I-BERT: Integer-only BERT Quantization

Sehoon Kim*, Amir Gholami*, Zhewei Yao*, Michael W. Mahoney, Kurt Keutzer
[1]University of California, Berkeley
{sehoonkim, amirgh, zheweiy, mahoneymw, keutzer}@berkeley.edu,

***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 many edge processors, and it has been a challenge to deploy these models for edge applications and devices that have resource constraints. While quantization can be a viable solution to this, previous work on quantizing Transformer based models uses floating-point arithmetic during inference, thus limiting model deployment on many edge processors. In this work, we propose a novel integer-only quantization scheme for Transformer based models that quantizes the entire inference process. In particular, we demonstrate how to approximate nonlinear operations in Transformer architectures, e.g., GELU, Softmax, and Layer Normalization, with lightweight integer computations. We use those approximations in our method, I-BERT, with an end-to-end integer-only inference, and without any floating point calculation. We test our approach on GLUE downstream tasks using RoBERTa-Base and RoBERTa-Large. For both cases, with an 8-bit integer-only quantization scheme, I-BERT achieves similar accuracy as compared to the full-precision baseline.**

I. Introduction

Transformer based language models [58] pre-trained from large unlabeled data (e.g., BERT [11], RoBERTa [36], and the GPT family [3, 44, 45]) have achieved a significant accuracy improvement when fine-tuned on a wide range of Natural Language Processing (NLP) tasks such as sentence classification [59] and question answering [47]. Despite state-of-the-art results in various NLP tasks, pre-trained Transformer based models are generally extremely large. For example, the BERT-Large model [11] contains 340M parameters. Much larger transformer models have been introduced in the past few years, with orders of magnitude more parameters [3, 32, 45, 46, 50, 53, 68]. Efficient deployment of those models has become a major challenge, even in the data centers, let alone at the edge, due to limited resources (energy, memory footprint, and compute) and the need for real-time inference.

One promising method to tackle this challenge is quantization [15, 27, 28, 65, 66, 72], a procedure which compresses 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 (a.k.a. fake quantization), where all or part of operations in the inference (e.g., GELU, Softmax, and Layer Normalization [1]) are carried out with floating point arithmetic [52, 71]. This approach has multiple drawbacks for deployment in real edge application scenarios. Most importantly, the resulting models cannot be deployed on neural accelerators or popular edge processors that do not support floating point arithmetic [6, 30]. Moreover, compared to the integer-only inference, these 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 [6, 30].

While prior work has shown the feasibility of integer-only inference [27, 69], these approaches have only focused on models in computer vision with simple CNN layers, Batch Normalization (BatchNorm) [25], and ReLU activations, which are all linear or piece-wise linear operators. Due to the non-linear operations used in Transformer architecture, e.g., GELU [20], Softmax, and Layer Normalization (LayerNorm) [1], these methods cannot be applied to Transformer based models. Unlike

*Equal contribution.

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 inference input, which cannot be naively computed with integer-only arithmetic. Importantly, it is known that GELU, Softmax, and LayerNorm require higher precision and result in accuracy drop when processed in low precision [2, 71]. For example, Q8BERT [71] keeps these operations in FP32 precision.

In this work, we propose I-BERT to address the above 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 efficient and accurate integer-only computation of GELU and Softmax. In particular, we approximate GELU and Softmax with polynomials that closely approximate these functions, and which can be evaluated with integer-only arithmetic. In particular, our 2nd-order polynomial approximation has a maximum error of 1.8×10^{-2} for GELU, and 1.9×10^{-3} for Softmax. See Section III-D and III-E for details.
- For LayerNorm, we perform integer-only computation by leveraging an algorithm for integer calculation of square root [9]. See Section III-F for details.
- We use these approximations of GELU, Softmax, and LayerNorm, to design integer-only quantization inference for Transformer based architectures. Specifically, we process Embedding and matrix multiplication (MatMul) with INT8 calculation, along with GELU, Softmax, and LayerNorm with 32-bit integer (INT32) calculation. We represent all parameters and activations with integers, and we never cast them into floating point. See Figure 1 (right) for a schematic description.
- We apply our integer-only quantization method, I-BERT, to RoBERTa-Base and RoBERTa-Large, and we evaluate their accuracy on the GLUE [59] downstream tasks. Our results show that I-BERT achieves similar results as compared to full-precision baseline. Specifically, I-BERT outperform the baseline by 0.3 and 0.5 on the GLUE downstream tasks for RoBERTa-Base and RoBERTa-Large, respectively. See Table II in Section IV for details.

II. Related Work

a) Efficient Neural Network: Several different approaches to reduce the memory footprint, latency, and power of modern neural network (NN) architectures. These techniques can be broadly categorized into (1) model pruning [18, 31, 35, 38, 40, 67], (2) knowledge distillation [21, 39, 43, 49, 70], (3) efficient neural architecture design [23, 24, 37, 51, 57], (4) hardware and neural architecture co-design [16, 17, 22, 29, 64], and (5) quantization [5, 7, 8, 14, 15, 27, 34, 48, 60, 66, 72, 73].

b) Quantization: For quantization, the parameters and/or activations are represented with low bit precision. 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, [2] and [71] propose an 8-bit quantization scheme for Transformer based models and compress the model size up to 25% of the original size. Another work [52] applies mixed-precision quantization on BERT where different precisions are assigned to different layers according to their sensitivity defined by Hessian information. However, to the best of our knowledge, all of the prior quantization work on Transformer based models uses simulated quantization (a.k.a. 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, [52] performs the entire inference using floating point arithmetic, as schematically shown in Figure 1 (left). While [2] and [71] attempt to process Embedding and MatMul efficiently with integer arithmetic, they keep the remaining operations (i.e., GELU, Softmax, and LayerNorm) in floating point, as illustrated in Figure 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 Figure 1 (right). This allows more efficient hardware design and better model deployment on specialized hardware as well as faster and less power consuming inference [6, 30].

In the field of computer vision, [27] introduces an integer-only quantization scheme for CNN models, including ResNet [19], Inception-v3 [56], and MobileNets [51], by replacing all floating point operations (e.g., convolution, MatMul, and ReLU) with integer operations. Similarly, the recent work of [69] extends this approach to low precision and mixed precision integer-only quantization. However, both of these works are limited to CNN models

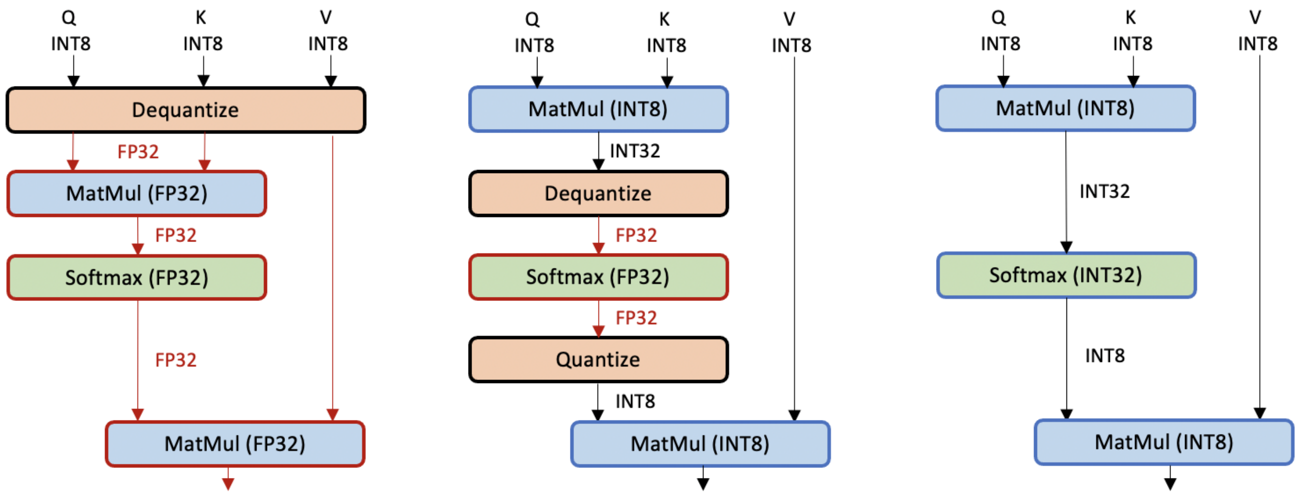

Fig. 1: *Comparison of different quantization schemes applied to the self-attention layer in the Transformer architecture. (Left) Simulated quantization, where all operations perform floating point arithmetic. Parameters are quantized and stored as integer, but they are dequantized into floating point for inference. (Middle) Simulated quantization, where only a part of operations perform 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.*

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 and Softmax. Our work aims to address this limitation by extending the integer-only scheme to the Transformer based models without accuracy drop.

III. Methodology

A. *Basic Quantization Method*

We use *uniform symmetric quantization* with static scaling as described in [28, 69]. In this approach, a real number r is uniformly mapped to an integer value $q \in [-2^{b-1}+1, 2^{b-1}-1]$, where b specifies the quantization bit precision. The formal definition is as follows:

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

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 used to control the outlier numbers, and S is the scaling factor defined as:

$$S = \frac{\alpha}{2^{b-1}-1}. \quad (2)$$

The reverse mapping from the quantized values q to the real value (a.k.a. dequantization) is as follows:

$$\tilde{r} = \mathrm{DQ}(q, S) = Sq \approx r, \quad (3)$$

where DQ denotes the dequantization operator.

This approach is referred to as uniform quantization, since 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 [5, 42, 66, 72]. 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.

Another differentiator is symmetric versus asymmetric quantization. In symmetric quantization, we choose a value α to clip the values in the range of $[-\alpha, \alpha]$. However, in asymmetric quantization, the left and right side of this range could be different values. This results in a bias term that needs to be considered when performing the multiplication or convolution operations. We only use symmetric quantization in this work.

Finally, 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. 2). Since 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 dynamc quantization, where the activation range and the scaling factor are calculated during inference/runtime. However, this approach leads to a 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 testing. 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. *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 piece-wise 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 [27, 69] rely on the linearity property that preserves scalar multiplication. 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. For example, $\mathrm{GELU}(Sq)$ is not equivalent to $S \cdot \mathrm{GELU}(q)$. One naive solution is to compute the results of these operations and store them in a look up table. However, such an approach can have overhead when deployed on chips with limited on-chip memory. Another solution is to dequantize the activations and convert them to floating point, and then compute these non-linear operations with single precision logic [2, 71]. However, this approach is not integer-only and cannot be used on specialized efficient hardware that does not support floating point arithmetic.

To address this challenge, we approximate the non-linear operations 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 approximation to these operations then we can perform the entire inference with integer-only arithmetic. For instance, a 2nd-order polynomial (represented as $a(x+b)^2+c$) can be efficiently calculated with integer-only arithmetic as shown in Algorithm 1.[1]

Algorithm 1 Integer-only computation of 2nd-order polynomial

Input: q, S: quantized input and scaling factor
Output: q_{out}, S_{out}: quantized output and scaling factor for the result of polynomial $a(x+b)^2+c$

```
function I-POLY(q, S)
    q_b ← ⌊b/S⌋
    q_c ← ⌊c/aS^2⌋
    S_out ← ⌊aS^2⌋
    q_out ← (q + q_b)^2 + q_c
    return q_out, S_out
end function
```

C. *Polynomial Approximation of Non-linear Functions*

There is a large body of work on approximating a function with a polynomial [55]. 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 (thus the name interpolating polynomial). It is known that there exists a unique polynomial of degree at most n that passes through all the data points [61]. We denote this polynomial as L, defined as follows:

$$L(x) = \sum_{i=0}^{n} f_i l_i(x) \ \text{ where } \ l_i(x) = \prod_{\substack{0 \le j \le n \\ j \ne i}} \frac{x - x_j}{x_i - x_j}. \quad (4)$$

As one can see, the polynomial approximation 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:

$$|f(x) - L(x)| = \left| \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0) \ldots (x - x_n) \right|, \quad (5)$$

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

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

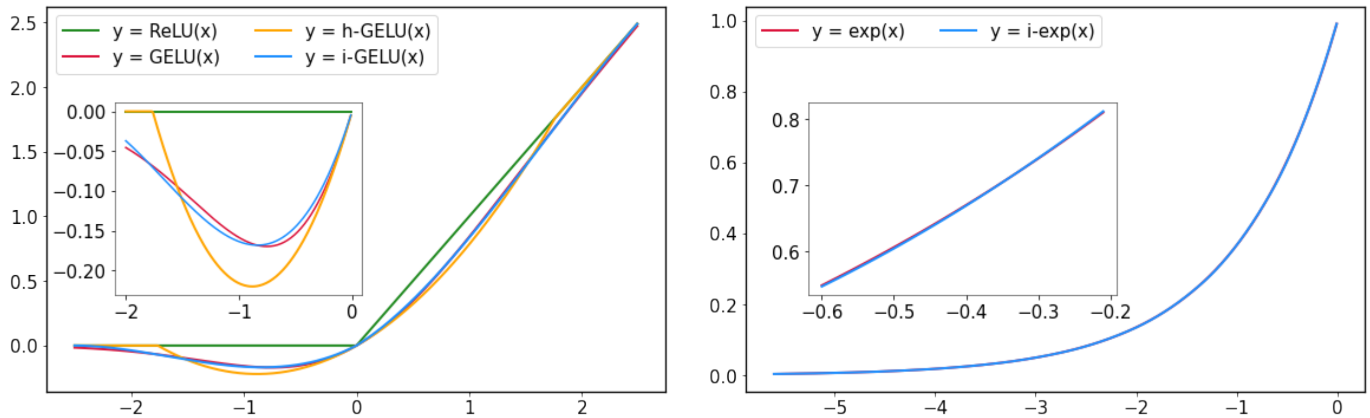

Fig. 2: *(Left) Comparison between RELU, GELU, h-GELU and i-GELU. (Right) Comparison between exponential (exp) and our integer-only exponential (i-exp). The inner plot focuses on the interval [-0.5, -0.1] where the gap between two functions is maximized.*

that interpolates a target function is guaranteed to be a good approximation for it. We refer interested readers to [55] for more details on polynomial interpolation.

While choosing a high-order polynomial results in better error, it is challenging to evaluate them with low-precision integer-only arithmetic. This is due to overflow that can happen when multiplying integer values. For every multiplication, we need to use double bit-precision to avoid overflow. Aside from this, using a high-order polynomial is not desirable since it has higher computational overhead for evaluating the target non-linear function. 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 Section III-D and III-E, respectively, where we show that one can get a close approximation by using only a 2nd-order polynomial.

D. *Integer-only GELU*

GELU [20] is a non-linear activation function used in Transformer based models, defined as follows:

$$\mathrm{GELU}(x) := x \cdot \frac{1}{2}\left[1 + \mathrm{erf}(\frac{x}{\sqrt{2}})\right], \tag{6}$$

where erf is the error function defined as

$$\mathrm{erf}(x) := \frac{2}{\sqrt{\pi}} \int_0^x \exp{(-t^2)} dt. \tag{7}$$

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. Due to the computational overhead of the integral, it is inefficient to compute directly the erf function for each input. For this reason, several different approximations have been proposed for evaluating GELU. For example, [20] uses Sigmoid to approximate erf:

$$\mathrm{GELU}(x) \approx x\sigma(1.702x), \tag{8}$$

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) as proposed by [22] to obtain an integer-only approximation for GELU:

$$\text{h-GELU}(x) := x\frac{\mathrm{ReLU6}(1.702x + 3)}{6} \approx \mathrm{GELU}(x). \tag{9}$$

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 Transformer based models results in a significant accuracy drop. This is due to the large gap between h-GELU and GELU as depicted in Table I.[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:

$$\begin{aligned} &\min_{a,b,c} \frac{1}{2}\|GELU(x) - x \cdot \frac{1}{2}\left[1 + \mathrm{L}(\frac{x}{\sqrt{2}})\right]\|_2^2, \\ &\text{s.t.} \quad L(x) = a(x+b)^2 + c, \end{aligned} \tag{10}$$

[2]Below, in our ablation study, we show this can lead to (up to 2.2 points) accuracy degradation, as reported in Table III.

Table I: *Comparison of different approximation methods for GELU. 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 (average) distance and L^∞ (maximum) distance of GELU and its approximations across the range of [-4, 4]. Our approximation method, i-GELU, outperforms other methods proposed in the previous works in terms of both average and maximum error.*

	Int-only	L^2 dist	L^∞ dist
$x\sigma(1.702x)$ [20]	✗	0.012	0.020
h-GELU [22]	✓	0.031	0.068
i-GELU (Ours)	✓	**0.0082**	**0.018**

Algorithm 2 Integer-only Computation of GELU

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

```
function I-ERF(q, S)
    Store the sign bit of q at q_sgn
    Clip q such that Sq lies in [0, 1.769]
    q_L, S_L ← I-POLY(|q|, S): Compute Eq. 11
    q_out, S_out ← q_sgn q_L, S_L
    return q_out, S_out
end function

function I-GELU(q, S)
    q_erf, S_erf ← I-ERF(q, S/√2)
    q_1 ← ⌊1/S_erf⌋
    q_out, S_out ← q(q_erf + q_1), S S_erf/2
    return q_out, S_out
end function
```

where $\mathrm{L}(x)$ is used to approximate the erf function. Directly optimizing Eq. 10 results to a bad approximation since the definition domain of erf is the entire real numbers. To address this, we only optimize $L(x)$ to a limited range since erf approaches to 1 (-1) when x is close to large positive (negative) numbers. Besides, since erf is an odd function, i.e., $\mathrm{erf}(-x) = -\mathrm{erf}(x)$, we only need to optimize the positive domain. These adjustments together result in the following polynomial,

$$\mathrm{L}(x) = \mathrm{sgn}(x)\left[a(\mathrm{clip}(|x|, \max = -b) + b)^2 + 1\right], \tag{11}$$

where $a = -0.2888$ and $b = -1.769$. Note that $L(x)$ in the limited range of $[0, -b]$ is being approximated. Using this polynomial we arrive at i-GELU, the integer-only approximation to GELU, defined as follows:

$$\text{i-GELU}(x) := x \cdot \frac{1}{2}\left[1 + \mathrm{L}(\frac{x}{\sqrt{2}})\right] \tag{12}$$

Algorithm 2 summarizes the integer-only computation of GELU using i-GELU.

We illustrate the behaviour of i-GELU in Figure 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 Table I, where i-GELU has an average error of 8.2×10^{-3} and the 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. 8, but without using any floating point arithmetic (note that computing the Sigmoid requires floating point, as presented in Eq. 8).

E. *Integer-only Softmax*

Softmax normalizes the input and maps it to a probability distribution, where the probabilities are proportional to the exponential of the input numbers:

$$\mathrm{Softmax}(x)_i := \frac{\exp x_i}{\sum_{j=1}^{k} \exp x_j} \quad \text{where} \quad x = [x_1, \ldots, x_k]. \tag{13}$$

Similar to GELU, prior Transformer quantization research [2, 71] treat this layer using floating point arithmetic. Approximating the Softmax layer with integer arithmetic is quite challenging, as the exponential function used in Softmax is unbounded and changes rapidly.

To avoid the overflow issue and limit the approximation range of Softmax, we subtract the maximum value from the exponential, i.e.,

$$\mathrm{Softmax}(x)_i = \frac{\exp(x_i - x_{\max})}{\sum_{j=1}^{k} \exp(x_j - x_{\max})}, \tag{14}$$

where $x_{\max} = \max_i(x_i)$. Note that now all the values of $\tilde{x}_i = x_i - x_{\max}$ will be non-positive, and this version does not involve any approximation. However, the calculation still needs floating point arithmetic. In order to address this, we first represent a negative real number $\tilde{x}_i$ as:

$$\tilde{x}_i = (-\ln 2)z_i + p_i, \tag{15}$$

where the quotient z_i is a non-negative integer and the remainder p_i is a real number in $(-\ln 2, 0]$. Then, the exponential of $\tilde{x}_i$ can be rewritten as:

$$\exp(\tilde{x}_i) = 2^{-z_i}\exp(p_i) = \exp(p_i) \gg z_i, \tag{16}$$

where $\gg$ is the bit shifting operation. As a result, we only need to approximate the exponential function in the

Algorithm 3 Integer-only Computation of 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)
  Let k a large enough integer
  $q_{\ln 2} \leftarrow \lfloor \ln 2/S \rfloor$
  $z \leftarrow \lfloor -q/q_{\ln 2} \rfloor$
  $q_p \leftarrow q + zq_{\ln 2}$
  $q_L, S_L \leftarrow$ I-POLY(q_p, S) : Compute Eq. 17
  $q_{out}, S_{out} \leftarrow q_L{<}{<}(k-z), 2^{-k}S_L$
  return q_{out}, S_{out}
end function

function I-SOFTMAX(q, S)
  Let k a large enough integer
  $q_{\exp}, S_{\exp} \leftarrow$ I-EXP(q, S)
  $q_{\text{sum}} \leftarrow \text{sum}(q_{\exp})$
  $q_{out}, S_{out} \leftarrow (q_{\exp}{<}{<}k)/q_{\text{sum}}, 2^{-k}S_{\exp}$
  return q_{out}, S_{out}
end function

interval $(-\ln 2, 0]$ where p_i lies. This is a much smaller range as compared to the domain of all real numbers.

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

$$L(p) = 0.3585(p + 1.353)^2 + 0.344 \approx \exp(p). \quad (17)$$

Substituting the exponential term in Eq. 16 with this polynomial results in i-exp, i.e.,

$$\text{i-exp}(\tilde{x}_i) := L(p_i){>}{>}z_i \quad (18)$$

where $z_i = \lfloor -\tilde{x}_i/\ln 2 \rfloor$ and $p_i = \tilde{x}_i + z_i \ln 2$. This can be calculated with integer arithmetic. Algorithm 3 describes the integer-only computation of the exponential and Softmax using i-exp. Figure 2 (right) plots the result of i-exp, which is nearly identical to 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.

F. Integer-only LayerNorm

LayerNorm is commonly used in Transformers and involves several non-linear operations, such as division, square, and square root, when it normalizes the input across the channel dimension. The normalization process is described as follows:

Algorithm 4 Integer Square Root

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

function INTSQRT(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

$$\tilde{x} = \frac{x - \mu_L}{\sigma_L}, \quad (19)$$

where μ_L and σ_L are the mean and standard deviation of the input across the channel dimension:

$$\mu_L = \frac{1}{C}\sum_{i=1}^{C} x_i \text{ and } \sigma_L = \sqrt{\frac{1}{C}\sum_{i=1}^{C}(x_i - \mu_L)^2}. \quad (20)$$

One subtle challenge here is that the statistics of inputs (i.e., μ_L and σ_L) change rapidly for NLP tasks and these values need to be calculated dynamically during runtime. While computing μ_L is straightforward, evaluating σ_L requires square-root function. This square-root function can be efficiently evaluated with integer-only arithmetic through an iterative algorithm proposed in [9], as described in Algorithm 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. The calculations only involve integer arithmetic and is guaranteed to be completed within the time complexity of $O(\log \log n)$. We refer the reader to [9] for more details. This algorithm is computationally lightweight, as it is completed within at most four iterations for any 32-bit inputs and each iteration consists only of one integer division, one integer addition, and one bit-shift operation. The rest of the the non-linear operations in LayerNorm such as division and square are straightforwardly computed with integer arithmetic. Thus, with the above techniques, we can efficiently evaluate LayerNorm with integer arithmetic as well.

IV. RESULTS

In this section, we test I-BERT on the RoBERTa [36] model using [41]. For the integer-only implementation,

Table II: *Integer-only quantization result for RoBERTa-Base and RoBERTa-Large on the development set of the GLUE benchmark. For the downstream tasks that have two evaluation metrics, we report the average of them. 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

	Int-only	MNLI-m	MNLI-mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg.
Baseline	✗	**87.8**	**87.4**	**90.4**	**92.8**	94.6	61.2	**91.1**	90.9	78.0	86.0
I-BERT	✓	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

	Int-only	MNLI-m	MNLI-mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg.
Baseline	✗	90.0	89.9	92.8	94.1	96.3	68.0	**92.2**	91.8	86.3	89.0
I-BERT	✓	**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

we replace all the floating point operations in the original model with the corresponding integer-only operations that were discussed in Section III. In particular, all the MatMul operations are performed with INT8 precision, and accumulated to INT32 precision. Furthermore, the Embedding layer is kept at INT8 precision. Moreover, the non-linear activations (i.e., GELU and Softmax), as well as the 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 the computational complexity of these operations (i.e., GELU, Softmax, and LayerNorm) is linear. We perform Requantization [69] operation after these operations to bring the precision down from INT32 to INT8 so that the successor operations (such as MatMul) can be performed with low precision. We test our methods on both RoBERTa-Base and RoBERTa-Large. However, our method is not restricted to RoBERTa. It can be applied to any other Transformer based model or, more generally, to other NN models that use similar non-linear activation functions.

We evaluate I-BERT on the General Language Understanding Evaluation [59] (GLUE) benchmark. GLUE 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 [69] 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.

For evaluating the results, we use the standard metrics for each task in GLUE. In particular, we use accuracy and F1 score for QQP [26] and MRPC [13], Pearson Correlation and Spearman Correlation for STS-B [4], and Mathews Correlation Coefficient for CoLA [62]. For the remaining tasks [10, 47, 54, 63], we use classification accuracy. Since there are two development sets for MNLI [63], 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 [33] as it has relatively small dataset and shows an unstable behaviour [12].

The integer-only quantization results for RoBERTa-Base and RoBERTa-Large are presented in Table II. As one can see, I-BERT consistently achieves comparable or better accuracy to the baseline. For RoBERTa-Base, the integer-only quantization achieves higher accuracy for all cases (up to 1.4 for RTE), except for MNLI-m and QQP tasks, where we observe a small accuracy degradation of

Table III: *Accuracy of models that use GELU, h-GELU and i-GELU for GELU computation. Note that the former is the full-precision, floating point computation while the latter two are the 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**

0.3 and 0.2, respectively. We observe a similar behaviour on the RoBERTa-Large model, where I-BERT accuracy matches or outperforms the baseline accuracy for all the downstream tasks. On average, I-BERT outperforms the baseline by 0.3 and 0.5 for RoBERTa-Base and RoBERTa-Large respectively.

V. Ablation Study

In this section, we empirically validate i-GELU, our approximation method for GELU. For comparison, we implement two variants of I-BERT by replacing i-GELU with GELU and h-GELU respectively. Note that the former is the exact computation of GELU with floating point arithmetic and the later is another integer-only approximation method for GELU (see Section III). We use RoBERTa-Large model as baseline, and we use QNLI, SST-2, MPRC, and RTE for downstream tasks. All models are trained and fine-tuned according to the procedure described in Section IV. We report the accuracy in Table III.

As one can see in the table, 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. Interestingly, approximating GELU with i-GELU results in strictly better accuracy for all four downstream tasks than approximating GELU with h-GELU. 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. This implies that i-GELU allows tighter approximation for GELU, as compared to h-GELU.

VI. Conclusions

We have proposed I-BERT, a novel integer-only quantization scheme for Transformer-based models, in which the entire inference is performed with integer-only arithmetic. At the core of our approach in I-BERT are approximation methods for nonlinear operations such as GELU, Softmax, and LayerNorm, which enable their approximation with integer computation. We empirically evaluated our quantization method on RoBERTa-Base and RoBERTa-Large, where our method exceeds the floating point baseline model by 0.3 and 0.5 points respectively in terms of the average GLUE score. Our method is not just limited to a specific model, but it can be applied to any Transformer based models or, in general, other models that use similar non-linear operations. For this reason, by enabling efficient hardware design as well as better model deployment on efficient specialized hardware, our method could be the key for bringing computationally heavy Transformer based language models onto edge devices.

Acknowledgments

The UC Berkeley team acknowledges gracious support from Intel corporation, Intel VLAB team, Google Cloud, Google TFTC 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

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram Saletore. Efficient 8-bit quantization of transformer neural machine language translation model. *arXiv preprint arXiv:1906.00532*, 2019.

[3] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[4] Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*, 2017.

[5] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srini-

vasan, and Kailash Gopalakrishnan. PACT: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

[6] ARM Cortex-M. https://developer.arm.com/ip-products/processors/cortex-m.

[7] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[8] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[9] Richard Crandall and Carl B Pomerance. *Prime numbers: a computational perspective*, volume 182. Springer Science & Business Media, 2006.

[10] Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, pages 177–190. Springer, 2005.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[12] Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*, 2020.

[13] William B Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

[14] Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. HAWQ-V2: Hessian Aware trace-Weighted Quantization of neural networks. *NeurIPS'19 workshop on Beyond First-Order Optimization Methods in Machine Learning.*, 2019.

[15] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. HAWQ: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 293–302, 2019.

[16] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, and Kurt Keutzer. SqueezeNext: Hardware-aware neural network design. *Workshop paper in CVPR*, 2018.

[17] Song Han and B Dally. Efficient methods and hardware for deep learning. *University Lecture*, 2017.

[18] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[20] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

[21] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *Workshop paper in NIPS*, 2014.

[22] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for MobilenetV3. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1314–1324, 2019.

[23] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[24] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[25] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[26] Shankar Iyer, Nikhil Dandekar, and Kornl Csernai. First quora dataset release: Question pairs.(2017). *URL https://data. quora. com/First-Quora-Dataset-Release-Question-Pairs*.

[27] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern*

Recognition, pages 2704–2713, 2018.

[28] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

[29] Kiseok Kwon, Alon Amid, Amir Gholami, Bichen Wu, Krste Asanovic, and Kurt Keutzer. Co-design of deep neural nets and neural net accelerators for embedded vision applications. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[30] Liangzhen Lai, Naveen Suda, and Vikas Chandra. CMSIS-NN: Efficient neural network kernels for arm cortex-m cpus. *arXiv preprint arXiv:1801.06601*, 2018.

[31] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[32] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

[33] Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*. Citeseer, 2012.

[34] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

[35] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[36] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[37] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet V2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018.

[38] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J Dally. Exploring the regularity of sparse structure in convolutional neural networks. *Workshop paper in CVPR*, 2017.

[39] Asit Mishra and Debbie Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *arXiv preprint arXiv:1711.05852*, 2017.

[40] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[41] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. FairSeq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[42] Eunhyeok Park, Sungjoo Yoo, and Peter Vajda. Value-aware quantization for training and inference of neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 580–595, 2018.

[43] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.

[44] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018.

[45] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[46] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[47] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[48] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[49] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. FitNets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

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

[51] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mo-

bilenetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[52] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Q-BERT: Hessian based ultra low precision quantization of bert. In *AAAI*, pages 8815–8821, 2020.

[53] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[54] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[55] Gilbert W Stewart. *Afternotes on numerical analysis*. SIAM, 1996.

[56] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[57] Mingxing Tan and Quoc V Le. EfficientNet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[58] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[59] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[60] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-aware automated quantization. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019.

[61] Edward Waring. Vii. problems concerning interpolations. *Philosophical transactions of the royal society of London*, (69):59–67, 1779.

[62] Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019.

[63] Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

[64] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[65] Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. Mixed precision quantization of convnets via differentiable neural architecture search. *arXiv preprint arXiv:1812.00090*, 2018.

[66] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[67] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5687–5695, 2017.

[68] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. XLNet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5753–5763, 2019.

[69] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael W Mahoney, and Kurt Keutzer. HAWQV3: Dyadic neural network quantization. *arXiv preprint arXiv:2011.10680*, 2020.

[70] Hongxu Yin, Pavlo Molchanov, Jose M Alvarez, Zhizhong Li, Arun Mallya, Derek Hoiem, Niraj K Jha, and Jan Kautz. Dreaming to distill: Data-free knowledge transfer via deepinversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8715–8724, 2020.

[71] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8BERT: Quantized 8bit bert. *arXiv*

preprint arXiv:1910.06188, 2019.

[72] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. LQ-Nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 365–382, 2018.

[73] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.