I-BERT: Integer-only BERT Quantization

Sehoon Kim [*1] **Amir Gholami** [*1] **Zhewei Yao** [*1] **Michael W. Mahoney** [1] **Kurt Keutzer** [1]

Abstract

Transformer based models, like BERT and RoBERTa, have achieved state-of-the-art results in many Natural Language Processing tasks. However, their memory footprint, inference latency, and power consumption are prohibitive for efficient inference at the edge, and even at the data center. While quantization can be a viable solution for this, previous work on quantizing Transformer based models use floating-point arithmetic during inference, which cannot efficiently utilize integer-only logical units such as the recent Turing Tensor Cores, or traditional integer-only ARM processors. In this work, we propose I-BERT, a novel quantization scheme for Transformer based models that quantizes the entire inference with integer-only arithmetic. Based on lightweight integer-only approximation methods for nonlinear operations, e.g., GELU, Softmax, and Layer Normalization, I-BERT performs an end-to-end integer-only BERT inference without any floating point calculation. We evaluate our approach on GLUE downstream tasks using RoBERTa-Base/Large. We show that for both cases, I-BERT achieves similar (and slightly higher) accuracy as compared to the full-precision baseline. Furthermore, our preliminary implementation of I-BERT shows a speedup of $2.4 - 4.0\times$ for INT8 inference on a T4 GPU system as compared to FP32 inference. The framework has been developed in PyTorch and has been open-sourced (Kim, 2021).

1. Introduction

The recent Transformer based Neural Network (NN) models (Vaswani et al., 2017), pre-trained from large unlabeled data (e.g., BERT (Devlin et al., 2018), RoBERTa (Liu et al., 2019), and the GPT family (Brown et al., 2020; Radford et al., 2018; 2019)), have achieved a significant accuracy improvement when fine-tuned on a wide range of Natural Language Processing (NLP) tasks such as sentence classification (Wang et al., 2018) and question answering (Rajpurkar et al., 2016). Despite the state-of-the-art results in various NLP tasks, pre-trained Transformer models are generally orders of magnitude larger than prior models. For example, the BERT-Large model (Devlin et al., 2018) contains 340M parameters. Much larger Transformer models have been introduced in the past few years, with even more parameters (Brown et al., 2020; Lepikhin et al., 2020; Radford et al., 2019; Raffel et al., 2019; Rosset, 2019; Shoeybi et al., 2019; Yang et al., 2019). Efficient deployment of these models has become a major challenge, even in data centers, due to limited resources (energy, memory footprint, and compute) and the need for real-time inference. Obviously, these challenges are greater for edge devices, where the compute and energy resources are more constrained.

One promising method to tackle this challenge is quantization (Dong et al., 2019; Jacob et al., 2018; Krishnamoorthi, 2018; Wu et al., 2018; 2016; Zhang et al., 2018), a procedure which compresses NN models into smaller size by representing parameters and/or activations with low bit precision, e.g., 8-bit integer (INT8) instead of 32-bit floating point (FP32). Quantization reduces memory footprint by storing parameters/activations in low precision. With the recent integer-only quantization methods, one can also benefit from faster inference speed by using low precision integer multiplication and accumulation, instead of floating point arithmetic. However, previous quantization schemes for Transformer based models use simulated quantization (aka fake quantization), where all or part of operations in the inference (e.g., GELU (Hendrycks & Gimpel, 2016), Softmax, and Layer Normalization (Ba et al., 2016)) are carried out with floating point arithmetic (Bhandare et al., 2019; Shen et al., 2020; Zafrir et al., 2019). This approach has multiple drawbacks for deployment in real edge application scenarios. Most importantly, the resulting NN models cannot be deployed on neural accelerators or popular edge processors that do not support floating point arithmetic. For instance, the recent server class of Turing Tensor Cores have added high throughput integer logic that are faster than single/half-precision. Similarly, some of the edge pro-

[*]Equal contribution [1]University of California, Berkeley. Correspondence to: Sehoon Kim <sehoonkim@berkeley.edu>, Amir Gholami <amirgh@berkeley.edu>, Zhewei Yao <zheweiy@berkeley.edu>, Michael W. Mahoney <mahoneymw@berkeley.edu>, Kurt Keutzer <keutzer@berkeley.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

cessor cores in ARM Cortex-M (ARM, 2020) family for embedded systems only contain integer arithmetic units, and they can only support NN deployment with the integer-only kernels (Lai et al., 2018). Moreover, one has to consider that compared to the integer-only inference, the approaches that use floating point arithmetic are inferior in latency and power efficiency. For chip designers wishing to support BERT-like models, adding floating point arithmetic logic occupies larger die area on a chip, as compared to integer arithmetic logic. Thus, the complete removal of floating point arithmetic for inference could have a major impact on designing applications, software, and hardware for efficient inference at the edge (ARM, 2020).

While prior work has shown the feasibility of integer-only inference (Jacob et al., 2018; Yao et al., 2020), these approaches have only focused on models in computer vision with simple CNN layers, Batch Normalization (BatchNorm) (Ioffe & Szegedy, 2015), and ReLU activations. These are all linear or piece-wise linear operators. Due to the non-linear operations used in Transformer architecture, e.g., GELU, Softmax, and Layer Normalization (LayerNorm), these methods cannot be applied to Transformer based models. Unlike ReLU, computing GELU and Softmax with integer-only arithmetic is not straightforward, due to their non-linearity. Furthermore, unlike BatchNorm whose parameters/statistics can be fused into the previous convolutional layer in inference, LayerNorm requires the dynamic computation of the square root of the variance for each input. This cannot be naïvely computed with integer-only arithmetic. Another challenge is that processing GELU, Softmax, and LayerNorm with low precision can result in signifciant accuracy degradation (Bhandare et al., 2019; Zafrir et al., 2019). For these reasons, other quantization methods such as (Bhandare et al., 2019; Shen et al., 2020; Zafrir et al., 2019) keep these operations in FP32 precision.

In this work, we propose I-BERT to address these challenges. I-BERT incorporates a series of novel integer-only quantization scheme for Transformer based models. Specifically, our contributions are:

- We propose new kernels for the efficient and accurate integer-only computation of GELU and Softmax. In particular, we approximate GELU and Softmax with lightweight second-order polynomials, which can be evaluated with integer-only arithmetic. We utilize different techniques to improve the approximation error, and achieve a maximum error of 1.8×10^{-2} for GELU, and 1.9×10^{-3} for Softmax. See § 3.4 and 3.5 for details.
- For LayerNorm, we perform integer-only computation by leveraging a known algorithm for integer calculation of square root (Crandall & Pomerance, 2006). See § 3.6 for details.
- We use these approximations of GELU, Softmax, and LayerNorm to design integer-only quantization for Transformer based models. Specifically, we process Embedding and matrix multiplication (MatMul) with INT8 multiplication and INT32 accumulation. The following non-linear operations (GELU, Softmax, and LayerNorm) are then calculated on the INT32 accumulated result and then requantized back to INT8. We represent all parameters and activations in the entire computational graph with integers, and we never cast them into floating point. See Fig. 1 (right) for a schematic description.
- We apply I-BERT to RoBERTa-Base/Large, and we evaluate their accuracy on the GLUE (Wang et al., 2018) downstream tasks. I-BERT achieves similar results as compared to full-precision baseline. Specifically, I-BERT outperforms the baseline by 0.3 and 0.5 on the GLUE downstream tasks for RoBERTa-Base and RoBERTa-Large, respectively. See Tab. 2 in § 4.1 for details.
- We deploy INT8 BERT models with the integer-only kernels for non-linear operations on a T4 GPU using TensorRT (NVIDIA, 2018). We show that INT8 inference achieves up to $4\times$ speedup as compared to FP32 inference. See Tab. 3 in § 4.2 for details.

2. Related Work

Efficient Neural Network. There are several different approaches to reduce the memory footprint, latency, and power of modern NN architectures. These techniques can be broadly categorized into: (1) pruning (Fan et al., 2019; Gordon et al., 2020; Han et al., 2015; LeCun et al., 1990; Li et al., 2016b; Mao et al., 2017; 2020; Michel et al., 2019; Molchanov et al., 2016; Raganato et al., 2020; Sanh et al., 2020; Yang et al., 2017); (2) knowledge distillation (Hinton et al., 2014; Jiao et al., 2019; Mishra & Marr, 2017; Polino et al., 2018; Romero et al., 2014; Sanh et al., 2019; Sun et al., 2019; 2020; Tang et al., 2019; Turc et al., 2019; Wang et al., 2020; Xu et al., 2020); (3) efficient neural architecture design (Dehghani et al., 2018; Howard et al., 2019; Iandola et al., 2016; Lan et al., 2019; Sandler et al., 2018; Tan & Le, 2019); (4) hardware-aware NN co-design (Gholami et al., 2018; Han & Dally, 2017; Kwon et al., 2018); and (5) quantization.

Here, we only focus on quantization and briefly discuss the related work.

Quantization. For quantization, the parameters and/or activations are represented with low bit precision (Choi et al., 2018; Courbariaux et al., 2015; 2016; Dong et al., 2019; Jacob et al., 2018; Li et al., 2016a; Rastegari et al., 2016; Wang et al., 2019; Wu et al., 2016; Zhang et al., 2018; Zhou et al., 2016). While this line of research mostly focuses on CNN models, there have been recent attempts to introduce quantization techniques into Transformer based models as well. For example, (Bhandare et al., 2019) and (Zafrir et al., 2019) propose an 8-bit quantization scheme for Transformer

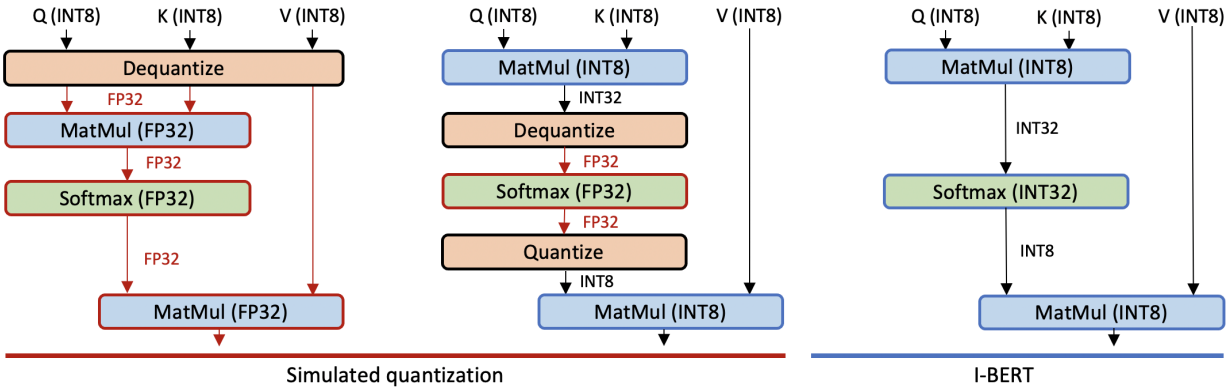

Figure 1. Comparison of different quantization schemes applied to the self-attention layer in the Transformer architecture. (Left) Simulated quantization, where all operations are performed with floating point arithmetic. Parameters are quantized and stored as integer, but they are dequantized into floating point for inference. (Middle) Simulated quantization, where only a part of operations are performed with integer arithmetic. Because the Softmax in this figure is performed with floating point arithmetic, the input to the Softmax should be dequantized; and the output from the Softmax should be quantized back into integer to perform the subsequent integer MatMul. (Right) The integer-only quantization that we propose. There is neither floating point arithmetic nor dequantization during the entire inference.

based models and compress the model size up to 25% of the original size. Another work (Shen et al., 2020) applies uniform and mixed-precision to quantize BERT model, where a second-order sensitivity method is used for the mixed-precision setting. (Fan et al., 2020) quantizes a different subset of weights in each training iteration to make models more robust to quantization. Recently, there have been attempts to quantize BERT with even lower precision. (Zadeh et al., 2020) presents a 3/4-bit centroid-based quantization method that does not require fine-tuning. (Bai et al., 2020; Zhang et al., 2020) leverage knowledge distillation (Hinton et al., 2014) to ternarize/binarize weights. (Jin et al., 2021) combines knowledge distillation and learned step size quantization (Esser et al., 2019) method to achieve up to 2-bit quantization of BERT.

However, to the best of our knowledge, all of the prior quantization work on Transformer based models use *simulated quantization* (aka fake quantization), where all or part of operations are performed with floating point arithmetic. This requires the quantized parameters and/or activations to be dequantized back to FP32 for the floating point operations. For example, (Shen et al., 2020; Zadeh et al., 2020) perform the entire inference using floating point arithmetic, as schematically shown in Fig. 1 (left). While (Bai et al., 2020; Bhandare et al., 2019; Zafrir et al., 2019; Zhang et al., 2020) attempt to process Embedding and MatMul efficiently with integer arithmetic, they keep the remaining operations (i.e., GELU, Softmax, and LayerNorm) in FP32, as illustrated in Fig. 1 (middle). However, our method I-BERT uses integer-only quantization for the entire inference process—i.e., without any floating point arithmetic and without any dequantization during the entire inference. This is illustrated in Fig. 1 (right). This allows more efficient hardware deployment on specialized accelerators or integer-only processors (ARM, 2020) as well as faster and less energy consuming inference. While we focus on uniform quantization, our method is complementary to other mixed and/or low-precision methods, and can be deployed for those settings as well.

To briefly discuss, there are also several quantization works for computer vision. (Jacob et al., 2018) introduces an integer-only quantization scheme for popular CNN models, by replacing all floating point operations (e.g., convolution, MatMul, and ReLU) with integer operations. Similarly, the recent work of (Yao et al., 2020) extends this approach to low precision and mixed precision dyadic quantization, which is an extension of integer-only quantization where no integer division is used. However, both of these works are limited to CNN models that only contain linear and piece-wise linear operators, and they cannot be applied to Transformer based models with non-linear operators, e.g., GELU, Softmax, and LayerNorm. Our work aims to address this limitation by extending the integer-only scheme to the Transformer based models without accuracy drop.

3. Methodology

3.1. Basic Quantization Method

Under *uniform symmetric quantization* scheme, a real number x is uniformly mapped to an integer value $q \in [-2^{b-1}, 2^{b-1} - 1]$, where b specifies the quantization bit precision. The formal definition is:

$$q = \mathrm{Q}(x, b, S) = \mathrm{Int}\left(\frac{\mathrm{clip}(x, -\alpha, \alpha)}{S}\right), \qquad (1)$$

where Q is the quantization operator, Int is the integer map (e.g., round to the nearest integer), clip is the truncation function, α is the clipping parameter used to control the outliers, and S is the scaling factor defined as $\alpha/(2^{b-1} - 1)$.

The reverse mapping from the quantized values q to the real values (aka dequantization) is:

$$\tilde{x} = \mathrm{DQ}(q, S) = Sq \approx x, \tag{2}$$

where DQ denotes the dequantization operator. This approach is referred to as uniform symmetric quantization. It is *uniform* because the spacing between quantized values and their corresponding mapping to real values is constant. However, several different non-uniform quantization methods have also been proposed (Choi et al., 2018; Park et al., 2018; Wu et al., 2016; Zhang et al., 2018). While non-uniform quantization approaches may better capture the distribution of parameters/activations than uniform quantization, they are in general difficult to deploy on hardware (as they often require a look up table which results in overhead). Thus, we focus only on uniform quantization in this work. In addition, this approach is *symmetric* because we clip the values symmetrically within a range $[-\alpha, \alpha]$; while in asymmetric quantization, the left and right side of this range could be asymmetric/different. Finally, we use *static quantization* where all the scaling factors S are fixed during inference to avoid runtime overhead of computing them. See § A for more details in quantization methods.

3.2. Non-linear Functions with Integer-only Arithmetic

The key to integer-only quantization is to perform all operations with integer arithmetic without using any floating point calculation. Unlike linear (e.g., MatMul) or piecewise linear operations (e.g., ReLU), this is not straightforward for non-linear operations (e.g., GELU, Softmax, and LayerNorm). This is because the integer-only quantization algorithms in previous works (Jacob et al., 2018; Yao et al., 2020) rely on the linear property of the operator. For example, $\mathrm{MatMul}(Sq)$ is equivalent to $S \cdot \mathrm{MatMul}(q)$ for the linear MatMul operation. This property allows us to apply integer MatMul to the quantized input q and then multiply the scaling factor S to obtain the same result as applying floating point MatMul to the dequantized input Sq. Importantly, this property *does not* hold for non-linear operations, e.g., $\mathrm{GELU}(Sq) \neq S \cdot \mathrm{GELU}(q)$. One naïve solution is to compute the results of these operations and store them in a look up table (Lai et al., 2018). However, such an approach can have overhead when deployed on chips with limited on-chip memory, and will create a bottleneck proportional to how fast the look up table could be performed. Another solution is to dequantize the activations and convert them to floating point, and then compute these non-linear operations with single precision logic (Bhandare et al., 2019; Zafrir et al., 2019). However, this approach is not integer-only and cannot be used on specialized efficient hardware that does not support floating point arithmetic, e.g., ARM Cortex-M (ARM, 2020).

Algorithm 1 Integer-only Computation of Second-order Polynomial $a(x+b)^2 + c$

Input: q, S: quantized input and scaling factor
Output: q_{out}, S_{out}: quantized output and scaling factor

function I-POLY(q, S) ▷ $qS = x$
  $q_b \leftarrow \lfloor b/S \rfloor$
  $q_c \leftarrow \lfloor c/aS^2 \rfloor$
  $S_{out} \leftarrow \lfloor aS^2 \rfloor$
  $q_{out} \leftarrow (q + q_b)^2 + q_c$
  return q_{out}, S_{out} ▷ $q_{out} S_{out} \approx a(x+b)^2 + c$
end function

To address this challenge, we approximate non-linear activation functions, GELU and Softmax, with polynomials that can be computed with integer-only arithmetic. Computing polynomials consists of only addition and multiplication, which can be performed with integer arithmetic. As such, if we can find good polynomial approximations to these operations, then we can perform the entire inference with integer-only arithmetic. For instance, a second-order polynomial represented as $a(x+b)^2 + c$ can be efficiently calculated with integer-only arithmetic as shown in Alg. 1.[1]

3.3. Polynomial Approximation of Non-linear Functions

There is a large body of work on approximating a function with a polynomial (Stewart, 1996). We use a class of *interpolating polynomials*, where we are given the function value for a set of $n+1$ different data points $\{(x_0, f_0), \ldots, (x_n, f_n)\}$, and we seek to find a polynomial of degree at most n that exactly matches the function value at these points. It is known that there exists a unique polynomial of degree at most n that passes through all the data points (Waring, 1779). We denote this polynomial by L, defined as:

$$L(x) = \sum_{i=0}^{n} f_i l_i(x) \;\text{ where }\; l_i(x) = \prod_{\substack{0 \le j \le n \\ j \neq i}} \frac{x - x_j}{x_i - x_j}. \tag{3}$$

Interestingly for our problem, we have two knobs to change to find the best polynomial approximation. Since we know the actual target function and can query its exact value for any input, we can choose the interpolating point (x_i, f_i) to be any point on the function. The second knob is to choose the degree of the polynomial. While choosing a high-order polynomial results in smaller error (see Appendix B), there are two problems with this. First, high-order polynomials have higher computational and memory overhead. Second, it is challenging to evaluate them with low-precision integer-only arithmetic, as overflow can happen when multiplying integer values. For every multiplication, we need to use dou-

[1] In Alg. 1, $\lfloor \cdot \rfloor$ means the floor function. Note that, q_b, q_c, and S_{out} can be pre-computed under static quantization. That is to say, there is no floating point calculation, e.g., of S/b, in inference.

ble bit-precision to avoid overflow. As such, the challenge is to find a good low-order polynomial that can closely approximate the non-linear functions used in Transformers. This is what we discuss next, for GELU and Softmax, in § 3.4 and 3.5, respectively, where we show that one can get a close approximation by using only a second-order polynomial.

3.4. Integer-only GELU

GELU (Hendrycks & Gimpel, 2016) is a non-linear activation function used in Transformer models, defined as:

$$\text{GELU}(x) := x \cdot \frac{1}{2}\left[1 + \text{erf}(\frac{x}{\sqrt{2}})\right], \quad \text{where } \text{erf}(x) := \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2) dt. \tag{4}$$

Here, erf is the error function. Figure 2 shows the behaviour of the GELU function (shown in red). GELU has a similar behaviour as ReLU (shown in green) in the limit of large positive/negative values, but it behaves differently near zero. Direct evaluation of the integration term in erf is not computationally efficient. For this reason, several different approximations have been proposed for evaluating GELU. For example, (Hendrycks & Gimpel, 2016) suggests using Sigmoid to approximate erf:

$$\text{GELU}(x) \approx x\sigma(1.702x), \tag{5}$$

where $\sigma(\cdot)$ is the Sigmoid function. This approximation, however, is not a viable solution for integer-only quantization, as the Sigmoid itself is another non-linear function which requires floating point arithmetic. One way to address this is to approximate Sigmoid with the so-called hard Sigmoid (h-Sigmoid) proposed by (Howard et al., 2019) (designed in the context of efficient computer vision models) to obtain an integer-only approximation for GELU:

$$\text{h-GELU}(x) := x\frac{\text{ReLU6}(1.702x + 3)}{6} \approx \text{GELU}(x). \tag{6}$$

We refer to this approximation as h-GELU. Although h-GELU can be computed with integer arithmetic, we observed that replacing GELU with h-GELU in Transformers results in a significant accuracy drop. This is due to the large gap between h-GELU and GELU as depicted in Tab. 1.[2] Figure 2 (left) also shows the noticeable gap between those two functions.

A simple way to address the above problem is to use polynomials to approximate GELU, by solving the following optimization problem:

$$\min_{a,b,c} \frac{1}{2}\left\|\text{GELU}(x) - x \cdot \frac{1}{2}\left[1 + \text{L}(\frac{x}{\sqrt{2}})\right]\right\|_2^2, \quad \text{s.t.} \quad L(x) = a(x+b)^2 + c, \tag{7}$$

[2]Later in our ablation study, we show this can lead to accuracy degradation of up to 2.2 percentages, as reported in Tab. 4.

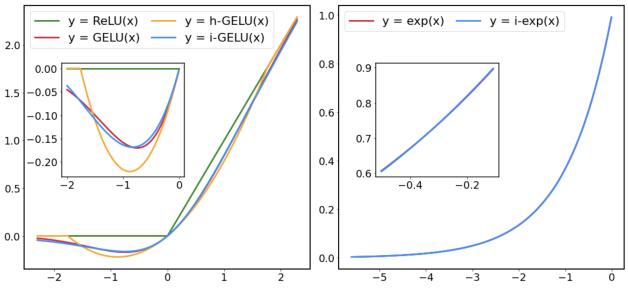

Figure 2. (Left) Comparison between RELU, GELU, h-GELU and i-GELU. (Right) Comparison between exponential (exp) and our integer-only exponential (i-exp).

where $\text{L}(x)$ is a second-order polynomial used to approximate the erf function. Directly optimizing Eq. 7 results in a poor approximation since the definition domain of erf contains the entire real numbers. To address this, we only optimize $L(x)$ in a limited range since erf approaches to 1 (-1) for large values of x. We also take advantage of the fact that erf is an odd function (i.e., $\text{erf}(-x) = -\text{erf}(x)$), and thus only consider approximating it in the positive domain. After finding the best interpolating points, i.e., (x_i, f_i) in Eq. 3, and applying these adjustments we arrive at the following polynomial:

$$\text{L}(x) = \text{sgn}(x)\left[a(\text{clip}(|x|, \text{max} = -b) + b)^2 + 1\right], \tag{8}$$

where $a = -0.2888$ and $b = -1.769$, and sgn denotes the sign function. [3] Using this polynomial we arrive at i-GELU, the integer-only approximation for GELU, defined as:

$$\text{i-GELU}(x) := x \cdot \frac{1}{2}\left[1 + \text{L}(\frac{x}{\sqrt{2}})\right]. \tag{9}$$

Algorithm 2 summarizes the integer-only computation of GELU using i-GELU. We illustrate the behaviour of i-GELU in Fig. 2 (left). As one can see, i-GELU closely approximates GELU, particularly around the origin. We also report the approximation error of i-GELU along with h-GELU in Tab. 1, where i-GELU has an average error of 8.2×10^{-3} and a maximum error of 1.8×10^{-2}. This is $\sim 3\times$ more accurate than h-GELU whose average and maximum errors are 3.1×10^{-2} and 6.8×10^{-2}, respectively. Also, i-GELU even slightly outperforms the Sigmoid based approximation of Eq. 5, but without using any floating point arithmetic. Note that computing the Sigmoid requires floating point. Later in the results section, we show that this improved approximation, actually results in better accuracy of i-GELU as compared to h-GELU (see Tab. 4).

[3]Note that $L(x)$ is approximating GELU in the range of $[0, -b]$.

Algorithm 2 Integer-only GELU

Input: q, S: quantized input and scaling factor
Output: q_{out}, S_{out}: quantized output and scaling factor

function I-ERF(q, S) ▷ $qS = x$
  $a, b, c \leftarrow -0.2888, -1.769, 1$
  $q_{\text{sgn}}, q \leftarrow \text{sgn}(q), \text{clip}(|q|, \max = -b/S)$
  $q_L, S_L \leftarrow$ I-POLY(q, S) with a, b, c ▷ Eq. 8
  $q_{out}, S_{out} \leftarrow q_{\text{sgn}} q_L, S_L$
  return q_{out}, S_{out} ▷ $q_{out} S_{out} \approx \text{erf}(x)$
end function

function I-GELU(q, S) ▷ $qS = x$
  $q_{\text{erf}}, S_{\text{erf}} \leftarrow$ I-ERF$(q, S/\sqrt{2})$
  $q_1 \leftarrow \lfloor 1/S_{\text{erf}} \rfloor$
  $q_{out}, S_{out} \leftarrow q(q_{\text{erf}} + q_1), S S_{\text{erf}}/2$
  return q_{out}, S_{out} ▷ $q_{out} S_{out} \approx \text{GELU}(x)$
end function

Table 1. Comparison of different approximation methods for GELU. The second column (Int-only) indicates whether each approximation method can be computed with integer-only arithmetic. As metrics for approximation error, we report L^2 and L^∞ distance from GELU across the range of [-4, 4].

	Int-only	L^2 dist	L^∞ dist
$x\sigma(1.702x)$	✗	0.012	0.020
h-GELU	✓	0.031	0.068
i-GELU (Ours)	✓	**0.0082**	**0.018**

3.5. Integer-only Softmax

Softmax normalizes an input vector and maps it to a probability distribution:

$$\text{Softmax}(\mathbf{x})_i := \frac{\exp x_i}{\sum_{j=1}^{k} \exp x_j}, \quad \text{where} \quad \mathbf{x} = [x_1, \ldots, x_k]. \tag{10}$$

Approximating the Softmax layer with integer arithmetic is quite challenging, as the exponential function used in Softmax is unbounded and changes rapidly. As such, prior Transformer quantization techniques (Bhandare et al., 2019; Zafrir et al., 2019) treat this layer using floating point arithmetic. Some prior work have proposed look up tables with interpolation (Schraudolph, 1999), but as before we avoid look up tables and strive for a pure arithmetic based approximation. In addition, although (Hauser & Purdy, 2001) proposes polynomial approximation methods for the exponential function, it uses significantly high-degree polynomials, and is only applicable on a limited finite domain.

Similar to GELU, we cannot use a high-order polynomial, but even using such polynomial is ineffective to approximate the exponential function in Softmax. However, it is possible to address problem by limiting the approximation range of Softmax. First, we subtract the maximum value from the input to the exponential for numerical stability:

$$\text{Softmax}(\mathbf{x})_i = \frac{\exp\left(x_i - x_{\max}\right)}{\sum_{j=1}^{k} \exp\left(x_j - x_{\max}\right)}, \tag{11}$$

where $x_{\max} = \max_i(x_i)$. Note that now all the inputs to the exponential function, i.e., $\tilde{x}_i = x_i - x_{\max}$, become non-positive. We can decompose any non-positive real number $\tilde{x}$ as $\tilde{x} = (-\ln 2)z + p$, where the quotient z is a non-negative integer and the remainder p is a real number in $(-\ln 2, 0]$. Then, the exponential of $\tilde{x}$ can be written as:

$$\exp(\tilde{x}) = 2^{-z} \exp(p) = \exp(p) >> z, \tag{12}$$

where $>>$ is the bit shifting operation. As a result, we only need to approximate the exponential function in the compact interval of $p \in (-\ln 2, 0]$. This is a much smaller range as compared to the domain of all real numbers. Interestingly, a variant of this method was used in the Itanium 2 machine from HP (Detrey & de Dinechin, 2005; Thomas et al., 2004), but with a look up table for evaluating $\exp(p)$.

We use a second-order polynomial to approximate the exponential function in this range. To find the coefficients of the polynomial, we minimize the L^2 distance from exponential function in the interval of $(-\ln 2, 0]$. This results in the following approximation:

$$L(p) = 0.3585(p + 1.353)^2 + 0.344 \approx \exp(p). \tag{13}$$

Substituting the exponential term in Eq. 12 with this polynomial results in i-exp:

$$\text{i-exp}(\tilde{x}) := L(p) >> z \tag{14}$$

where $z = \lfloor -\tilde{x}/\ln 2 \rfloor$ and $p = \tilde{x} + z \ln 2$. This can be calculated with integer arithmetic. Algorithm 3 describes the integer-only computation of the Softmax fucntion using i-exp. Figure 2 (right) plots the result of i-exp, which is nearly identical to the exponential function. We find that the largest gap between these two functions is only 1.9×10^{-3}. Considering that 8-bit quantization of a unit interval introduces a quantization error of $1/256 = 3.9 \times 10^{-3}$, our approximation error is relatively negligible and can be subsumed into the quantization error.

3.6. Integer-only LayerNorm

LayerNorm is commonly used in Transformers and involves several non-linear operations, such as division, square, and square root. This operation is used for normalizing the input activation across the channel dimension. The normalization process is described as:

$$\tilde{x} = \frac{x - \mu}{\sigma} \quad \text{where} \quad \mu = \frac{1}{C}\sum_{i=1}^{C} x_i \quad \text{and} \quad \sigma = \sqrt{\frac{1}{C}\sum_{i=1}^{C}(x_i - \mu)^2}. \tag{15}$$

Algorithm 3 Integer-only Exponential and Softmax

Input: q, S: quantized input and scaling factor
Output: q_{out}, S_{out}: quantized output and scaling factor

function I-EXP(q, S) ▷ $qS = x$
  $a, b, c \leftarrow 0.3585, 1.353, 0.344$
  $q_{\ln 2} \leftarrow \lfloor \ln 2 / S \rfloor$
  $z \leftarrow \lfloor -q / q_{\ln 2} \rfloor$
  $q_p \leftarrow q + z q_{\ln 2}$ ▷ $q_p S = p$
  $q_L, S_L \leftarrow$ I-POLY(q_p, S) with a, b, c ▷ Eq. 13
  $q_{out}, S_{out} \leftarrow q_L >> z, S_L$
  return q_{out}, S_{out} ▷ $q_{out} S_{out} \approx \exp(x)$
end function

function I-SOFTMAX(q, S) ▷ $qS = x$
  $\tilde{q} \leftarrow q - \max(q)$
  $q_{\exp}, S_{\exp} \leftarrow$ I-EXP$(\tilde{q}, S)$
  $q_{out}, S_{out} \leftarrow q_{\exp} / \mathrm{sum}(q_{\exp}), S_{\exp}$
  return q_{out}, S_{out} ▷ $q_{out} S_{out} \approx \mathrm{Softmax}(x)$
end function

Algorithm 4 Integer-only Square Root

Input: n: input integer
Output: integer square root of n, i.e., $\lfloor \sqrt{n} \rfloor$

function I-SQRT(n)
  if $n = 0$ **then return** 0
  Intialize x_0 to $2^{\lceil Bits(n)/2 \rceil}$ and i to 0
  repeat
    $x_{i+1} \leftarrow \lfloor (x_i + \lfloor n / x_i \rfloor) / 2 \rfloor$
    if $x_{i+1} \geq x_i$ **then return** x_i
    else $i \leftarrow i + 1$
end function

Here, μ and σ are the mean and standard deviation of the input across the channel dimension. One subtle challenge here is that the input statistics (i.e., μ and σ) change rapidly for NLP tasks, and these values need to be calculated dynamically during runtime. While computing μ is straightforward, evaluating σ requires the square-root function.

The square-root function can be efficiently evaluated with integer-only arithmetic through an iterative algorithm proposed in (Crandall & Pomerance, 2006), as described in Alg. 4. Given any non-negative integer input n, this algorithm iteratively searches for the exact value of $\lfloor \sqrt{n} \rfloor$ based on Newton's Method and only requires integer arithmetic. This algorithm is computationally lightweight, as it converges within at most four iterations for any INT32 inputs and each iteration consists only of one integer division, one integer addition, and one bit-shifting operation. The rest of the the non-linear operations in LayerNorm such as division and square are straightforwardly computed with integer arithmetic.

4. Results

In this section, we first measure the accuracy of I-BERT using the General Language Understanding Evaluation (Wang et al., 2018) (GLUE) benchmark (§ 4.1). Then, we discuss the latency speedup of I-BERT using direct hardware deployment and compare it with pure FP32 model (§ 4.2). Finally, we conduct ablation studies to showcase the effectiveness of our integer-only approximation methods (§ 4.3).

4.1. Accuracy Evaluation on GLUE

We implement I-BERT on the RoBERTa (Liu et al., 2019) model using (Ott et al., 2019). For the integer-only implementation, we replace all the floating point operations in the original model with the corresponding integer-only operations that were discussed in § 3. In particular, we perform MatMul and Embedding with INT8 precision, and the non-linear operations with INT32 precision, as using INT32 for computing these operations has little overhead. See § C.1 for implementation details. For each of the GLUE downstream tasks, we train both FP32 baseline and integer-only I-BERT models, and evaluate the accuracy on the development set. See Appendix C.2 and C.3 for training and evaluation details. While we only test RoBERTa-Base/Large, our method is not restricted to RoBERTa. The integer-only approximations can be performed for any NN models including Transformers that uses similar non-linear operations.

The integer-only quantization results for RoBERTa-Base/Large are presented in Tab. 2. As one can see, I-BERT consistently achieves comparable or slightly higher accuracy than baseline. For RoBERTa-Base, I-BERT achieves higher accuracy for all cases (up to 1.4 for RTE), except for MNLI-m, QQP, and STS-B tasks, where we observe a small accuracy degradation up to 0.3. We observe a similar behaviour on the RoBERTa-Large model, where I-BERT matches or outperforms the baseline accuracy for all the downstream tasks. On average, I-BERT outperforms the baseline by 0.3/0.5 for RoBERTa-Base/Large, respectively.

4.2. Latency Evaluation

We evaluate the latency speedup of INT8 inference of I-BERT, by direct deployment on a Tesla T4 GPU with Turing Tensor Cores that supports accelerated INT8 execution. Although T4 GPU is not a pure integer-only hardware, we select it as our target device due to its extensive software support (Chen et al., 2018; NVIDIA, 2018), and in particular Nvidia's TensorRT library (NVIDIA, 2018). Furthermore, as we do not exploit any T4-specific exclusive features or requirements, our work can be extensively deployed on other hardware as well. See § C.4 for the detailed environment setup. For evaluation, we implement two variants of BERT-Base/Large: (1) pure FP32 models using naïve FP32 kernels for non-linear operations; and (2) quantized INT8 models using customized kernels for the non-linear operations. The customized kernels compute GELU, Softmax, and LayerNorm based on the integer-only methods described in § 3. We measure the inference latency for different sequence

Table 2. Integer-only quantization result for RoBERTa-Base and RoBERTa-Large on the development set of the GLUE benchmark. Baseline is trained by the authors from the pre-trained models, and I-BERT is quantized and fine-tuned from the baseline. We also report the difference (Diff) between the baseline accuracy and the I-BERT accuracy.

(a) RoBERTa-Base

	Precision	Int-only	MNLI-m	MNLI-mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg.
Baseline	FP32	✗	**87.8**	**87.4**	**90.4**	**92.8**	94.6	61.2	**91.1**	90.9	78.0	86.0
I-BERT	INT8	✓	87.5	**87.4**	90.2	**92.8**	**95.2**	**62.5**	90.8	**91.1**	**79.4**	**86.3**
Diff			-0.3	0.0	-0.2	0.0	+0.6	+1.3	-0.3	+0.2	+1.4	+0.3

(b) RoBERTa-Large

	Precision	Int-only	MNLI-m	MNLI-mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg.
Baseline	FP32	✗	90.0	89.9	92.8	94.1	96.3	68.0	**92.2**	91.8	86.3	89.0
I-BERT	INT8	✓	**90.4**	**90.3**	**93.0**	**94.5**	**96.4**	**69.0**	**92.2**	**93.0**	**87.0**	**89.5**
Diff			+0.4	+0.4	+0.2	+0.4	+0.1	+1.0	0.0	+1.2	+0.7	+0.5

Table 3. Inference latency speedup of INT8 inference with respect to FP32 inference for BERT-Base and BERT-Large. Latency is measured for different sentence lengths (SL) and batch sizes (BS).

SL	128				256				Avg.
BS	1	2	4	8	1	2	4	8	
Base	2.42	3.36	3.39	3.31	3.11	2.96	2.94	3.15	3.08
Large	3.20	4.00	3.98	3.81	3.19	3.51	3.37	3.40	3.56

lengths (128 and 256) and batch sizes (1, 2, 4, and 8).

Table 3 shows the inference latency speedup of INT8 models with respect to FP32 models. As one can see, the INT8 inference of I-BERT is on average 3.08× and 3.56× faster than pure FP32 inference for BERT-Base and BERT-Large, respectively, achieving up to 4.00× speedup. The result implies that, when deployed on specialized hardware that supports efficient integer computations, I-BERT can achieve significant speedup as compared to FP32 models. Further speedups are possible with NVIDIA's custom Transformer plugins (Mukherjee et al., 2019) which fuse the multi-head attention and Softmax layers (see § C.4).

While the greatest value of our work will become evident when our approach enables quantization on lower-end microprocessors without floating-point hardware, this demonstration must wait for improved software support for implementing quantized NN models on those processors. In the meantime, we believe the promise of our approach is illustrated by these latency reductions shown above.

4.3. Ablation Studies

Here, we perform an ablation study to show the benefit of i-GELU as compared to other approximation methods for GELU, and in particular h-GELU in Eq. 6. For comparison, we implement two variants of I-BERT by replacing i-GELU with GELU and h-GELU, respectively. The former is the exact computation of GELU with floating point arithmetic, and the later is another integer-only approximation method for GELU (see § 3). We use RoBERTa-Large model as baseline along with the QNLI, SST-2, MPRC, and RTE tasks. All models are trained and fine-tuned according to the procedure described in § 4.1, and the final accuracies are reported in Tab. 4.

Table 4. Accuracy of models that use GELU, h-GELU and i-GELU for GELU computation. Note that the former is full-precision, floating point computation while the latter two are integer-only approximations.

	Int-only	QNLI	SST-2	MRPC	RTE	Avg.
GELU	✗	94.4	96.3	92.6	85.9	92.3
h-GELU	✓	94.3	96.0	92.8	84.8	92.0
i-GELU	✓	**94.5**	**96.4**	**93.0**	**87.0**	**92.7**

As one can see, replacing GELU with h-GELU approximation results in accuracy degradation for all downstream tasks except for MRPC. Accuracy drops by 0.5 on average and up to 1.1 for RTE task. Although accuracy slightly improves for MRPC, the amount of increase is smaller than replacing GELU with i-GELU. This empirically demonstrates that h-GELU is not sufficiently tight enough to approximate GELU well. Approximating GELU with i-GELU results in strictly better accuracy for all four downstream tasks than h-GELU. In particular, i-GELU outperforms h-GELU by 0.7 on average, and it achieves comparable or slightly better result to the non-approximated full-precision GELU. i-GELU also performs better than GELU, which is quite interesting, but at this time, we do not have an explanation for this behaviour.

5. Conclusions

We have proposed I-BERT, a novel integer-only quantization scheme for Transformers, where the entire inference is performed with pure integer arithmetic. Key elements of I-BERT are approximation methods for nonlinear operations such as GELU, Softmax, and LayerNorm, which enable their approximation with integer computation. We empirically evaluated I-BERT on RoBERTa-Base/Large models, where our quantization method improves the average GLUE score by 0.3/0.5 points as comapred to baseline. Furthermore, we directly deployed the quantized models and measured the end-to-end inference latency, showing that I-BERT can achieve up to 4.00× speedup on a Tesla T4 GPU as compared to floating point baseline. As part of future work, one could consider using our approximation to improve the training speed as well. For instance, one could consider replacing GELU with i-GELU during training. Also, further studies are needed to evaluate the performance benefit of i-GELU as compared to GELU.

Acknowledgments

The UC Berkeley team acknowledges gracious support from Intel corporation, Intel VLAB team, Google Cloud, Google TRC team, and Nvidia, as well as valuable feedback from Prof. Dave Patterson, and Prof. Joseph Gonzalez. Amir Gholami was supported through a gracious fund from Samsung SAIT. Michael W. Mahoney would also like to acknowledge the UC Berkeley CLTC, ARO, NSF, and ONR. Our conclusions do not necessarily reflect the position or the policy of our sponsors, and no official endorsement should be inferred.

References

ARM. Cortex-M, https://developer.arm.com/ip-products/processors/cortex-m, 2020.

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.

Bhandare, A., Sripathi, V., Karkada, D., Menon, V., Choi, S., Datta, K., and Saletore, V. Efficient 8-bit quantization of transformer neural machine language translation model. *arXiv preprint arXiv:1906.00532*, 2019.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I., and Specia, L. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*, 2017.

Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L., et al. TVM: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 578–594, 2018.

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

Courbariaux, M., Bengio, Y., and David, J.-P. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pp. 3123–3131, 2015.

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

Crandall, R. and Pomerance, C. B. *Prime numbers: a computational perspective*, volume 182. Springer Science & Business Media, 2006.

Dagan, I., Glickman, O., and Magnini, B. The pascal recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, pp. 177–190. Springer, 2005.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, Ł. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Detrey, J. and de Dinechin, F. A parameterized floating-point exponential function for fpgas. In *Proceedings. 2005 IEEE International Conference on Field-Programmable Technology, 2005.*, pp. 27–34. IEEE, 2005.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dodge, J., Ilharco, G., Schwartz, R., Farhadi, A., Hajishirzi, H., and Smith, N. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*, 2020.

Dolan, W. B. and Brockett, C. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

Dong, Z., Yao, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. HAWQ: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 293–302, 2019.

Esser, S. K., McKinstry, J. L., Bablani, D., Appuswamy, R., and Modha, D. S. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.

Fan, A., Grave, E., and Joulin, A. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*, 2019.

Fan, A., Stock, P., Graham, B., Grave, E., Gribonval, R., Jegou, H., and Joulin, A. Training with quantization noise for extreme fixed-point compression. *arXiv preprint arXiv:2004.07320*, 2020.

Gholami, A., Kwon, K., Wu, B., Tai, Z., Yue, X., Jin, P., Zhao, S., and Keutzer, K. SqueezeNext: Hardware-aware neural network design. *Workshop paper in CVPR*, 2018.

Gordon, M. A., Duh, K., and Andrews, N. Compressing bert: Studying the effects of weight pruning on transfer learning. *arXiv preprint arXiv:2002.08307*, 2020.

Han, S. and Dally, B. Efficient methods and hardware for deep learning. *University Lecture*, 2017.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015.

Hauser, J. W. and Purdy, C. N. Approximating functions for embedded and asic applications. In *Proceedings of the 44th IEEE 2001 Midwest Symposium on Circuits and Systems. MWSCAS 2001 (Cat. No. 01CH37257)*, volume 1, pp. 478–481. IEEE, 2001.

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

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *Workshop paper in NIPS*, 2014.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al. Searching for MobilenetV3. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1314–1324, 2019.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Iyer, S., Dandekar, N., and Csernai, K. First quora dataset release: Question pairs.(2017). *URL https://data. quora. com/First-Quora-Dataset-Release-Question-Pairs*, 2017.

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.

Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., and Liu, Q. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

Jin, J., Liang, C., Wu, T., Zou, L., and Gan, Z. Kdlsq-bert: A quantized bert combining knowledge distillation with learned step size quantization. *arXiv preprint arXiv:2101.05938*, 2021.

Kim, S. https://github.com/kssteven418/i-bert, 2021.

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

Kwon, K., Amid, A., Gholami, A., Wu, B., Asanovic, K., and Keutzer, K. Co-design of deep neural nets and neural net accelerators for embedded vision applications. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2018.

Lai, L., Suda, N., and Chandra, V. CMSIS-NN: Efficient neural network kernels for arm cortex-m cpus. *arXiv preprint arXiv:1801.06601*, 2018.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. GShard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Levesque, H., Davis, E., and Morgenstern, L. The winograd schema challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*. Citeseer, 2012.

Li, F., Zhang, B., and Liu, B. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016a.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016b.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. RoBERTa: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Mao, H., Han, S., Pool, J., Li, W., Liu, X., Wang, Y., and Dally, W. J. Exploring the regularity of sparse structure in convolutional neural networks. *Workshop paper in CVPR*, 2017.

Mao, Y., Wang, Y., Wu, C., Zhang, C., Wang, Y., Yang, Y., Zhang, Q., Tong, Y., and Bai, J. Ladabert: Lightweight adaptation of bert through hybrid model compression. *arXiv preprint arXiv:2004.04124*, 2020.

Michel, P., Levy, O., and Neubig, G. Are sixteen heads really better than one? *arXiv preprint arXiv:1905.10650*, 2019.

Mishra, A. and Marr, D. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *arXiv preprint arXiv:1711.05852*, 2017.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

Mukherjee, P., Weill, E., Taneja, R., Onofrio, D., Ko, Y.-J., and Sharma, S. Real-time natural language understanding with bert using tensorrt, hhttps://developer.nvidia.com/blog/nlu-with-tensorrt-bert/, 2019.

NVIDIA. TensorRT: https://developer.nvidia.com/tensorrt, 2018.

Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., and Auli, M. FairSeq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

Park, E., Yoo, S., and Vajda, P. Value-aware quantization for training and inference of neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 580–595, 2018.

Polino, A., Pascanu, R., and Alistarh, D. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. Improving language understanding by generative pre-training, 2018.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Raganato, A., Scherrer, Y., and Tiedemann, J. Fixed encoder self-attention patterns in transformer-based machine translation. *arXiv preprint arXiv:2002.10260*, 2020.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. SQuAD: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. XNOR-Net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pp. 525–542. Springer, 2016.

Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Gatta, C., and Bengio, Y. FitNets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

Rosset, C. Turing-NLG: A 17-billion-parameter language model by microsoft. *Microsoft Blog*, 2019.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. MobilenetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Sanh, V., Wolf, T., and Rush, A. M. Movement pruning: Adaptive sparsity by fine-tuning. *arXiv preprint arXiv:2005.07683*, 2020.

Schraudolph, N. N. A fast, compact approximation of the exponential function. *Neural Computation*, 11(4):853–862, 1999.

Shen, S., Dong, Z., Ye, J., Ma, L., Yao, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. Q-BERT: Hessian based ultra low precision quantization of bert. In *AAAI*, pp. 8815–8821, 2020.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-LM: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Stewart, G. W. *Afternotes on numerical analysis*. SIAM, 1996.

Sun, S., Cheng, Y., Gan, Z., and Liu, J. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355*, 2019.

Sun, Z., Yu, H., Song, X., Liu, R., Yang, Y., and Zhou, D. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984*, 2020.

Tan, M. and Le, Q. V. EfficientNet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

Tang, R., Lu, Y., Liu, L., Mou, L., Vechtomova, O., and Lin, J. Distilling task-specific knowledge from bert into simple neural networks. *arXiv preprint arXiv:1903.12136*, 2019.

Thomas, J. W., Okada, J. P., Markstein, P., and Li, R.-C. The libm library and floatingpoint arithmetic in hp-ux for itanium-based systems. Technical report, Technical report, Hewlett-Packard Company, Palo Alto, CA, USA, 2004.

Turc, I., Chang, M.-W., Lee, K., and Toutanova, K. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962*, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Wang, K., Liu, Z., Lin, Y., Lin, J., and Han, S. HAQ: Hardware-aware automated quantization. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019.

Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., and Zhou, M. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *arXiv preprint arXiv:2002.10957*, 2020.

Waring, E. Vii. problems concerning interpolations. *Philosophical transactions of the royal society of London*, 1779.

Warstadt, A., Singh, A., and Bowman, S. R. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019.

Williams, A., Nangia, N., and Bowman, S. R. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

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

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.

Xu, C., Zhou, W., Ge, T., Wei, F., and Zhou, M. Bert-of-theseus: Compressing bert by progressive module replacing. *arXiv preprint arXiv:2002.02925*, 2020.

Yang, T.-J., Chen, Y.-H., and Sze, V. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5687–5695, 2017.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. XLNet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pp. 5753–5763, 2019.

Yao, Z., Dong, Z., Zheng, Z., Gholami, A., Yu, J., Tan, E., Wang, L., Huang, Q., Wang, Y., Mahoney, M. W., and Keutzer, K. HAWQV3: Dyadic neural network quantization. *arXiv preprint arXiv:2011.10680*, 2020.

Zadeh, A. H., Edo, I., Awad, O. M., and Moshovos, A. Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 811–824. IEEE, 2020.

Zafrir, O., Boudoukh, G., Izsak, P., and Wasserblat, M. Q8BERT: Quantized 8bit bert. *arXiv preprint arXiv:1910.06188*, 2019.

Zhang, D., Yang, J., Ye, D., and Hua, G. LQ-Nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 365–382, 2018.

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.

Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., and Zou, Y. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

A. Quantization Methods

A.1. Symmetric and Asymmetric Quantization

Symmetric and asymmetric quantization are two different methods for uniform quantization. Uniform quantization is a uniform mapping from floating point $x \in [x_{\min}, x_{\max}]$ to b-bit integer $q \in [-2^{b-1}, 2^{b-1} - 1]$. Before the mapping, input x that does not fall into the range of $[x_{\min}, x_{\max}]$ should be clipped. In asymmetric quantization, the left and the right side of the clipping range can be different, i.e., $-x_{\min} \neq x_{\max}$. However, this results in a bias term that needs to be considered when performing multiplication or convolution operations (Jacob et al., 2018). For this reason, we only use symmetric quantization in this work. In symmetric quantization, the left and the right side of the clipping range must be equal, i.e., $-x_{\min} = x_{\max} = \alpha$, and the mapping can be represented as Eq. 1.

A.2. Static and Dynamic Quantization

There is a subtle but important factor to consider when computing the scaling factor, S. Computing this scaling factor requires determining the range of parameters/activations (i.e., α parameter in Eq. 1). Since the model parameters are fixed during inference, their range and the corresponding scaling factor can be precomputed. However, activations vary across different inputs, and thus their range varies. One way to address this issue is to use dynamic quantization, where the activation range and the scaling factor are calculated during inference. However, computing the range of activation is costly as it requires a scan over the entire data and often results in significant overhead. Static quantization avoids this runtime computation by precomputing a fixed range based on the statistics of activations during training, and then uses that fixed range during inference. As such, it does not have the runtime overhead of computing the range of activations. For maximum efficiency, we adopt static quantization, with all the scaling factors fixed during inference.

B. Error Term of Eq. 3

As one can see, the polynomial approximation of Eq. 3 exactly matches the data at the interpolating points (x_j, f_j). The error between a target function $f(x)$ and the polynomial approximation $L(x)$ is then:

$$|f(x) - L(x)| = \left| \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)\dots(x - x_n) \right|, \quad (16)$$

where ξ is some number that lies in the smallest interval containing $x_0, ..., x_n$. In general, this error reduces for large n (for a properly selected set of interpolating points). Therefore, a sufficiently high-order polynomial that interpolates a target function is guaranteed to be a good approximation for it. We refer interested readers to (Stewart, 1996) for more details on polynomial interpolation.

C. Experimental Details

C.1. Implementation

In I-BERT, all the MatMul operations are performed with INT8 precision, and are accumulated to INT32 precision. Furthermore, the Embedding layer is kept at INT8 precision. Moreover, the non-linear operations (i.e., GELU, Softmax, and LayerNorm) are processed with INT32 precision, as we found that keeping them at high precision is important to ensure no accuracy degradation after quantization. Importantly, note that using INT32 for computing these operations has little overhead, as input data is already accumulated with INT32 precision, and these non-linear operations have linear computational complexity. We perform Requantization (Yao et al., 2020) operation after these operations to bring the precision down from INT32 back to INT8 so that the follow up operations (e.g., next MatMuls) can be performed with low precision.

C.2. Training

We evaluate I-BERT on the GLUE benchmark (Wang et al., 2018), which is a set of 9 natural language understanding tasks, including sentimental analysis, entailment, and question answering. We first train the pre-trained RoBERTa model on the different GLUE downstream tasks until the model achieves the best result on the development set. We report this as the baseline accuracy. We then quantize the model and perform quantization-aware fine-tuning to recover the accuracy degradation caused by quantization. We refer the readers to (Yao et al., 2020) for more details about the quantization-aware fine-tuning method for integer-only quantization. We search the optimal hyperparameters in a search space of learning rate $\{5e{-}7, 1e{-}6, 1.5e{-}6, 2e{-}6\}$, self-attention layer dropout $\{0.0, 0.1\}$, and fully-connected layer dropout $\{0.1, 0.2\}$, except for the one after GELU activation that is fixed to 0.0. We fine-tune up to 6 epochs for larger datasets (e.g., MNLI and QQP), and 12 epochs for the smaller datasets. We report the best accuracy of the resulting quantized model on the development set as I-BERT accuracy.

C.3. Accuracy Evaluation on the GLUE Tasks

For evaluating the results, we use the standard metrics for each task in GLUE. In particular, we use classification accuracy and F1 score for QQP (Iyer et al., 2017) and MRPC (Dolan & Brockett, 2005), Pearson Correlation and Spearman Correlation for STS-B (Cer et al., 2017), and Mathews Correlation Coefficient for CoLA (Warstadt et al., 2019). For the remaining tasks (Dagan et al., 2005; Ra-

jpurkar et al., 2016; Socher et al., 2013; Williams et al., 2017), we use classification accuracy. For the tasks with multiple metrics, we report the average of them. Since there are two development sets for MNLI (Williams et al., 2017), i.e., MNLI-match (MNLI-m) for in-domain evaluation, and MNLI-mismatch (MNLI-mm) for cross-domain evaluation, and we report the accuracy on both datasets. We exclude WNLI (Levesque et al., 2012) as it has relatively small dataset and shows an unstable behaviour (Dodge et al., 2020).

C.4. Environment Setup for Latency Evaluation

We use TensorRT 7.2.1 to deploy and tune the latency of BERT-Base and BERT-Large models (both INT8 and FP32) on Google Cloud Platform virtual machine with a single Tesla T4 GPU, CUDA 11.1, and cuDNN 8.0.

We should also mention that the most efficient way of implementing BERT with TensorRT is to use NVIDIA's plugins (Mukherjee et al., 2019) that optimize and accelerate key operations in the Transformer architecture via operation fusion. Our estimates are that INT8 inference using NVIDIA's plugins is about 2 times faster than naïvely using TensorRT APIs. However, we cannot modify those plugins to support our integer-only kernels as they are partially closed sourced and pre-compiled. Therefore, our latency evaluation is conducted without fully utilizing NVIDIA's plugins. This leaves us a chance for further optimization to achieve our latency speedup relative to FP32 even more significant. As such, one could expect the potential for a further $\sim 2\times$ speed up with INT8 quantization.