention-2's HMMA utilization of 13.08%, but since IMMA provides twice the peak throughput on RTX 5090, QFlash delivers higher absolute performance, demonstrating that integer-only design achieves high efficiency without sacrificing accuracy.

2 Background & Related Work

2.1 FlashAttention

Standard self-attention requires computing the similarity between all query key pairs and storing the results in a matrix, leading to memory and computational complexity of $O(N^2)$ with respect to the sequence length.

As a result, both the computation cost and GPU memory usage increase rapidly when processing long sequences. FlashAttention was proposed to address this issue by leveraging the GPU memory hierarchy to avoid storing the intermediate similarity matrix in memory and instead performing computations in a tile-based manner. The key idea is to use online softmax [Milakov and Gimelshein, 2018], which incrementally updates the softmax normalization as each block is processed. In standard softmax, all similarity values must be collected at once to compute the denominator.

In contrast, online softmax processes similarity values block by block, updating the running maximum and exponential sum as new blocks are computed. This approach maintains numerical stability while eliminating the need to store the entire similarity matrix in memory. Thanks to this design, FlashAttention performs computations directly in high-speed GPU memory and writes results to global memory only when necessary. As a result, the number of memory accesses is significantly reduced, and the overall memory complexity is relaxed from $O(N^2)$ to $O(N)$.

2.2 Vision Transformer Quantization

Prior work on Vision Transformer (ViT) compression has predominantly targeted linear operators, notably matrix multiplications [Li *et al.*, 2023; Yuan *et al.*, 2022; Zhong *et al.*, 2024; Ramachandran *et al.*, 2024]. Subsequent studies extended quantization to non-linear components (Softmax, GELU, and LayerNorm) using polynomial, log-domain, or bit-shift approximations [Kim *et al.*, 2021; Lin *et al.*, 2021; You *et al.*, 2024; Hu *et al.*, 2024; Kim *et al.*, 2024]. These contributions, however, remain at the algorithmic level and do not translate the approximations into kernel-level speedups.

I-ViT [Li and Gu, 2023] advances the state of the art by bit-shift approximating Softmax and GELU and applying TVM-based intra-operator optimizations. Nevertheless, it forgoes inter-operator fusions akin to FlashAttention, leaving memory traffic largely unoptimized and thereby preserving considerable headroom for further latency reduction in quantized ViTs.

There have been some efforts to optimize quantized attention kernels by fusing attention operators [Kluska *et al.*, 2024; Chen *et al.*, 2024]. These approaches reduce computation and memory usage by quantizing the key matrix multiplications in attention. However, the core operation of attention—Softmax—is still computed in the floating-point domain, which limits their applicability for integer-only inference.

More recent works accelerate attention and nonlinear operators through hardware–software co-design. Fused Tensor Core [Jahadi *et al.*, 2025] fuses matrix multiplication and Softmax on GPUs by offloading max and sum operations to Tensor Cores, but still relies on floating-point exponentiation. PICACHU [Qin *et al.*, 2025] and QUARK [Zhao *et al.*, 2025] accelerate Softmax, GELU, and LayerNorm using dedicated hardware or CGRA-based designs. While effective, these approaches either keep Softmax in the floating-point domain or depend on specialized hardware support, limiting their generality for software-only, integer-only attention on commodity GPUs.

To address these limitations, this work proposes a fully integer-only FlashAttention kernel that quantizes all attention operations, including Softmax, using a software-only design on commodity GPUs.

2.3 Advantages of Integer-only Quantization

Peak performance of CUDA cores and Tensor Cores. For CUDA cores, the latency of integer and floating-point operations is almost the same, so full quantization is not very useful [Arafa *et al.*, 2019]. On the RTX 5090, however, measurements show that Tensor Cores can run 174K operators per cycle with HMMA (FP16) and 348K with IMMA (INT8), which is two times higher[1]. Applying quantization to all operators therefore increases IMMA use and reduces data communication, giving a clear benefit.

Performance per watt of Tensor Cores. Half-precision matrix multiply and accumulate (HMMA) uses less energy per operation, but the instruction overhead is still high, which

[1]Calculated based on 170 SMs configuration of RTX 5090 architecture.

reduces the overall performance per watt. Integer matrix multiply (IMMA) may consume a bit more energy per operation, but the instruction overhead is lower, so it reaches higher compute density in real workloads [Dally, 2023]. From this view, quantizing the model to use IMMA improves both energy efficiency and performance density. In this work, we show that using IMMA with quantization improves not only latency and throughput but also energy efficiency.

Use of integer-only accelerators. In mixed-precision designs, matrix multiplications are quantized, but key operations such as softmax still depend on floating-point units. This causes extra communication cost between integer and floating-point hardware, which limits the acceleration and adds inefficiency from floating-point units. Low-cost integer-only accelerators cannot support such mixed-precision needs, so extra floating-point units must be added in a heterogeneous chip. This increases chip area and power budget, which raises the cost of model deployment.

3 Challenges of Fully Integer FlashAttention

3.1 C1: Scale Explosion in Tile-wise Integer Accumulation

The first challenge is how to manage scaling in a tile-wise accumulation setting to avoid overflow and severe accuracy degradation. FlashAttention computes attention outputs tile by tile and incrementally accumulates each partial result into the running output. Since the accumulated result from previous tiles is reused as input for subsequent tiles, numerical stability becomes critical once quantization is applied. In integer-based FlashAttention, the exponential function is replaced with an integer approximation. The resulting approximation errors and scale variations are repeatedly injected into the accumulation process across tiles, which can lead to rapid scale growth and instability if not carefully controlled. A detailed comparison of scale management strategies is provided in Appendix B.1.

3.2 C2: Inefficiency of Shift-based Exp/Softmax on GPU

The integer softmax proposed in I-ViT [Li and Gu, 2023] typically transforms the exponential function into an $\exp 2(\cdot)$ form and implements it using shift operations combined with linear approximations. While this approach is theoretically efficient, it introduces practical inefficiencies on GPUs when applied to integer inputs.

Specifically, computing $\exp 2(x)$ requires decomposing the input into integer and fractional parts. For quantized integer inputs, this decomposition involves integer division. On GPUs, integer division has significantly higher latency than multiplication or shift operations, making it a major performance bottleneck for softmax computation. Implementation details of our GPU-friendly solution are discussed in Appendix B.2.

3.3 C3: Quantization Granularity in Integer Fused Attention

Attention inputs have a multi-dimensional structure, and quantization granularity can be applied at the tensor, head, or token level. The choice of granularity directly affects both accuracy and performance.

In FlashAttention, stable exponential computation requires subtracting the maximum value from $QK^\top$ scores, with the maximum being updated at every tile. When this process is performed in an integer-only setting, updating the maximum requires either all tiles to be computed under the same scale or the values to be dequantized into floating point for comparison. However, dequantization introduces additional computation and memory accesses, which significantly undermines the performance benefits of fused attention kernels.

For this reason, we aim to maintain a shared scale that enables direct comparison of integer values without dequantization. From this perspective, per-token quantization generates different scales for each token, making it difficult to apply in integer fused attention during max updates and accumulation. As a result, per-tensor and per-head quantization are the practical granularity choices for integer-based FlashAttention. A visual illustration of granularity constraints is presented in Appendix B.3.

4 Integer-Only Fused Attention Method

In this section, we describe in detail how the proposed QFlash algorithm operates. We first define the notation used in the description. Let b denote the bit-width, and let $\mathbb{I}_{\mathrm{int}(b)}$ denote the set of signed b-bit integers as defined in Equation (1). Let s_X be the scale factor, and let $\hat{\mathbf{X}}$ denote the integer quantization of $\mathbf{X}$.

$$\mathbb{I}_{\mathrm{int}(b)} := \{z \in \mathbb{Z} \mid -2^{b-1} \le z \le 2^{b-1} - 1\}. \tag{1}$$

For a real matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$, the scale factor s_X and the quantized matrix $\hat{\mathbf{X}}$ are defined as in Equation (2).

$$s_X = \frac{\|\mathbf{X}\|_\infty}{2^{b-1}-1} \in \mathbb{R}, \quad \hat{\mathbf{X}} = \left\lfloor \frac{\mathbf{X}}{s_X} \right\rceil \in \mathbb{I}^{m\times n}_{\mathrm{int}(b)}. \tag{2}$$

QFlash maintains the tile-based fused attention computation of FlashAttention [Dao *et al.*, 2022] while quantizing all operators, including the softmax, to achieve both data size reduction and compute efficiency. The detailed procedure follows Figure 2 and Algorithm 1. One execution of the outer loop in Algorithm 1 computes a single output tensor $\hat{\mathbf{O}}_i$. As shown in Figure 2, the process consists of the following eleven steps: ① Compute Query–Key Score → ② Compute Row-Max → ③ Update Max → ④ Compute Score Intensity → ⑤ Compute Max-Offset Factor → ⑥ Requantization → ⑦ Compute Row-Sum → ⑧ Compute Value MatMul → ⑨ Accumulate Denominator → ⑩ Accumulate Numerator → ⑪ Normalize.

4.1 MatMul

QFlash performs integer matrix multiplications for ① Compute Query–Key MatMul and ⑧ Compute Value MatMul. These operations can be expressed as the dot product of two quantized matrices $\hat{\mathbf{A}} \in \mathbb{I}^{B_r\times d}_{\mathrm{int8}}$ and $\hat{\mathbf{B}} \in \mathbb{I}^{d\times B_c}_{\mathrm{int8}}$. Here, B_r denotes the size of the query block, B_c denotes the size of the key/value block, and d represents the inner dimension of the

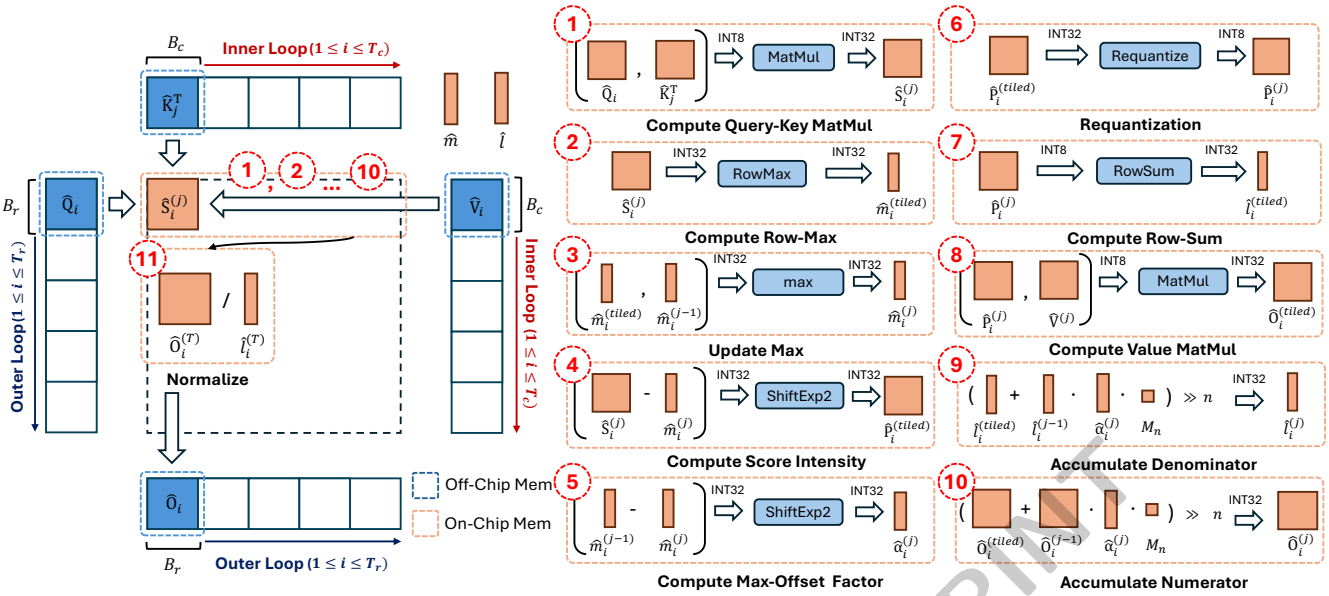

Figure 2: Proposed QFlash forward pass (Algorithm 1), showing the integer-based attention pipeline with quantization, shift-exp softmax approximation, and blockwise accumulation.

multiplication. The inputs are given in int8 format, and the results are accumulated in int32 format. Equation (3) defines the scaling relationship corresponding to this integer matrix multiplication.

$$\hat{\mathbf{C}} = \hat{\mathbf{A}} \cdot \hat{\mathbf{B}} \in \mathbb{I}_{\text{int32}}^{B_r \times B_c}, \quad s_C = s_A \cdot s_B \in \mathbb{R}. \tag{3}$$

This operation can be highly optimized on GPUs and dedicated accelerators, replacing floating-point multiplications to significantly reduce both computational cost and memory access overhead.

4.2 Compute Row-Max and Update Max

For a matrix $\hat{\mathbf{X}} \in \mathbb{I}_{\text{int32}}^{B_r \times B_c}$, the ② Compute Row-Max step calculates the maximum value of each row to obtain a vector $\hat{m} \in \mathbb{I}_{\text{int32}}^{B_r}$. Subsequently, in the ③ Update Max step, these values are compared with those from the previous tile to update the row-wise maxima progressively.

$$\hat{m}_i = \max_{0 \le j < B_c} \hat{\mathbf{X}}[i, j], \quad \hat{m} = \left[\hat{m}_1, \ldots, \hat{m}_{B_r}\right] \in \mathbb{I}_{\text{int32}}^{B_r}. \tag{4}$$

Equation (4) defines the extraction of the maximum value from each row, which serves as the reference point for the subsequent softmax computation. By subtracting the row-wise maximum from all elements, potential overflow in the exponential calculation is prevented. In addition, *Update Max* incorporates the running maximum across tiles during block-wise operations, ensuring stable numerical computation throughout the entire sequence. This tile-wise max update requires consistent scaling for direct integer comparison, which motivates Challenge C3.

4.3 Compute Score Intensity and Max-Offset Factor

In QFlash, the exponential operation is used in step ④ Compute Score Intensity and step ⑤ Compute Max-Offset Factor. The Score Intensity step computes the exponential term of softmax in integer form to measure attention strength. The Max-Offset Factor step uses the Row-Max value to keep the exponential operation stable.

We adapt the ShiftExp algorithm from I-ViT to approximate the exponential function of softmax with integer arithmetic. The key of softmax is the e^x computation. By multiplying with the constant $\log_2(e)$, we can rewrite it as a power of two. The input matrix is $\hat{\mathbf{X}} \in \mathbb{I}_{\text{int32}}^{m \times n}$ with scale factor $s_X \in \mathbb{R}$. After Row-Max is applied, $\hat{\mathbf{X}}$ always has negative values. therefore, the exponential is applied only to negative inputs. This process is given in Equation (5).

$$e^{s_X \cdot \hat{\mathbf{X}}} = 2^{(s_X \cdot \log_2 e) \cdot \hat{\mathbf{X}}} = 2^{\tilde{s}_X \cdot \hat{\mathbf{X}}}, \tag{5}$$

Here, $\tilde{s}_X = s_X \cdot \log_2 e$. Equation (6) rewrites the input $\hat{\mathbf{X}}$ as an integer part $\mathbf{Q} \in \mathbb{Z}_{\le 0}$ and a fractional part $\mathbf{R}$. This decomposition typically requires integer division on GPUs, which motivates Challenge C2.

$$2^{\tilde{s}_X \cdot \hat{\mathbf{X}}} = 2^{(\tilde{s}_X \cdot \mathbf{R}) + \mathbf{Q}} \tag{6}$$

Since the fractional part $\mathbf{R}$ lies in $(-1, 0]$, it can be replaced with a simple linear approximation as in Equation (7).

$$2^{(\tilde{s}_X \cdot \mathbf{R})} \approx \frac{\tilde{s}_X \cdot \mathbf{R}}{2} + 1 = \tilde{s}_X \cdot \left(\frac{\mathbf{R}}{2} + \left\lfloor \frac{1}{\tilde{s}_X} \right\rfloor \right) \tag{7}$$

Note that $\left\lfloor \frac{1}{\tilde{s}_X} \right\rfloor$ is a constant determined only by $\tilde{s}_X$. Thus, it is precomputed outside the kernel, and the kernel performs

Algorithm 1 QFlash Algorithm

Require: $\hat{\mathbf{Q}}, \hat{\mathbf{K}}, \hat{\mathbf{V}} \in \mathbb{I}_{\text{int8}}^{N\times d}$, scale factors $s_Q, s_K, s_V \in \mathbb{R}$ and $s = s_Q \cdot s_K \cdot d^{-1/2} \cdot \log_2 e$, block sizes B_c, B_r, multiplier M_r, shift bits r.
Divide $\hat{\mathbf{Q}}$ into $T_r = \lceil N/B_r \rceil$ blocks $\hat{\mathbf{Q}}_1, \ldots, \hat{\mathbf{Q}}_{T_r}$ of size $B_r \times d$, and divide $\hat{\mathbf{K}}, \hat{\mathbf{V}}$ into $T_c = \lceil N/B_c \rceil$ blocks $\hat{\mathbf{K}}_1, \ldots, \hat{\mathbf{K}}_{T_c}$ and $\hat{\mathbf{V}}_1, \ldots, \hat{\mathbf{V}}_{T_c}$ of size $B_c \times d$ each.
Divide output $\hat{\mathbf{O}} \in \mathbb{I}_{\text{int8}}^{N\times d}$ into T_r blocks $\hat{\mathbf{O}}_1, \ldots, \hat{\mathbf{O}}_{T_r}$ of size $B_r \times d$ each.
for $i = 1$ **to** T_r **do**
  Load $\hat{\mathbf{Q}}_i$ from $\hat{\mathbf{Q}}[(i-1)B_r : iB_r, :]$ from off-chip to SRAM
  On chip, initialize $\hat{\mathbf{O}}_i = \mathbf{0} \in \mathbb{Z}^{B_r \times d}$, $\hat{l}_i^{(0)} = \mathbf{0} \in \mathbb{Z}^{B_r}$, $\hat{m}_i^{(0)} = -2^{21} \in \mathbb{Z}^{B_r}$
  for $j = 1$ **to** T_c **do**
    Load $\hat{\mathbf{K}}_j^\top$ from $\hat{\mathbf{K}}^\top[:, (j-1)B_c : jB_c]$ and $\hat{\mathbf{V}}_j$ from $\hat{\mathbf{V}}[(j-1)B_c : jB_c, :]$ from off-chip to SRAM
    On chip, compute $\hat{\mathbf{S}}_i^{(j)} \leftarrow \hat{\mathbf{Q}}_i \hat{\mathbf{K}}_j^\top$ ▷ ①
    On chip, compute $\hat{m}_i^{(j)} \leftarrow \max(\hat{m}_i^{(j-1)}, \text{rowmax}(\hat{\mathbf{S}}_i^{(j)}))$ ▷ ②, ③
    On chip, compute $\hat{\alpha}_i^{(j)} \leftarrow \text{ShiftExp2}(\hat{m}_i^{(j-1)} - \hat{m}_i^{(j)}, s)$ ▷ ④
    On chip, compute $\hat{\mathbf{P}}_i^{(\text{tiled})} \leftarrow \text{ShiftExp2}(\hat{\mathbf{S}}_i^{(j)} - \hat{m}_i^{(j)}, s)$ ▷ ⑤
    On chip, compute $\hat{\mathbf{P}}_i^{(j)} \leftarrow \text{Requantization}(\hat{\mathbf{P}}_i^{(\text{tiled})}, M, b)$ ▷ ⑥
    On chip, compute $\hat{l}_i^{(j)} \leftarrow \text{ScaleRelease}(\hat{l}_i^{(j-1)}, \hat{\alpha}_i^{(j)}, s) + \text{rowsum}(\hat{\mathbf{P}}_i^{(j)})$ ▷ ⑦, ⑨
    On chip, compute $\hat{\mathbf{O}}_i^{(j)} \leftarrow \text{ScaleRelease}(\hat{\mathbf{O}}_i^{(j-1)}, \hat{\alpha}_i^{(j)}, s) + \hat{\mathbf{P}}_i^{(j)} \hat{\mathbf{V}}_j$ ▷ ⑧, ⑩
  end for
  On chip, compute $\hat{\mathbf{O}}_i \leftarrow \lfloor \hat{\mathbf{O}}_i^{T_c} / \hat{l}_i^{T_c} \rfloor$ ▷ ⑪
  Write $\hat{\mathbf{O}}_i$ to off-chip as the i-th block of $\hat{\mathbf{O}}$
end for
Return the output $\hat{\mathbf{O}} \in \mathbb{I}_{\text{int8}}^{N\times d}$ and the output scale factor $s_O = s_V$.

only element-wise integer operations. Finally, Equation (8) shows the combination of the integer part and the approximated fractional part to compute the exponential.

$$e^{s_X \cdot \hat{\mathbf{X}}} \approx \tilde{s}_X \cdot \left(\frac{\mathbf{R}}{2} + \left\lfloor \frac{1}{\tilde{s}_X} \right\rceil \right) \cdot 2^{\mathbf{Q}} \tag{8}$$

This formulation enables an efficient integer-only approximation of the exponential operation using simple arithmetic and bit-shift operations.

4.4 Requantization

The result of the matrix multiplication is stored as $\hat{\mathbf{X}} \in \mathbb{I}_{\text{int32}}^{B_r \times B_c}$. To be used in subsequent operations, this value must be converted back into the range of $\hat{\mathbf{Y}} \in \mathbb{I}_{\text{int8}}^{B_r \times B_c}$ through the ⑥ Requantization step. This is achieved by applying a scaling transformation that reduces the integer range. First, we define a fixed-point multiplier as in Equation (9). Here, the scale ratio is converted into a binary logarithm to compute n, and the shift size r is set by considering the bit-width b.

$$n = \left\lfloor \log_2 \left(\frac{s_X}{s_Y} \right) \right\rfloor, \quad r = b - n. \tag{9}$$

Based on this, the requantization is performed as shown in Equation (10).

$$M_r = \left\lfloor \frac{s_X}{s_Y} \cdot 2^r \right\rceil, \quad \hat{\mathbf{Y}} = \left(\hat{\mathbf{X}} \cdot M_r \right) \gg r. \tag{10}$$

We precompute n, r, and $\left\lfloor \frac{s_X}{s_Y} \cdot 2^r \right\rceil$ outside the kernel, since they depend only on the scale factors and bit-width and are independent of the input $\hat{\mathbf{X}}$.

4.5 Compute Row-Sum

For softmax normalization, the denominator requires the sum of each row. Thus, for the matrix $\hat{\mathbf{X}} \in \mathbb{I}_{\text{int8}}^{B_r \times B_c}$, the ⑦ Compute Row-Sum step calculates the element-wise sum of each row.

$$\hat{s}_i = \sum_{j=1}^{B_c} \hat{\mathbf{X}}[i, j], \quad \hat{s} = \left[\hat{s}_1, \ldots, \hat{s}_{B_r} \right] \in \mathbb{I}_{\text{int8}}^{B_r}. \tag{11}$$

Equation (11) describes the computation of the row-wise sums, where $\hat{s}_i$ represents the sum of the elements in the i-th row. These values serve as the key denominator terms in the softmax normalization.

4.6 Accumulate Numerator and Denominator

For softmax normalization, steps ⑨ Accumulate Denominator and ⑩ Accumulate Numerator must be performed. If these accumulations are carried out directly in the quantized domain, overflow may occur. A common approach dequantizes values before accumulation, but this introduces floating-point operations and weakens integer-only benefits. Related to Challenge C1, we approximate the inverse scale as an integer to perform division, enabling stable numerator/denominator accumulation and integer-only softmax normalization without overflow. We refer to this operation as

Workload	Source	#Win	H	N	D	Shape ($W \times H \times N \times D$)
A1	ViT/DeiT-Tiny	✗	3	197	64	$(3 \times 197 \times 64)$
A2	ViT/DeiT-Small	✗	6	197	64	$(6 \times 197 \times 64)$
A3	ViT/DeiT-Base	✗	12	197	64	$(12 \times 197 \times 64)$
A4	Swin-T/S Stage-1	64	3	49	32	$(64 \times 3 \times 49 \times 32)$
A5	Swin-T/S Stage-2	16	6	49	32	$(16 \times 6 \times 49 \times 32)$
A6	Swin-T/S Stage-3	4	12	49	32	$(4 \times 12 \times 49 \times 32)$
A7	Swin-T/S Stage-4	1	24	49	32	$(1 \times 24 \times 49 \times 32)$

Table 1: Unique attention workload configurations used in ViT/DeiT and Swin inference at 224×224. $\#Win$ denotes the number of windows (only applicable to Swin), H the number of heads, N the context length, and D the head dimension.

ScaleRelease, whose definition and analysis are provided in Appendix B.1.

4.7 Normalization

After the inner loop, step ⑪ performs *normalization*. Both the numerator and denominator are accumulated as integers, and their ratio corresponds to softmax normalization without explicitly materializing the full score matrix. Since the scale factors cancel out during normalization, dequantization is implicitly handled. Importantly, we implement this division *entirely in integer arithmetic*, eliminating floating-point operations while producing int8 outputs directly usable for downstream computation.

5 Evaluation

All experiments were run on an NVIDIA GeForce RTX 5090 GPU with TVM 0.8, PyTorch 2.7.1 (CUDA 12.8), and Triton 3.3.1. We evaluated latency, energy, and accuracy at the operator level and then tested Top-1 accuracy against I-ViT, I-BERT, FQ-ViT, FlashAttention-2, INT-FlashAttention (Full/Half), and QAttn.

We extract seven unique attention workloads from different ViT/DeiT/Swin model variants, as summarized in Table 1. In addition, we evaluate both batch size 1 and batch size 8 to cover real-time inference and throughput-oriented inference settings, respectively.

5.1 Inference Latency

Latency comparison. Figure 3 compares the latency of different attention kernels across the seven attention workloads listed in Table 1. Since I-ViT is a representative *integer-only* attention kernel, we measure speedup primarily against I-ViT to assess the effectiveness of our fully integer QFlash design. In the batch-1 case, QFlash achieves up to $4.54\times$ and $7.54\times$ speedup over I-ViT on ViT/DeiT and Swin workloads, respectively. In the batch-8 case, QFlash is up to $6.73\times$ and $8.53\times$ faster than I-ViT on ViT/DeiT and Swin workloads, respectively. Despite the overhead of fully integer execution, QFlash consistently outperforms the integer-only baseline and remains generally faster or comparable to FP16 and mixed-precision kernels across workloads.

Tensor Core utilization. To analyze the latency improvement of our kernel code, we used NVIDIA Nsight Compute [Leinhauser *et al.*, 2021] to profile Tensor Core utilization. We report utilization as a ratio to the peak throughput,

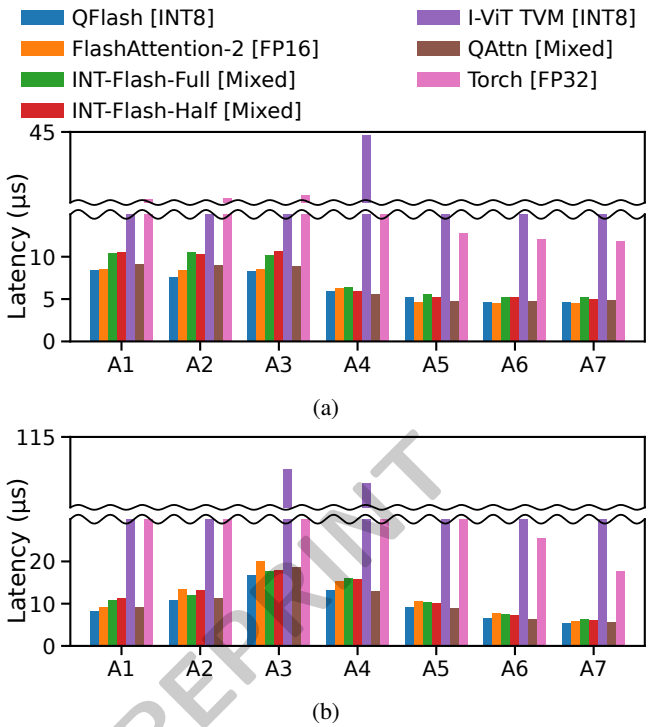

Figure 3: Latency comparison of different attention kernels across the seven attention workloads (A1–A7) listed in Table 1. Results are reported for (a) batch size 1 and (b) batch size 8.

Method	A2		A7	
	IMMA	HMMA	IMMA	HMMA
Flash-2	–	13.08%	–	4.69%
QAttn	3.75%	7.51%	1.25%	2.51%
INT-Flash-Half	3.38%	6.75%	1.12%	2.25%
INT-Flash-Full	7.18%	–	2.13%	–
QFlash (Ours)	8.00%	–	2.55%	–

Table 2: Tensor Core utilization (Peak %) for different attention implementations on workloads A2 and A7 with batch size 8.

using two workloads (A2 and A7) with batch size 8. The results are summarized in Table 2. The key observation is that on the RTX 5090, Tensor Cores achieve 174K operators per cycle with HMMA (FP16) and 348K with IMMA (INT8), meaning the peak throughput of IMMA is exactly twice that of HMMA. Therefore, even though the IMMA utilization of QFlash is only 8.0% compared to 13.08% HMMA utilization in FlashAttention-2, the higher peak throughput of IMMA means that QFlash still delivers greater overall performance. The same conclusion holds for A7.

5.2 Energy Consumption

We compared the energy consumption of five kernels on workload A2 with batch size 8. Power consumption was recorded using `nvidia-smi` [Yu *et al.*, 2023; Bridges *et al.*, 2016], and the number of operations was measured with Nsight Compute. All kernels executed the same total amount of computation. As listed in Table 3, QFlash

Workload	Flash	INT-Flash (Full)	INT-Flash (Half)	QAttn	QFlash (Ours)
A2 (B=8)	929.6	840.0	920.5	795.2	**754.6**

Table 3: Energy consumption (μJ) on workload A2 (Batch = 8).

Method	A2		A7	
	SQNR	MSE	SQNR	MSE
I-ViT	25.80	7.05e-3	25.22	9.38e-3
QAttn	34.12	1.04e-3	34.75	1.04e-3
INT-Flash-Half	38.19	4.04e-4	39.07	3.86e-4
INT-Flash-Full	36.92	5.41e-4	37.80	5.17e-4
QFlash (Ours)	32.50	1.51e-3	31.02	2.47e-3

Table 4: Quantitative evaluation of attention implementations in terms of SQNR and MSE.

shows the lowest energy consumption. There are two reasons for this result. First, QFlash achieves shorter execution time for the same workload, which reduces the integrated energy. Second, QFlash consistently uses the IMMA instruction for Tensor Cores, which provides higher energy efficiency per operation compared to HMMA [Dally, 2023; Horowitz, 2014]. The combination of these two effects makes QFlash the kernel with the lowest energy consumption under identical conditions.

5.3 Accuracy Evaluation

Quantization-level accuracy. Accuracy was measured with Signal-to-Quantization-Noise Ratio (SQNR), which indicates the relative signal-to-noise ratio after quantization, and Mean Squared Error (MSE), which quantifies the absolute error. Table 4 reports the results for two attention workloads, A2 and A7, evaluated with batch size 8. For both input sizes, QFlash shows SQNR above 30dB and MSE around 10^{-3}, which means the distortion from quantization is small. The quality is higher than the baseline integer method I-ViT, but slightly lower than INT-Flash-Half and INT-Flash-Full due to their mixed-precision design. QFlash chooses a design that favors speed and energy efficiency, while still giving acceptable accuracy and a balanced trade-off.

Top-1 evaluation setup. We measure end-to-end Top-1 accuracy on ViT, DeiT, and Swin models by quantizing only the attention modules while keeping the remaining layers in floating point to isolate the effect of attention quantization. For activation scaling, we use dynamic quantization, where scaling factors are computed on-the-fly during inference. Since I-ViT, I-BERT, FQ-ViT, INT-FlashAttention, and QAttn adopt different quantization granularities, we employ their original implementations without modification for a fair comparison: I-ViT, I-BERT, FQ-ViT, and QAttn apply ∇ per-tensor quantization, while INT-FlashAttention applies $\triangle$ per-token and $\circ$ per-head quantization. Finer granularity requires more scaling factors, leading to a trade-off between accuracy and efficiency.

Top-1 accuracy results. As reported in Table 5, QFlash maintains accuracy close to FP32 on ViT and DeiT models and exceeds I-ViT, I-BERT, and FQ-ViT on ViT backbones. On Swin, QFlash yields lower Top-1 accuracy than I-BERT and FQ-ViT. Nevertheless, QFlash achieves an efficient integer-only attention kernel with minimal scaling overhead via per-tensor quantization.

Model (FP32 Acc)	Method	BOPs (G)	Int-Only	Acc	Q	K	V
ViT-S (FP32: 81.38)	QFlash	17.2	✓	**82.24**	∇	∇	∇
	I-ViT	17.2	✓	81.19	∇	∇	∇
	I-BERT	17.2	✓	81.00	∇	∇	∇
	FQ-ViT	17.2	✓	81.06	∇	∇	∇
	INT-Flash-Full	17.2	✗	80.60	$\triangle$	$\triangle$	$\circ$
	INT-Flash-Half	34.3	✗	80.62	$\triangle$	$\triangle$	–
	QAttn	34.3	✗	80.23	∇	∇	∇
ViT-B (FP32: 85.10)	QFlash	34.3	✓	**85.84**	∇	∇	∇
	I-ViT	34.3	✓	85.02	∇	∇	∇
	I-BERT	34.3	✓	83.97	∇	∇	∇
	FQ-ViT	34.3	✓	84.82	∇	∇	∇
	INT-Flash-Full	34.3	✗	84.74	$\triangle$	$\triangle$	$\circ$
	INT-Flash-Half	68.7	✗	84.83	$\triangle$	$\triangle$	–
	QAttn	68.7	✗	84.68	∇	∇	∇
DeiT-T (FP32: 72.21)	QFlash	17.2	✓	**71.70**	∇	∇	∇
	I-ViT	17.2	✓	71.70	∇	∇	∇
	I-BERT	17.2	✓	71.20	∇	∇	∇
	FQ-ViT	17.2	✓	71.53	∇	∇	∇
	INT-Flash-Full	17.2	✗	71.64	$\triangle$	$\triangle$	$\circ$
	INT-Flash-Half	34.3	✗	71.64	$\triangle$	$\triangle$	–
	QAttn	34.3	✗	71.25	∇	∇	∇
DeiT-S (FP32: 79.85)	QFlash	34.3	✓	79.46	∇	∇	∇
	I-ViT	34.3	✓	79.53	∇	∇	∇
	I-BERT	34.3	✓	79.48	∇	∇	∇
	FQ-ViT	34.3	✓	79.39	∇	∇	∇
	INT-Flash-Full	34.3	✗	79.64	$\triangle$	$\triangle$	$\circ$
	INT-Flash-Half	68.7	✗	**79.68**	$\triangle$	$\triangle$	–
	QAttn	68.7	✗	79.43	∇	∇	∇
DeiT-B (FP32: 81.85)	QFlash	68.7	✓	81.59	∇	∇	∇
	I-ViT	68.7	✓	81.47	∇	∇	∇
	I-BERT	68.7	✓	81.59	∇	∇	∇
	FQ-ViT	68.7	✓	81.72	∇	∇	∇
	INT-Flash-Full	68.7	✗	81.85	$\triangle$	$\triangle$	$\circ$
	INT-Flash-Half	137.0	✗	**81.90**	$\triangle$	$\triangle$	–
	QAttn	137.0	✗	81.85	∇	∇	∇
Swin-T (FP32: 81.35)	QFlash	13.5	✓	80.06	∇	∇	∇
	I-ViT	13.5	✓	80.83	∇	∇	∇
	I-BERT	13.5	✓	**81.11**	∇	∇	∇
	FQ-ViT	13.5	✓	80.90	∇	∇	∇
	INT-Flash-Full	13.5	✗	80.04	$\triangle$	$\triangle$	$\circ$
	INT-Flash-Half	26.9	✗	80.05	$\triangle$	$\triangle$	–
	QAttn	26.9	✗	80.13	∇	∇	∇
Swin-S (FP32: 83.20)	QFlash	22.0	✓	81.86	∇	∇	∇
	I-ViT	22.0	✓	82.81	∇	∇	∇
	I-BERT	22.0	✓	83.09	∇	∇	∇
	FQ-ViT	22.0	✓	83.15	∇	∇	∇
	INT-Flash-Full	22.0	✗	82.11	$\triangle$	$\triangle$	$\circ$
	INT-Flash-Half	43.9	✗	**83.20**	$\triangle$	$\triangle$	–
	QAttn	43.9	✗	82.24	∇	∇	∇

Table 5: Accuracy and efficiency comparison across quantization granularities. Symbols indicate granularity: $\triangle$ per-token, $\circ$ per-head, ∇ per-tensor. For fairness, we use each prior work's original algorithm and kernel implementation, which are tailored to their target granularity.

6 Conclusion

In this paper, we proposed QFlash to address the dependence of Transformer attention on floating-point operations and the off-chip memory bottleneck from intermediate tensors. QFlash implemented the full attention process, including softmax, using INT8/INT32 integer-only operations with shift-based log/exp approximations and integer normalization. Experiments showed that QFlash achieved higher speed and precision than I-ViT and improved energy efficiency compared to FP16 FlashAttention, providing a practical solution for efficient and accurate integer-only attention in large-scale Transformer inference.

Ethical Statement

There are no ethical issues.

Acknowledgments

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2025-02214497, Development of low-level optimization program API technology for AI semiconductors) (No.RS-2023-00277060, Development of open edge AI SoC hardware and software platform).

References

[Arafa *et al.*, 2019] Yehia Arafa, AH Badawy, Gopinath Chennupati, Nandakishore Santhi, and Stephan Eidenbenz. Instructions' latencies characterization for nvidia gpgpus. *arXiv preprint arXiv:1905.08778*, 2019.

[Bridges *et al.*, 2016] Robert A Bridges, Neena Imam, and Tiffany M Mintz. Understanding gpu power: A survey of profiling, modeling, and simulation methods. *ACM Computing Surveys (CSUR)*, 49(3):1–27, 2016.

[Chen *et al.*, 2024] Shimao Chen, Zirui Liu, Zhiying Wu, Ce Zheng, Peizhuang Cong, Zihan Jiang, Yuhan Wu, Lei Su, and Tong Yang. Int-flashattention: Enabling flash attention for int8 quantization. *arXiv preprint arXiv:2409.16997*, 2024.

[Dally, 2023] Bill Dally. Hardware for deep learning. In *2023 IEEE Hot Chips 35 Symposium (HCS)*, pages 1–58. IEEE Computer Society, 2023.

[Dao *et al.*, 2022] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

[Dao, 2023] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[Horowitz, 2014] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)*, pages 10–14. IEEE, 2014.

[Hu *et al.*, 2024] Xing Hu, Yuan Cheng, Dawei Yang, Zhihang Yuan, Jiangyong Yu, Chen Xu, and Sifan Zhou. I-llm: Efficient integer-only inference for fully-quantized low-bit large language models. *arXiv preprint arXiv:2405.17849*, 2024.

[Jahadi *et al.*, 2025] Reza Jahadi, Phil Munz, and Ehsan Atoofian. Fused tensor core: A hardware–software co-design for efficient execution of attentions on gpus. *IEEE Embedded Systems Letters*, 17(5):317–320, 2025.

[Kim *et al.*, 2021] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization. In *International conference on machine learning*, pages 5506–5518. PMLR, 2021.

[Kim *et al.*, 2024] Gihwan Kim, Jemin Lee, Sihyeong Park, Yongin Kwon, and Hyungshin Kim. Mixed non-linear quantization for vision transformers. In *European Conference on Computer Vision*, pages 97–112. Springer, 2024.

[Kluska *et al.*, 2024] Piotr Kluska, Adrián Castelló, Florian Scheidegger, A Cristiano I Malossi, and Enrique S Quintana-Ortí. Qattn: Efficient gpu kernels for mixed-precision vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3648–3657, 2024.

[Leinhauser *et al.*, 2021] Matthew Leinhauser, Jeffrey Young, Sergei Bastrakov, René Widera, Ronnie Chatterjee, and Sunita Chandrasekaran. Performance analysis of picongpu: particle-in-cell on gpus using nvidia's nsight systems and nsight compute. Technical report, Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States), 2021.

[Li and Gu, 2023] Zhikai Li and Qingyi Gu. I-vit: Integer-only quantization for efficient vision transformer inference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17065–17075, 2023.

[Li *et al.*, 2023] Zhikai Li, Junrui Xiao, Lianwei Yang, and Qingyi Gu. Repq-vit: Scale reparameterization for post-training quantization of vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17227–17236, 2023.

[Lin *et al.*, 2021] Yang Lin, Tianyu Zhang, Peiqin Sun, Zheng Li, and Shuchang Zhou. Fq-vit: Post-training quantization for fully quantized vision transformer. *arXiv preprint arXiv:2111.13824*, 2021.

[Milakov and Gimelshein, 2018] Maxim Milakov and Natalia Gimelshein. Online normalizer calculation for softmax. *arXiv preprint arXiv:1805.02867*, 2018.

[Qin *et al.*, 2025] Jiajun Qin, Tianhua Xia, Cheng Tan, Jeff Zhang, and Sai Qian Zhang. Picachu: Plug-in cgra handling upcoming nonlinear operations in llms. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 845–861, 2025.

[Ramachandran *et al.*, 2024] Akshat Ramachandran, Souvik Kundu, and Tushar Krishna. Clamp-vit: Contrastive data-free learning for adaptive post-training quantization of vits. In *European Conference on Computer Vision*, pages 307–325. Springer, 2024.

[Shah *et al.*, 2024] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *Advances in Neural Information Processing Systems*, 37:68658–68685, 2024.

[You *et al.*, 2024] Haoran You, Yipin Guo, Yichao Fu, Wei Zhou, Huihong Shi, Xiaofan Zhang, Souvik Kundu, Amir Yazdanbakhsh, and Yingyan Celine Lin. Shiftaddllm: Accelerating pretrained llms via post-training multiplication-less reparameterization. *Advances in Neural Information Processing Systems*, 37:24822–24848, 2024.

[Yu *et al.*, 2023] Junyeol Yu, Jongseok Kim, and Euiseong Seo. Know your enemy to save cloud energy: Energy-performance characterization of machine learning serving. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 842–854. IEEE, 2023.

[Yuan *et al.*, 2022] Zhihang Yuan, Chenhao Xue, Yiqi Chen, Qiang Wu, and Guangyu Sun. Ptq4vit: Post-training quantization for vision transformers with twin uniform quantization. In *European conference on computer vision*, pages 191–207. Springer, 2022.

[Zhao *et al.*, 2025] Zhixiong Zhao, Haomin Li, Fangxin Liu, Yuncheng Lu, Zongwu Wang, Tao Yang, Li Jiang, and Haibing Guan. Quark: Quantization-enabled circuit sharing for transformer acceleration by exploiting common patterns in nonlinear operations. In *2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2025.

[Zhong *et al.*, 2024] Yunshan Zhong, Jiawei Hu, You Huang, Yuxin Zhang, and Rongrong Ji. Erq: Error reduction for post-training quantization of vision transformers. In *Forty-first International Conference on Machine Learning*, 2024.