PRACTICAL EDGE KERNELS FOR INTEGER-ONLY VISION TRANSFORMERS UNDER POST-TRAINING QUANTIZATION

Zining Zhang [1 2] **Bingsheng He** [2] **Zhenjie Zhang** [3]

ABSTRACT

In the domain of computer vision, transformer models have shown noteworthy success, prompting extensive research on optimizing their inference, particularly concerning their deployment on edge devices. While quantization has emerged as a viable solution for enabling energy efficiency in Convolutional Neural Networks (CNNs), achieving direct quantization of complex activation and normalization operators in transformer models proves to be a challenging task. Existing methods that rely on 64-bit integers often suffer from data truncation issues when deployed to energy-constrained edge devices, resulting in a significant loss of model accuracy. In this paper, we propose a range-constrained quantization technique for activation and normalization operators in transformers that addresses the dilemma between data range and precision. Our approach is the first 32-bit integer-based edge kernel implementation for vision transformers with post-training integer-only quantization, ensuring both efficiency and accuracy. Experimental results demonstrate a remarkable 5 times kernel speedup when deployed on two different ARM CPUs, with negligible accuracy loss in comparison to full-precision vision transformers. This innovative work is poised to significantly impact the deployment of transformer models on energy-efficient edge devices.

1 INTRODUCTION

Recently, the Transformer models have garnered substantial attention in the field of computer vision tasks, as evidenced by a number of noteworthy publications (Dosovitskiy et al., 2020; Liu et al., 2021a; Touvron et al., 2021). These models have demonstrated superior performance compared to traditional Convolutional Neural Networks (CNN). However, there are significant challenges associated with the deployment of vision transformers, given their large model size, high power consumption and high inference latency. Notably, computer vision applications, including facial and gesture recognition and object detection, are increasingly being deployed on edge devices, which are characterized by lower latency and energy requirements in business settings (Tu & Lin, 2019; Capotondi et al., 2020; Nikouei et al., 2018). As a consequence, addressing this technical gap has led to noteworthy research efforts into the development of quantization techniques for vision transformers, as seen in recent studies (Liu et al., 2021b; Ding et al., 2022; Yuan et al., 2021; Lin et al., 2022).

Quantization techniques have been extensively researched for convolutional neural networks (CNNs) (Wu et al., 2016; Zhu et al., 2020; Jacob et al., 2018). However, these techniques cannot be directly applied to transformer models due to their complex activation functions and normalizations. Unlike CNNs that use simple rectified linear units (ReLU (Dahl et al., 2013), transformer models require exponential-based activation functions in softmax (Bridle, 1989). Softmax is straightforward to compute in floating-point, but poses computational challenges in the integer domain. Additionally, Gaussian error linear unit (GELU) activation (Hendrycks & Gimpel, 2016) in transformer models includes a Gaussian error function routine, which is difficult to calculate in integers. Furthermore, quantized layer normalization (Ba et al., 2016) in transformer models necessitates on-the-fly computation of mean and standard deviations for integer tensors, unlike batch normalization (Ioffe & Szegedy, 2015) in CNNs which pre-computes these statistics during training.

Due to the huge technical challenges on full quantization, recent research studies have endeavored to employ novel transformer architectures optimized for edge devices (Zhang et al., 2020; Lin et al., 2020; Li & Gu, 2022). These methods rely on labeled data and substantial computational resources to enable transfer of the original model's power to fully quantized versions via quantization-aware training (QAT). As a result, they are not amenable to post-training quantization (PTQ). Alternative techniques (Bondarenko et al., 2021; Liu et al., 2021b; Ding et al., 2022) circumvent quantization on activation functions and layer normalizations

[1]NUS Centre for Trusted Internet and Community [2]National University of Singapore [3]Neuron Mobility Pte. Ltd.. Correspondence to: Zining Zhang <zzn@nus.edu.sg>.

Proceedings of the 6^{th} MLSys Conference, Miami Beach, FL, USA, 2023.

Table 1. Accuracy and Truncation Ratios in I-BERT (Kim et al., 2021) and FQ-ViT (Lin et al., 2022) on DeiT-S (Touvron et al., 2021) when deploying on edge devices. Each row means replacing the corresponding floating point kernels to integer kernels in the quantized model. Trunc. Ratio (%): the number of truncations in the result elements over the total number of elements. Truncations include left bit-shift and multiplications that exceed the range of int32, as well as smaller number dividing by larger number resulting in 0. Acc. (%): The image classification accuracy on ImageNet (Deng et al., 2009) evaluation dataset. In partial quantizations, it is **79.51** originally.

OPERATORS	TRUNC. RATIO % (I-BERT)	ACC. % (I-BERT)	TRUNC. RATIO % (FQ-VIT)	ACC. % (FQ-VIT)
INT. SOFTMAX	47.75	0.100	22.34	1.832
INT. GELU	92.41	0.140	N.A.	N.A.
INT. LAYERNORM	68.70	0.100	68.74	0.100

by first converting tensors into floating-point format, then performing intricate operations such as exponential and error functions using floating-point implementations, and at last converting back to integer domain. These approaches, referred to as *partial quantizations*, incur significant data conversion overheads, consume considerable on-chip power and computation cycles and fail to be deployable on integer-only chips like ARM Cortex-M (arm, 2022) and numerous edge devices (Tang et al., 2021).

To achieve full quantization of transformer-specific operators, the method proposed by I-Bert (Kim et al., 2021) involves using polynomial approximations to compute all operations within the integer domain. To deal with the high activation sensitivity of Natural Language Processing (NLP) tasks, the entire model must be re-trained from scratch (Bondarenko et al., 2021). Despite this limitation, we posit that the use of high precision integer kernels can still be directly applied to achieve PTQ in vision transformers. FQ-ViT, in line with I-Bert's approach, endeavors to achieve full quantization of vision transformers through PTQ (Lin et al., 2022). The method employs log-scale representations to enhance data precision for softmax results that are close to zero, as well as channel-wise quantization for layer normalization.

Motivated by the proficient execution of integer domain kernels proposed by I-Bert and FQ-ViT, we implement their kernels on an AMD CPU A13. To evaluate the performance of the implemented kernels, we undertake inference experiments on the image classification task of ImageNet (Deng et al., 2009). We perform kernel replacements one-at-a-time to construct partially quantized vision transformers with DeiT architecture (Touvron et al., 2021), followed by an analysis of the inference outcomes of the updated models. Contrary to our expectations, the modified models showed unanticipatedly poor accuracy as indicated in the "Acc. (*)" columns of Table 1.

In order to probe into the core reason for the observed decline in accuracy, we delved into the intermediate outcomes arising from the usage of quantized softmax, GELU and layer normalization operators. In doing so, we recorded the ratios of data truncations occasioned by integer arithmetic such as multiplications, divisions and bit shifts. The consequential truncation ratios were thereafter illustrated in the "Trunc. Ratio (*)" columns in Table 1. The outcomes indicate that truncations dominate the computations undertaken in integer kernels, leading to a pronounced loss of information.

Generally, research efforts in the field aim to enhance the accuracy of quantized models. However, such models are typically constructed and evaluated on platforms with high energy consumption. Our experiments have brought to light a scarcely explored *dilemma* regarding the balance between data representation ranges and precision in power-constrained edge devices: Higher precision calculations require an increase in the number of bits utilized, which can easily result in data truncation during multiplication, bit-shifts or division arithmetic in fixed point.

A possible solution for tackling the computational demands of edge devices is to make use of 64-bit integers. However, 64-bit integer operations are not widely supported in edge devices. For instance, instructions such as int64 *multiplications* are not supported within the Neon instruction set (neo, 2022). Furthermore, essential instructions such as divisions and rescale operators *vmulh* are not implemented for int64 on Neon as well as on more advanced SVE/SVE2 (sve, 2022) instruction sets, thus rendering int64 kernels impractical on edge devices. Additionally, the deployment of int64 arithmetic also exhibits drawbacks related to its higher memory usage and execution time, as well as reduced levels of data parallelism. Lowering the bit-width of the input data provides an alternate solution; however, in Table 1, the data truncation condition persists, especially for FQ-ViT where inputs are reduced to as low as 8-bit integers. Setting activations in transformers below 8 bits leads to significant accuracy impact, thereby necessitating the re-training of new models from scratch (Bai et al., 2020; Zhang et al., 2020; Chung et al., 2020). Moreover, directly processing low-bitwidth data during ultra-low bitwidth quantization is generally unsupported in CPUs or GPUs.

To the best of our knowledge, this article is the first work that addresses the data range and precision dilemma in the

context of full post-training quantization for vision transformers on edge devices. Our proposed methodology addresses this challenge by scrutinizing the specific features of softmax, GELU and layer normalizations. We develop practical quantized edge kernels that enhance the integer version of these operators, ensuring efficiency and accuracy. Specifically, our technique incorporates an integer *requantization* process, complemented by *additional calibrations*, aimed at adapting the overflowing numbers to a reasonable scale without compromising information integrity. We provide a detailed elucidation on the process of determining the ideal parameters for these range modifications. The integer kernels are deployed on two distinct ARM CPUs, achieving significant inference speed optimization of up to 5 times compared to floating-point arithmetic. The efficacy of the integer operators is also validated on several vision transformers, with a majority of the results demonstrating negligible accuracy degradation of approximately $1\sim2\%$ compared to full-precision versions, further emphasizing their robustness in various applications.

In summary, our contributions are as follows:

- This work first addresses the data range vs. precision dilemma in PTQ integer-only vision transformers, enabling their deployment on edge devices.
- We present novel quantization techniques that surpass the data range limitations of fixed point arithmetic. Our proposed workflow involves an integration of requantization, supplementary calibration and bit-width optimization.
- We address the essence of channel-wise layer normalization for solving the inaccuracy caused by quantized inputs to layer normalizations.
- We deploy quantized kernels on different ARM CPUs, and observe up to 5 times speedup to fp32 kernels.
- We develop the first fully quantized edge kernels that are widely applicable on various vision transformers with negligible accuracy drops.

This paper is organized as follows: Section 2 introduces the related work on quantizations and background knowledge. Section 3 explains the proposed quantization methods. Section 4 shows the results of the experiment and Section 5 summarizes the paper.

2 Related Work & Background

2.1 Neural Network Quantization

Quantization is a widely used model compression and speed-up method in the scenario of increasing demand of neural network model deployment on edge devices. Initially, quantization techniques are mostly applied for the predominant CNN models in CV tasks (Wu et al., 2016; Zhu et al., 2020; Jacob et al., 2018; Krishnamoorthi, 2018). MMM (Matrix-Matrix Multiplication) operations and convolutional operations are well developed for 8-bit activation inputs and are widely used in common mobile frameworks like Torchlite (iOS, 2022) and Tensorflow-lite (TFL, 2022). The quantizations for ReLU activations (Dahl et al., 2013) and batch normalizations (Ioffe & Szegedy, 2015) are straightforward due to their linear and static nature.

Recently, due to the great success of transformers in computer vision, the demand for compressing and accelerating transformer models keeps rising. Some work changes the architecture of transformers, in order to replace the time-consuming modules with more efficient ones (Lin et al., 2020; Li & Gu, 2022; Li et al., 2022). However, all these methods require training or fine-tuning which is not widely applicable, since the training data and computation powers are not easily accessible. There are also quantization methods that retain transformer architectures and focus on quantization parameter search (Liu et al., 2021b; Yuan et al., 2021; Ding et al., 2022; Bondarenko et al., 2021). These methods require fp32 operators on softmax, GELU and layer normalizations, thus are partial quantizations.

Unlike CNN models, transformers contain the above 3 complicated operators that are hard to be calculated in integer domains. Instead of altering these operators or using fp32 kernels, I-BERT (Kim et al., 2021) uses polynomial approximations to these operators that are commonly used in floating-point implementations but with lower order of degrees in the integer domain. However, it needs to train a new model to retain accuracy.

Inspired by I-Bert, FQ-ViT (Lin et al., 2022) develops training-free quantization methods for vision transformers. It also uses polynomial approximations for exponential functions in softmax operator. It uses log-scale bit representation for softmax results to increase data precision of close-to-zero results, but loses precision on close-to-one results. For layer normalization, it applies different quantization scales to different channels with each differing by a power-of-two, which is shown to be redundant in Section 3.2.1. Besides, FQ-ViT also ignores GELU activation, which makes it still a partial quantization method.

Note that, both I-Bert and FQ-ViT use floating-point arithmetics to simulate integer operations, e.g. using $\lfloor \cdot \rceil$ after floating-point divisions.

2.2 Uniform Quantization

In fully quantized neural networks, both the activations and weights are quantized to integer domains. In this work, we

use uniform quantizations, i.e. the real values are uniformly mapped to 2^b integers, where b is the number of bits representing the integer. Depending on whether the mapped integers are symmetric around zero, or are positive-only, we have symmetric and asymmetric uniform quantization defined in Equation 1 and Equation 2, respectively:

$$\Phi(x) = \text{clip}(\lfloor \frac{x}{S} \rceil, -2^{b-1}, 2^{b-1} - 1) \tag{1}$$

$$\Phi(x) = \text{clip}(\lfloor \frac{x}{S} \rceil + zp, 0, 2^b - 1) \tag{2}$$

where S represents the scale factor, i.e. the scale represented by 1 unit of integer. zp represents the zero point, i.e. the integer mapped by the value of 0.0 from the floating point. Usually, to balance speed and accuracy in 8 bit neural network, int8 symmetric quantization is used for weights, and uint8 asymmetric quantization is used for activations.

The quantization parameters S and zp are calculated from collected statistics from weights or activations. There are various statistical observers. One commonly used observer is MinMax observer (Wu et al., 2016). For MinMax observers of weights, the minimum value $v_{\min}$ and maximum value $v_{\max}$ are simply recorded. The value with a larger absolute $v_{\text{abs}} = \max(\text{abs}(v_{\min}), \text{abs}(v_{\max}))$ is used for the calculation of the scale factor S in symmetric quantization:

$$S = \frac{2v_{\text{abs}}}{2^b - 1}, \tag{3}$$

For quantization of activations, the activation statistics are unknown until runtime. Thus a small calibration dataset (labels are not needed) is fed to the neural network for collecting the statistics of all activations. For MinMax observers on activations, after collecting the statistics, the scale factor S and the zero point zp of asymmetric quantization are calculated as follows:

$$S = \frac{v_{\max} - v_{\min}}{2^b - 1}, zp = \text{clip}(\lfloor -\frac{v_{\min}}{S} \rceil, 0, 2^b - 1) \tag{4}$$

2.3 Transformer-specific Operators

Softmax is a normalized exponential function which approximates the hard max result. The calculation of softmax for each element $x_{1 \le i \le n}$ of input vector $x \in \mathbb{R}^n$ is as follows:

$$\text{softmax}(x)_i = \frac{e^{x_i - \max(x)}}{\sum_j e^{x_j - \max(x)}} \tag{5}$$

where the $-\max(x)$ term is for numerical stability.

GELU approximates identity function for positive inputs, and is close to zero for negative inputs. The calculation of GELU is as follows:

$$\text{GELU}(x) = x \cdot \frac{1}{2}\Big[1 + \text{erf}(x/\sqrt{2})\Big] \tag{6}$$

where erf() is the Gaussian error function, which can be depicted as follows:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{7}$$

Layer normalization is applied to transformers for normalizing each vector along the sequence. For input $\mathbf{X} \in \mathbb{R}^{L,D}$, where L is sequence length and D is the size of hidden dimension, layer normalization is calculated as:

$$\text{LN}(X) = \frac{\mathbf{X} - \mathbb{E}_{1 \le i \le D}[\mathbf{X}_i]}{\sqrt{\text{Var}_{1 \le i \le D}[\mathbf{X}_i] + \epsilon}} \times \gamma + \beta \tag{8}$$

where $\gamma, \beta \in \mathbb{R}^D$ are learned affine transform parameters.

2.4 Remez Algorithm

Nonlinear functions such as exponential or Gaussian error functions in transformer-specific operators are approximated by polynomial functions in common neural network runtimes (Abadi et al., 2016; Paszke et al., 2019). Remez algorithm is applied to minimize the maximum absolute errors of the polynomial approximations. Specifically, the linear system:

$$b_0 + b_1 x_i + \cdots + b_n x_i^n + (-1)^i E = f(x_i) \tag{9}$$

needs to be solved in each iteration of the Remez algorithm. Polynomial coefficients $b_{0 \le j \le n}$ and maximal absolute error E are $n + 2$ unknowns. At the start of each iteration, $n + 2$ control points x_i need to be settled as the extremas of the error function $E(x) = f(x) - P(x)$ where $P(x) = b_0 + b_1 x + \cdots + b_n x^n$. For the initial iteration, the control points are selected by interpolating with the roots of an orthogonal polynomial. The iterations continue until the control points are located at the same points of extremas of the error function $E(x)$.

2.5 Connecting Remez Algorithm with Quantization

For integer domain polynomial approximations, both the inputs x and coefficients b_j need to be quantized.

I-BERT converts the original floating-point polynomials to a quantized version by assuming symmetric quantizations are applied on inputs, i.e. $qS \approx x$ where $q = \Phi(x)$. Specifically:

$$\begin{aligned} P(x) &= b_0 + b_1 x + \cdots + b_n x^n \\ &\approx b_0 + b_1 qS + \cdots + b_n (qS)^n \\ &\approx (\frac{b_0}{b_n S^n} + \frac{b_1}{b_n S^{n-1}} q + \cdots + q^n) b_n S^n \\ &\approx (\lfloor \frac{b_0}{b_n S^n} \rceil + \lfloor \frac{b_1}{b_n S^{n-1}} \rceil q + \cdots + q^n) b_n S^n \\ &= Q(q) \times b_n S^n \end{aligned} \tag{10}$$

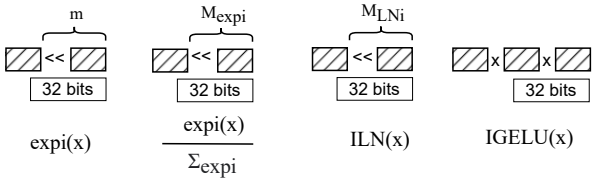


Figure 1. The truncation problems in integer version of Softmax, LayerNorm and GELU kernels.

The term $b_n S^n$ is the scale factor for the polynomial result. For $Q(q)$, there are only integer arithmetics since the quantization of polynomial coefficients can be computed beforehand.

3 METHODOLOGY

In this section, we explain different methodologies for softmax, layer normalization, and GELU are explained in Section 3.1, Section 3.2 and Section 3.3 respectively. Figure 1 shows a summary of all truncation issues in the three integer versions of the operators. Softmax has left shifts from two steps of the operation, which are intended to represent fractional results in the integer domain. LayerNorm contains a division on standard deviation of the collected data. And GELU can easily exceed the 32-bit integer range limit due to its complexity of the calculation. The following sections explain how to choose the optimal shift parameters and input bit-width to solve these problems while maintaining the accuracy of the calculations.

3.1 Integer Softmax

The Remez algorithm is utilized to approximate exponential function via polynomials and its coefficients are quantized based on input scale factors for calculating the softmax in the integer domain. Specifically, a 2nd-degree polynomial in the form of $P_{\exp}(x) = b_0 + b_1 x + b_2 x^2$ is employed to approximate the exponential function. However, as polynomial approximations only exhibit high accuracy within limited ranges, *range reduction* is required for exponential function inputs. In most practical scenarios, reduction of the range to $[-\ln 2/2, \ln 2/2]$ is performed for acceptable numerical precision. It should also be noted that, in the context of the softmax function, input values are normalized by the subtraction of the maximum element as represented in Equation 5. Consequently, all inputs to exponential functions are negative. As a result, the reduced range for polynomial approximations is set to $[-\ln 2, 0]$ in accordance with the suggestion presented in (Kim et al., 2021).

The workflow for determining the value of $\exp(x)$ within the confines of the integer domain can be concisely expressed as below: First, we need to reduce the range of the normalized input by converting input x to r where $x = -d \ln 2 + r$. The quotient d is a positive integer with the remainder r constrained in the interval $(-\ln 2, 0]$. Then we use the 2nd-degree polynomial obtained from the Remez algorithm to calculate the exponential function $\exp(r)$ in the quantized integer domain. The final step involves the calculation of $\exp(x)$ through a simple right shift of $\exp(r)$ by d bits. This is because:

$$\exp(x) = \exp(-d \ln 2 + r) = \exp(r)/(2 << d) \quad (11)$$

With in the integer domains, $\ln 2$ needs to be quantized to the same scale as x ($\approx qS$), i.e. $q_{\ln 2} = \lfloor \ln 2/S \rceil$. The range reduction process in the integer domain is represented as $q = -d_q \lfloor \ln 2/S \rceil + r_q$, where $r_q \in (\lfloor \ln 2/S \rceil, 0]$. Therefore, a fully quantized version of polynomial approximation to $\exp(x)$ is:

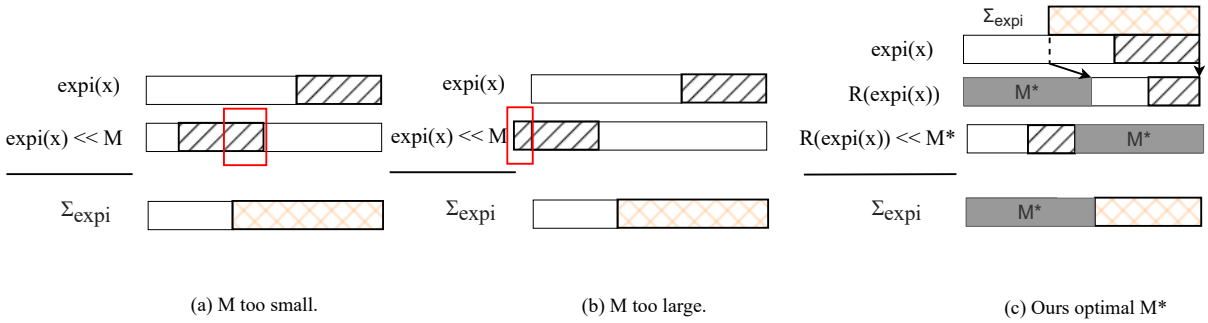
$$\begin{aligned}
\exp(x) &\approx \exp(qS) = \exp((-d_q \lfloor \ln 2/S \rceil + r_q)S) \\
&\approx \exp(-d_q \ln 2 + r_q S) = \exp(r_q S)/2^{d_q} \\
&\approx P_{\exp}(r_q S)/(2 << d_q) \\
&\approx Q_{\exp}(r_q) b_2 S^2/(2 << d_q) \\
&\approx (Q_{\exp}(r_q) >> d_q) \times b_2 S^2 \\
&= (Q_{\exp}(r_q) << (m - d_q)) \times \frac{b_2 S^2}{2^m} \\
&= \frac{b_2 S^2}{2^m} \times \text{expi}(x)
\end{aligned} \quad (12)$$

where $\frac{b_n S^2}{2^m}$ is the pre-calculated scaling factor. $P_{\exp}, Q_{\exp}$ are polynomial approximations in floating and integer domains respectively as defined in Equantion 10, and the parameters b_j are the coefficients of the polynomial.

After getting the integer-only approximation $Q_{\exp}(r_q) << (m - d_q)$ (denoted as $\text{expi}(x)$), we sum them up along the hidden dimension D and the result is denoted as $\sum_{\text{expi}}$. The softmax results for each element is thus calculated as $\frac{\text{expi}(x)}{\sum_{\text{expi}}}$. During the evaluation of softmax calculations in the integer domain, there are two potential truncation events that may result in a model deployment that is deemed unusable, as displayed in Table 1. The corresponding discussions and solutions pertaining to the two events are presented in subsequent sub-sections.

3.1.1 *Bit Truncations on division* $\frac{\text{expi}(x)}{\sum_{\text{expi}}}$

Since softmax outputs are in the range $[0, 1]$, the numerator $\text{expi}(x)$ is always smaller than the denominator (sum) $\sum_{\text{expi}}$. Usually, an additional left shift $<< M$ is applied to the numerator, and the additional scale 2^M is compensated by being divided from the output scale factor. However, this introduces a new parameter to the softmax quantization mechanism, which entangles the quantization parameters. It is hard to select an M which is able to retain the precision during softmax divisions while being immune to bit-shift truncations. Figure 2(a) shows the case where M



(a) M too small. (b) M too large. (c) Ours optimal M*

Figure 2. Comparisons among different left shift bit M selections. (a) and (b) show the hard decisions of M. Sketched black boxes represent bits occupied by $\exp(x)$ and orange ones represent those of $\sum_{\exp}$. (c) represents the proposed requantization method followed by M^* bit shift, where determination of M^* is discussed in Section 3.1.2.

is too small. The division result becomes a small integer and thus the information contained in the red box might be lost; Figure 2(b) shows the case where M is too large. The bit-shift causes truncations on the most significant bits which results in severe information loss. Determining the feasible M presents a challenging task for two reasons: 1) The range of $\exp(x)$ outputs is unknown. 2) Even if this range is ascertained, the restriction imposed on the available displacement toward the left can often lead to undesired deficits in precision.

In order to overcome the aforementioned challenges, a proposed approach involving the utilization of a requantization method is presented. This method serves to gather necessary data range information and compress the initial results from $\exp(x)$, resulting in the creation of additional space for left shifts. Firstly, let's assume we constrain the bit range of numerator $\exp(x)$ to an already known optimal range denoted as B^*. The requantization function $R(\cdot)$ is used to make sure that any $\exp(x)$ outcome is transformed to be fully constrained within this range:

$$\begin{aligned} q_{\exp} &= R(\exp(x)|B^*) \\ &= \text{clip}(\frac{S}{S_r} \bigotimes \exp(x), -2^{B^*-1}, 2^{B^*-1}-1) \end{aligned} \tag{13}$$

where S_r is the new scale of quantized value under bit range B^*. The operator $\bigotimes$ represents an integer-only implementation of multiplication between a fractional number ($\frac{S}{S_r}$) and an integer ($\exp(x)$) as introduced in (Wu et al., 2016). The quantization parameter S_r is determined in a similar process as quantizing the input tensors x to q. However, in this softmax scenario, we collect the statistics on the sum of $\exp(x)$, i.e. $\sum_{\exp}$. The sum of the exponential results provides stable statistics, which help to improve the reliability of the data collected from the calibration dataset of a limited size. Requantization is shown in the first two rows of Figure 2(c). After this requantization process, we can right shift the result by $M^* = 31 - B^*$ bits for the subsequent integer domain softmax division, i.e. $\frac{\exp(x)<<M^*}{\sum_{\exp}}$, which is shown in the 3rd row of the figure.

The reason for performing the requantization process is motivated by the following two properties, when assuming that the statistics collected in calibrations represent the data well:

1. *The left shift M^* on $R(\exp(x))$ is unlikely to cause truncation on the most significant bits.*

2. *The left shift M^* maximally uses the vacant bits on the left side.*

Regarding 1), it follows that $\exp(x)$ is bounded by the summation $\sum_{\exp}$, that is, $\exp(x) \leq \sum_{\exp}$, provided that $\sum_{\exp}$ can be restricted within B^* bits. Concerning 2), it is demonstrated by the softmax property that, $\max(\frac{\exp(x)}{\sum_{\exp}}) \approx 1$, and hence $\max(\exp(x)) \approx \sum_{expi}$. This implies that a left shift of $31 - B^*$ bits on $\max(\exp(x))$ suffices to yield an approximation of $2^{31} - 1$, serving as the maximal value of int32 integers.

3.1.2 *Decision of B^**

In order to finalize the workflow of requantization process, an optimal value for B^* and consequently for M^* must be determined. Despite the problem of truncation being resolved regardless of the value of B, selecting the appropriate value still presents a predicament. If B is excessively large, the range occupied by $R(\exp(x))$ will increase, potentially leading to a loss of precision resulting from the division process, as illustrated in Figure 2(a). Conversely, if B is too small, the requantization process will compress the original $\exp(x)$ excessively. To overcome this predicament, we propose two methods for approximating B^* that minimize the approximation error.

Output-oriented In this method, we look for the best M^*, and determine B^* by $31 - M^*$. We observe that, the integer

softmax output $\frac{R(\text{expi}(x))<<M}{\sum_{\text{exp}}}$ is bounded within M bits:

$$\begin{aligned} \frac{\exp(x)}{\sum_{\text{exp}}} &\leq 1 \\ \frac{R(\text{expi}(x))}{\sum_{\text{expi}}} &\leq 1 \\ \frac{R(\text{expi}(x)) << M}{\sum_{\text{expi}}} &\leq 2^M \end{aligned} \tag{14}$$

Based on this knowledge, we collect the statistics of softmax outputs, and determine M^* based on an additional calibration. The minimum difference in the softmax values, $\delta_{\min}$, is gathered via an additional observer along the softmax summation dimension. Presuming the calibration data representing the inference data well, a nice precision property in terms of the Unit of the Last Place (*ulp*) is anticipated for the lossless quantization of integer representation of softmax results: $1ulp \leq \delta_{\min}$, i.e.:

$$\begin{aligned} \frac{1}{2^{B^*}} &\leq \delta_{\min} \\ B^* &\geq \lceil -\log \delta_{\min} \rceil \end{aligned} \tag{15}$$

Note that $\delta_{\min}$ is calculated on the fractional result of $\frac{R(expi(x))}{\sum_{\text{expi}}}$ instead of the floating-point version of softmax. Also note that, the *minimal softmax value* should also be losslessly represented under B^* bits. In summary:

$$\delta_{\min} = \min_i O_i - O_{i-1} \tag{16}$$

$$O_i = \begin{cases} 0, & i = 0 \\ \frac{R(\text{expi}(x_i))}{\sum_{\text{expi}}}, & i > 0, i \text{ sorts } x_i \end{cases} \tag{17}$$

We directly set $B^* = \lceil -\log \delta_{\min} \rceil$ after getting the calibration results.

Loss based optimization Since B^* is a layer-wise parameter, we can apply various mixed-precision optimization techniques with losses like mean squared error (MSE), nuclear norm (Liu et al., 2021b) or Hessian matrix (Yuan et al., 2021). We demonstrate an MSE-based optimization method, which simply optimizes on the softmax approximation loss:

$$L_{\text{mse}} = \left\| \frac{\exp(x)}{\sum_{\text{exp}}} - \lfloor \frac{R(\text{expi}(x)) << M}{\sum_{\text{expi}}} \rceil \times \frac{1}{2^M} \right\|_2 \tag{18}$$

We search for the M^* which gives the lowest L_{mse} for the calibration data. The accuracy impact of *output-oriented* method and loss-based method is shown in Section 4.

3.1.3 Bit Truncations on $>> d_q$

The right shift operation in Equation 12 is another calculation that causes bit truncations which lead to information loss. A common practice is to first shift left for m bits, where m is large enough to be greater than d_q in most cases, and then perform the right shift $>> d_q$, which is equivalent to a left-shift by $m - d_q$ bits. The additional left shifts are compensated by dividing the output scale factor $b_n S^2$ by 2^m. However, the decision of m is non-trivial if we use 32-bit results of quantized MMM kernel directly as indicated in (Kim et al., 2021). Left shift by $m - d_q$ can easily exceed the 32-bit limit and cause data truncations.

Instead of using the raw quantized MMM result, we requantize the MMM result to a representation with a lower number of bits e.g. unsigned int8 (uint8). It has been verified by other work (Yuan et al., 2021) that performing $\exp()$ function on uint8 MMM outputs has little impact on the model accuracy.

In this case, we generalize polynomial approximation from symmetric quantization (Equantion 10) to asymmetric quantization. The original floating point value x is thus approximated by $(q - zp)S$ where zp is the zero point. There is a normalization term $-\max(x)$ for the input of softmax, which cancels out the zero point:

$$\begin{aligned} (q - zp)S - \max((q - zp)S) &= (q - zp - \max(q) - zp)S \\ &= (q - \max(q))S \end{aligned} \tag{19}$$

Thus the softmax version of Equation 10 is to simply replace all q terms to $q - \max(q)$ without introducing the zero point term even in the asymmetric quantization settings. Suppose b bits are used to represent the unsigned integer q, after being normalized to $q - \max(q)$, the result can occupy up to $b + 1$ bits. The maximum number of bits the quantized $Q_{\exp}(r_q)$ can thus be calculated:

$$\begin{aligned} &\lfloor \frac{b_0}{b_2 S^2} \rceil + \lfloor \frac{b_1}{b_2 S} \rceil q + q^2 \\ \leq &2^b + 2^b \times 2^{b+1} + 2^{2(b+1)} \\ \leq &2^{2(b+1)+1} \end{aligned} \tag{20}$$

which means the polynomial approximation outputs can take up to $2b + 3$ bits.

Back to the questions on how to choose the left shift amount m, since the exponential function is calculated in int32, we set $m = 31 - (2b + 3)$. In the case when $b = 8$, m is 12 which is a more acceptable range for retaining the data precision, compared to int16 or int32 where m is negative.

3.2 Integer Layer Normalization

The integer version of layer normalization has similar bit truncation problems as softmax due to left bit-shifts. To avoid this problem, we apply asymmetric quantizations on the layer normalization inputs to uint16, given that uint8

quantization significantly affects the model accuracy (Lin et al., 2022). Thus the original formula is transformed as follows:

$$\begin{aligned}\mathrm{LNi}(x) &= \frac{\gamma}{S_o}\frac{(q-zp)S-\mathbb{E}[q-zp]S}{\sqrt{Var[q-zp]S^2+1}}+\lfloor\frac{\beta}{S_o}\rceil+zp_o \\ &\approx \frac{\gamma}{S_o}\bigotimes\lfloor\frac{q-\mathbb{E}[q]}{\sqrt{\mathbb{E}[(q-\mathbb{E}[q])^2]+1}}\rceil+\lfloor\frac{\beta}{S_o}\rceil+zp_o \\ &= \frac{\gamma}{S_o}\bigotimes\lfloor\frac{q-\mathbb{E}[q]}{\sqrt{\mathbb{E}[q^2]-\mathbb{E}[q]^2+1}}\rceil+\lfloor\frac{\beta}{S_o}\rceil+zp_o\end{aligned} \tag{21}$$

where input scale S and zero point zp are both canceled out in the subtract/division operations. S_o and zp_o are the quantization parameters of the outputs of layer normalization.

The integer-only terms in Equation 21 is denoted as $Q_{ln}=\frac{q-\mathbb{E}[q]}{\sqrt{\mathbb{E}[q^2]-\mathbb{E}[q]^2+1}}$. The multiplication with fractional number $\frac{\gamma}{S_o}$ is implemented in the integer domain similar as softmax. We observe that the numerator is bounded by 17 bits given that q and $E[q]$ are constrained within 16 bits. The small number of bits (only half of 32 bits) makes the left shift dilemma in softmax introduced in Section 3.1.1 not significant in layer normalization. We can left-shift the numerator by $M=32-17=15$ bits, and compensate on the scaling factor, i.e.

$$\mathrm{LNi}(x)=\frac{\gamma/2^M}{S_o}\bigotimes\lfloor\frac{(q-\mathbb{E}[q])<<M}{\sqrt{\mathbb{E}[q^2]-\mathbb{E}[q]^2+1}}\rceil+\lfloor\frac{\beta}{S_o}\rceil+zp_o \tag{22}$$

The output is clipped for uint8 output at last, i.e. $\mathrm{clip}(\mathrm{LNi}(x),0,255)$.

3.2.1 *Channel-wise Layer Normalization Trap*

In (Lin et al., 2022), FQ-ViT approximates floating-point softmax and layer normalization used in partially quantized models, but it achieves even higher accuracy. This counter-intuitive phenomenon drives us to investigate the hidden cause of accuracy drops in partially quantized models.

The major loss of the calculation precision is from the layer normalization operator (as will be justified in Section 4). FQ-ViT uses a channel-wise layer normalization which is different from the tensor-wise normalization used in partial quantization. It seems like channel-wise layer normalization can help solve the accuracy problem of uint8 layer normalization. However, we find that under the layer normalization scenario, *uint8 channel-wise* quantization brings no higher accuracy or efficiency than *uint16 tensor-wise* quantization. The following paragraphs show the justifications.

In channel-wise quantizations, inputs are quantized differently on each element of the last dimension. Suppose that we have D elements in the last dimension, we also have D scale factors $S_{1\le i\le D}$, and D zero points $zp_{1\le i\le D}$. Note that, in Equation 21, there is a reduction operation $\mathbb{E}[q]$ which is conducted on the dimension D. In this case, the integers q_i on different scales need to be mapped onto the same scale when calculating $\mathbb{E}[q]$:

$$\mathbb{E}[q]=\sum_{i=1}^{D}\Phi((q_i-zp_i)S_i) \tag{23}$$

The input passed to layer normalizations changes immediately from different channel-wise scales into a uniform tensor-wise scale. The uniform scale results require a *larger data range*, thus is equivalent to using uint16 or uint32 in the first place. Different from batch normalization, which performs normalizations on different channels separately, channel-wise layer normalization doesn't make much sense, since using tensor-wise layer normalization with higher bit-width is calculation-equivalent, and saves the time of rescaling to the same scales. This is the reason why we use 16-bit integer in our integer layer normalization kernel implemented using Equation 22.

3.3 Integer GELU

For GELU calculation, the error function $\mathrm{erf}()$ is also approximated by a 2nd-degree polynomial as $\exp()$. Different from division truncations in layer normalization and softmax, data truncations in the integer domain are mainly caused by the *multiplication* between input and the error function as shown in Equation 6. In this case, the approximation to GELU can be treated as a 3rd-degree polynomial:

$$\begin{aligned}&\lfloor\frac{b_0}{b_2S^2}\rceil q+\lfloor\frac{b_1}{b_2S}\rceil q^2+q^3 \\ \le&2^{2b}+2^{3b}+2^{3b} \\ \le&2^{3b+1}\times(1+2^{-b-1}) \\ \le&2^{3b+1+\log(1+2^{-b-1})}\end{aligned} \tag{24}$$

As b goes larger, $\log(1+2^{-b-1})\approx 2^{-b-1}$, thus can be ignored here. To fully utilize 31 bits in int32, we can use $b=10$ for the number of bits of requantization introduced in Section 3.1.1. To be able to use a lower number of bits, $b=8$ is also acceptable, and in our experiments, uint8 GELU inputs perform similarly to 10-bit integers.

4 EXPERIMENTS

4.1 Experimental Setup

4.1.1 *Model Quantization*

The tests are conducted on an image classification task on ImageNet (Deng et al., 2009) evaluation dataset. The vision transformer models under tests are ViT (Dosovitskiy et al., 2020), DeiT (Touvron et al., 2021) and Swin Transformer

(abbreviated as Swin) (Liu et al., 2021a). We use different versions of these models, i.e. tiny (T), small (S), base (B), large (L), and annotate them as suffices of the model names.

The calibration dataset uses 100 samples from the ImageNet training dataset. In ISoftmax, uint8 input is applied. The shift parameter m is 12 as deduced in Section 3.1.3, while M for each layer is determined by either output-oriented method or loss-based method as introduced in Section 3.1.2. For ILN, the inputs are quantized to uint16. The shift parameter M is 15 as explained in Section 3.2. In IGELU, inputs are quantized to uint8.

4.1.2 Baselines

We perform the experiments on full-precision models for accuracy analysis. In addition to full-precision baselines, we also include partial quantizations, which perform floating-point arithmetics only on activation functions and normalizations. They are denoted as *Partial FP32* as used in (Liu et al., 2021b; Yuan et al., 2021).

For the full quantization comparisons, we mainly compare to FQ-ViT. However, the original model fails to infer meaningful results under int32 requirements, as shown in Table 1. Thus, we bypass the int32 restrictions on it. Int64 versions of some arithmetics are not supported on the test devices, thus we perform its integer algorithms using floating-point arithmetics. Additionally, FQ-ViT is not a full quantization since the quantization method for GELU is not provided. We use our IGELU kernel to fill this gap. This int64 FQ-ViT with IGELU is denoted as *FQ-ViT+IGELU(int64)*.

There are quantization-parameter-tuning methods (Liu et al., 2021b; Ding et al., 2022; Yuan et al., 2021) (i.e. focusing on finding S and zp) which use floating-point activations and normalizations. They are orthogonal to our range-precision balancing solutions. We include PTQ for ViT (Liu et al., 2021b) for comparisons since it is most relevant to the topic of vision transformers.

4.1.3 Platform

For testing hardware, we use ARM CPUs for runtime profiling on edge devices. Specifically, we use iOS platforms with two Apple chips: A13 and M1. The platform with A13 uses 2 Lightning (big) cores and 4 Thunder (little) cores with a CPU clock rate of up to 2.65GHz, and 4GB memory. M1 is a new generation of Apple chip with 4 performance cores and 4 efficiency cores. The clock rate is up to 3.2GHz. M1 platform is with 8 GB memory.

For the software, we use the Torch-Lite (iOS, 2022) framework. We add integer versions of kernels implemented in Neon instruction sets and add them to ATen library (pyt, 2022) of PyTorch. The vision transformers can be easily tested on edge devices by building iOS apps using Swift which calls torch APIs through Objective-C.

4.1.4 Metrics

We present an evaluation of the quantized models by reporting their top-1 accuracy and latency on the evaluation dataset. Furthermore, we conduct an ablation study for each kernel to examine the impact of data range and precision on the proposed method. We report the truncation ratios for identifying the data truncation vulnerability, as defined in Table 1. To assess the kernel precision using the proposed integer algorithm, we compute the mean squared error (MSE for each kernel by averaging over 1000 samples for its first appearance during DeiT-S inference. Additionally, we investigate the acceleration obtained from the integer version of the kernels by executing them on the two platforms for 5 trials, each consisting of 1000 repetitions, and reporting the average latency in milliseconds. Our methods perform the same modifications on the model weights as partial quantization, thus the memory savings is the same and is not shown due to limited space.

4.2 Top-1 Accuracy

Table 2 showcases the accuracy of vision transformer models upon being subjected to different quantization techniques. Our method, in particular, exhibits a level of accuracy akin to that of full-precision models while experiencing only minimal drops in accuracy amounting to mainly $1 \sim 2\%$. The dearth of open source code for *PTQ for ViT*, leads us to source our results directly from the corresponding paper. In comparison to the results produced by FQ-ViT+IGELU(int64), which employs 64-bit integers, we observe that our int32 method outperforms it. Section 4.4 offers a detailed insight into the sources of the higher accuracy achieved by our method.

We also set partial quantizations as baselines. As explained in Section 3.2.1, the uint8 channel-wise layer normalization which is equivalent to uint16 tensor-wise layer normalization help improve the model performance. We verify this claim by performing the partial quantization on two settings: uint8 and uint16 for layer normalization inputs. The result shows a significant accuracy drop in ViT models and Swin-B as reported in FQ-ViT (Lin et al., 2022). However, after we switch to uint16 inputs, the accuracy is close to full-precision models. This helps to verify the importance of uint16 layer normalization as introduced in Section 3.2.1.

The determination of the ISoftmax shift amount M^* can be accomplished through two methods: the *Output-oriented* and *Loss-based*. The table presented herein reveals that both approaches result in similar accuracy. Specifically, the Output-oriented method enables direct calculation of M^* from calibration, thus negating the need for an optimization process, and is consequently more computation-friendly.

Table 2. Accuracy comparison between baselines and ours. In addition to full-precision fp32, we have partial quantizations with uint8 or uint16 inputs for layer normalizations. PTQ for ViT (Liu et al., 2021b) is a vision-transformer-specific quantization-parameter-tuning method. FQ-ViT (Lin et al., 2022) is implemented with int64 calculations allowed on kernels, and is made fully quantized with the help of our IGELU kernel. Our methods include output-oriented (Output) and loss-based (MSE) for softmax B^* selection.

METHOD	DEIT-T	DEIT-S	DEIT-B	VIT-B	VIT-L	SWIN-T	SWIN-S	SWIN-B
FULL FP32	72.13	79.83	81.80	84.54	85.83	81.38	83.23	83.60
PARTIAL FP32 (UINT16 LN)	71.87	79.51	81.49	83.67	85.46	81.04	83.14	83.50
PARTIAL FP32 (UINT8 LN)	70.93	74.88	77.52	28.51	3.60	64.38	74.37	25.58
PTQ FOR VIT	-	77.47	80.48	-	-	-	-	-
FQ-VIT+IGELU(INT64)	70.42	77.88	80.36	81.20	84.21	79.71	82.14	82.33
OURS (MSE)	**71.08**	**78.49**	**80.74**	81.69	84.47	80.03	**82.29**	**82.67**
OURS (OUTPUT)	71.06	78.47	80.73	**81.81**	**84.84**	**80.07**	82.27	82.65

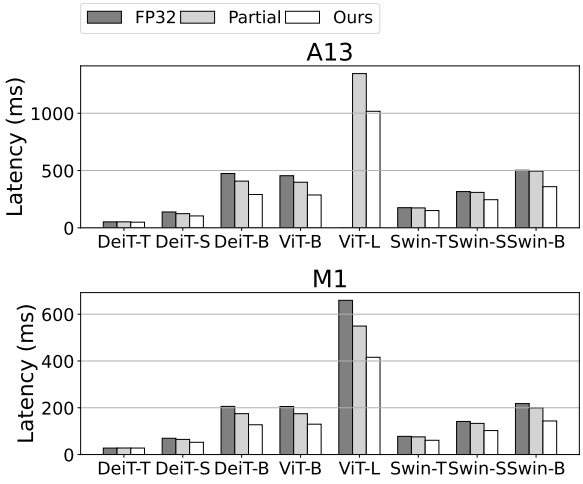


Figure 3. End-to-end latency test on vision transformers: DeiT, ViT and Swin Transformer. FP32 models, partial quantization and our full quantization method are compared. Note that ViT-L exceeds the memory limits of our A13 platform.

4.3 End-to-End Latency

End-to-end latency measurements were performed on vision transformers, and the results are presented in Figure 3. Partial quantization resulted in a latency improvements of up to 16.7%; however, our quantization method realized latency improvements of up to 38.6%, depending on the model size and architecture. For small models, the latency improvements from integer versions of MMM kernels were insignificant compared to the overheads associated with quantization, particularly floating-point and fixed-point conversions in partial quantizations. In contrast, our full quantization methods consistently yielded improvements of over 21%, with the exception of Deit-T and Swin-T, which were tiny models. It should be noted that without quantization methods, resource-constrained edge devices could experience out-of-memory problems when dealing with large models such as ViT-L.

Table 3. Latency comparison on **A13**. Tensor shapes: softmax $(12, 197, 197)$, GELU $(197, 3072)$, layer normalization $(197, 768)$. B means batch size.

OPERATORS	FP32 (MS)	OURS (MS)	SPEEDUP
ISOFTMAX (B=1)	1.90	0.54	3.52×
IGELU (B=1)	6.22	1.42	4.38×
ILN (B=1)	0.50	0.09	5.56×
ISOFTMAX (B=16)	34.77	11.04	3.15×
IGELU (B=16)	111.14	24.17	4.60×
ILN (B=16)	6.95	1.42	4.89×

Table 4. Latency comparison on **M1**. Tensor shapes: softmax (12, 197, 197), GELU (197, 3072), layer normalization (197, 768). B means batch size.

OPERATORS	FP32 (MS)	OURS (MS)	SPEEDUP
ISOFTMAX (B=1)	1.02	0.23	4.43×
IGELU (B=1)	2.71	0.87	3.11×
ILN (B=1)	0.33	0.07	4.71×
ISOFTMAX (B=16)	17.02	3.85	4.42×
IGELU (B=16)	49.18	15.50	3.17×
ILN (B=16)	4.01	0.82	4.89×

4.4 Ablation Study

4.4.1 Kernel Latency

Table 3 and Table 4 display the results of our latency comparison of full-precision kernels and integer kernels on A13 and M1 chips, respectively. We conducted experiments incorporating the DeiT-B model's tensor shapes of the operator inputs. Batch sensitivity was assessed by performing experiments for batch sizes of 1 and 16.

Both integer kernels and floating-point kernels need to perform data conversion between inputs/outputs data types and

Table 5. MSE comparison for softmax. int64 is the integer kernel used in I-Bert (Kim et al., 2021). LIS is the log-scale integer softmax used in FQ-ViT (Lin et al., 2022). Our ISoftmax uses uint8 inputs/outputs and int32 calculations, denoted as (8/32)

OPERATORS	MSE
FP32	0.0
INT64	3.19×10^{-6}
LIS	45.33×10^{-6}
ISOFTMAX(8/32)	4.78×10^{-6}

Table 6. MSE comparison for GELU. int64 is the integer kernel used in I-Bert (Kim et al., 2021). Ours IGELU uses uint8 inputs/outputs and int32 calculations, denoted as (8/32)

OPERATORS	MSE
FP32	0.0
INT64	2.59×10^{-4}
IGELU(8/32)	2.96×10^{-4}

calculation data types. Thus we include the data conversion between uint8 and fp32 for the latency test of floating-point kernels, and data conversion between uin8/uint16 and int32 for integer-only kernels.

From the table, the integer kernels significantly improve the execution latency to a quarter of that of full-precision kernels. Some operators cause a human-perceptible latency as large as ≥ 100 milliseconds. As each operator repeats multiple times in transformer models, it significantly affects user experience for real-time edge device inference. Using full quantization methods helps alleviate this problem.

4.4.2 *Truncation Ratio vs. Precision*

The major motivation of this study entails the predicament of balancing data truncation and precision. Our proposed kernels, as detailed in Section 3, eliminate the prevalent issue of data truncation (i.e., all truncation ratios equaling 0). However, the effects of the requantization process - which involves bit compression - have yet to be evaluated. This section aims to scrutinize the impact of utilizing our suggested quantization techniques on softmax, GELU and layer normalization in terms of calculation *precision*.

Table 5, 6, and 7 show the MSE between the integer version of these kernels and the floating-point kernels. Overall, MSE is extremely small in our proposed kernels. Although the calculations of our kernels are constrained in int32, the MSE is similar to the int64 kernels proposed by I-Bert.

MSE of kernels proposed in FQ-ViT are also reported. In log integer softmax (LIS), an integer number n represents the decimal number $1/2^n$. This gives a higher ulp for close-to-zero numbers. However, the most significant values in softmax outputs are close to 1. The integer approximations to these significant numbers in this log-scale are either represented by 1 or 0.5 which causes larger errors. As shown in the table, MSE of LIS is a magnitude larger than our ISoftmax kernel. Although the error $4.53e-5$ looks small, it is the major cause of the accuracy drop of FQ-ViT as shown in Table 2 compared to our proposed kernels. For the layer normalization, we compare ours with the power-of-two factor (PTF) kernel in FQ-ViT as shown in Table 7. PTF uses uint8 for channel-wise quantization for PTF inputs. However, as discussed in Section 3.2.1, the inputs are immediately converted to tensor-wise quantized values with higher bit width to calculate the mean and standard deviation. Thus, it has a similar algorithm as int64 version of layer normalization in I-Bert. The table shows almost equivalent MSE among int64, PTF and our ILN kernels.

Table 7. MSE comparison for layer normalization. int64 is the integer kernel used in I-Bert (Kim et al., 2021). PTF is used in FQ-ViT (Lin et al., 2022). Ours ILN uses uint16 inputs, uint8 outputs and int32 calculations, denoted as (16/32)

OPERATORS	MSE
FP32	0.0
INT64	3.74×10^{-4}
PTF	3.75×10^{-4}
ILN(16/32)	3.74×10^{-4}

5 CONCLUSION

In this work, we show a widely ignored data truncation problem in fully quantized PTQ of vision transformers deployed on edge devices. We solve the range-precision dilemma in integer domains by carefully requantizing the data range and choosing the bit-width considering both the truncation rate and precision. We design the integer kernels based on the above method for transformer-specific operators: softmax, GELU, and layer normalization. These kernels show the elimination of truncation problems while retaining similar inference accuracy as full-precision models on the image classification task. The integer kernels also significantly improve the execution speed.

ACKNOWLEDGEMENT

This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG2-TC-2021-002), and Centre for Trusted Internet and Community, National University of Singapore. We would like to thank Dr. Yao Chen for his invaluable advice on this work. And we thank Professor Jingtong Hu for shepherding our paper.

REFERENCES

Tensorflow lite. https://www.tensorflow.org/lite, 2022.

ios — pytorch. https://pytorch.org/mobile/ios/, 2022.

Introducing neon development article. https://developer.arm.com/documentation/dht0002/a/Introducing-NEON/NEON-architecture-overview, 2022.

pytorch/aten at master · pytorch/pytorch. https://github.com/pytorch/pytorch/tree/master/aten, 2022.

Learn the architecture - introducing sve2. https://developer.arm.com/documentation/102340/0001/Introducing-SVE2, 2022.

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 265–283, 2016.

arm. Arm. cortex-m. https://developer.arm.com/ip-products/processors/cortex-m, 2022.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Bai, H., Zhang, W., Hou, L., Shang, L., Jin, J., Jiang, X., Liu, Q., Lyu, M., and King, I. Binarybert: Pushing the limit of bert quantization. *arXiv preprint arXiv:2012.15701*, 2020.

Bondarenko, Y., Nagel, M., and Blankevoort, T. Understanding and overcoming the challenges of efficient transformer quantization. *arXiv preprint arXiv:2109.12948*, 2021.

Bridle, J. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. *Advances in neural information processing systems*, 2, 1989.

Capotondi, A., Rusci, M., Fariselli, M., and Benini, L. Cmix-nn: Mixed low-precision cnn library for memory-constrained edge devices. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(5):871–875, 2020.

Chung, I., Kim, B., Choi, Y., Kwon, S. J., Jeon, Y., Park, B., Kim, S., and Lee, D. Extremely low bit transformer quantization for on-device neural machine translation. *arXiv preprint arXiv:2009.07453*, 2020.

Dahl, G. E., Sainath, T. N., and Hinton, G. E. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 8609–8613. IEEE, 2013.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Ding, Y., Qin, H., Yan, Q., Chai, Z., Liu, J., Wei, X., and Liu, X. Towards accurate post-training quantization for vision transformer. In *Proceedings of the 30th ACM International Conference on Multimedia*, pp. 5380–5388, 2022.

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

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 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 *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2704–2713, 2018.

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.

Krishnamoorthi, R. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

Li, Z. and Gu, Q. I-vit: Integer-only quantization for efficient vision transformer inference. *arXiv preprint arXiv:2207.01405*, 2022.

Li, Z., Yang, T., Wang, P., and Cheng, J. Q-vit: Fully differentiable quantization for vision transformer. *arXiv preprint arXiv:2201.07703*, 2022.

Lin, Y., Li, Y., Liu, T., Xiao, T., Liu, T., and Zhu, J. Towards fully 8-bit integer inference for the transformer model. *arXiv preprint arXiv:2009.08034*, 2020.

Lin, Y., Zhang, T., Sun, P., Li, Z., and Zhou, S. Fq-vit: Post-training quantization for fully quantized vision transformer. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 1173–1179, 2022.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021a.

Liu, Z., Wang, Y., Han, K., Zhang, W., Ma, S., and Gao, W. Post-training quantization for vision transformer. *Advances in Neural Information Processing Systems*, 34: 28092–28103, 2021b.

Nikouei, S. Y., Chen, Y., Song, S., Xu, R., Choi, B.-Y., and Faughnan, T. R. Real-time human detection as an edge service enabled by a lightweight cnn. In *2018 IEEE International Conference on Edge Computing (EDGE)*, pp. 125–129. IEEE, 2018.

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. *Advances in neural information processing systems*, 32, 2019.

Tang, X., Han, S., Zhang, L. L., Cao, T., and Liu, Y. To bridge neural network design and real-world performance: A behaviour study for neural networks. *Proceedings of Machine Learning and Systems*, 3:21–37, 2021.

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jégou, H. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pp. 10347–10357. PMLR, 2021.

Tu, Y. and Lin, Y. Deep neural network compression technique towards efficient digital signal modulation recognition in edge device. *IEEE Access*, 7:58113–58119, 2019.

Wu, J., Leng, C., Wang, Y., Hu, Q., and Cheng, J. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4820–4828, 2016.

Yuan, Z., Xue, C., Chen, Y., Wu, Q., and Sun, G. Ptq4vit: Post-training quantization framework for vision transformers. *arXiv preprint arXiv:2111.12293*, 2021.

Zhang, W., Hou, L., Yin, Y., Shang, L., Chen, X., Jiang, X., and Liu, Q. Ternarybert: Distillation-aware ultra-low bit bert. *arXiv preprint arXiv:2009.12812*, 2020.

Zhu, F., Gong, R., Yu, F., Liu, X., Wang, Y., Li, Z., Yang, X., and Yan, J. Towards unified int8 training for convolutional neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1969–1979, 2020.