Integer Fine-tuning of Transformer-based Models

Mohammadreza Tayaranian,\textsuperscript{1} Alireza Ghaffari,\textsuperscript{1} Marzieh S. Tahaei,\textsuperscript{1} Mehdi Rezagholizadeh,\textsuperscript{1} Masoud Asgharian,\textsuperscript{2} Vahid Partovi Nia\textsuperscript{1}

\textsuperscript{1} Huawei Noah’s Ark Lab
Montreal Research center
7101 Park Avenue, Montreal, Canada
\textsuperscript{2} McGill University
Department of Mathematics and Statistics
805 Rue Sherbrooke West, Montreal, Canada

Abstract
Transformer based models are used to achieve state-of-the-art performance on various deep learning tasks. Since transformer-based models have large numbers of parameters, fine-tuning them on downstream tasks is computationally intensive and energy hungry. Automatic mixed-precision FP32/FP16 fine-tuning of such models has been previously used to lower the compute resource requirements. However, with the recent advances in the low-bit integer back-propagation, it is possible to further reduce the computation and memory foot-print. In this work, we explore a novel integer training method that uses integer arithmetic for both forward propagation and gradient computation of linear, convolutional, layer-norm, and embedding layers in transformer-based models. Furthermore, we study the effect of various integer bit-widths to find the minimum required bit-width for integer fine-tuning of transformer-based models. We fine-tune BERT and ViT models on popular downstream tasks using integer layers. We show that 16-bit integer models match the floating-point baseline performance. Reducing the bit-width to 10, we observe 0.5 average score drop. Finally, further reduction of the bit-width to 8 provides an average score drop of 1.7 points.

Introduction
Over the past few years, integration of attention mechanisms into deep learning models led to the creation of transformer-based models. These models have shown state-of-the-art performance in various deep learning tasks such as natural language processing (NLP) and image classification. Training the transformer-based models include a pre-training phase on a large dataset, followed by a fine-tuning phase on a more specialized downstream task. Although by design, the size of the fine-tuning datasets and their number of iterations are smaller than pre-training, the fine-tuning phase still requires considerable amount of memory and computational resource. Thus, researchers actively try to optimize the training procedure of the transformer-based models for both pre-training and fine-tuning phases.

Transformer-based models require high computational resources and memory cost due to their large number of parameters. Having large number of parameters incur challenges for inference, training, and also fine-tuning of such models. Also note the training phase, i.e. pre-training and fine-tuning, involves more operations compared to inference. To be more specific, the training phase includes gradient computation and weight update, which make the training a more compute intensive task. Furthermore, by optimizing the training procedure of transformer-based models, it is possible to bring the fine-tuning to the edge devices that do not have enough memory capacity and computational resources.

Recently researchers have used efficient fine-tuning methods to reduce the computational complexity of the transformer-based models. These methods mainly involve reducing the number of trainable parameters during the fine-tuning phase (Zaken, Ravfogel, and Goldberg 2021; Vucetic et al. 2022; Houlsby et al. 2019; Hu et al. 2021).

Another prominent method to reduce the computational complexity of deep learning models is quantization. In the quantization literature, deep learning models are compressed by representing their parameters in lower bit-width data types. Representing the parameters in the lower-bit
data types leads to more efficient computation compared to floating-point numbers and also reduces the memory footprint of the model.

Although quantization techniques are applicable for both inference and training phases, previous attempts to quantize transformer-based models were only focused on low-bit inference (Bhandare et al. 2019; Kim et al. 2021; Zafrir et al. 2019; Lin et al. 2021).

Unlike previous works, we propose to use integer values for both fine-tuning and inference phases to exploit the compression that is gained from representing the gradients as integer values. To the best of our knowledge, this is the first time that integer numbers are used for back-propagation of the transformer-based models. Moreover, as shown in Figure 1, integer data types offer better computation throughput and energy consumption compared to the floating-point data type. It is also noteworthy to mention that currently, the only low-bit fine-tuning method for transformer-based models is to use NVIDIA’s automatic mixed-precision (AMP) 1 which uses FP16 for some intermediate values, e.g. activations, parameters, and gradients, while using single-precision floating-point (FP32) for weight update as proposed in (Micikevicius et al. 2017).

Here, we propose a novel methodology to perform fine-tuning of transformer-based models using integer arithmetic. In this method, we replace floating-point numbers by integer numbers with a negligible degradation in the fine-tuning performance metrics. We use the dynamic fixed-point number format to convert floating-point numbers to fixed-point integers. Conversions between the floating-point and fixed-point number formats has been previously introduced by (Ghaffari et al. 2022). Note that these integer mapping functions enable us to use integer arithmetic for both the gradient computation and forward propagation of compute intensive layers. During fine-tuning, we substitute all the linear, convolutional, layer normalization (layer-norm), and embedding layers with their integer versions and achieve the FP32 baseline fine-tuning performance (e.g. accuracy, Matthews correlation, etc.) using integer arithmetic.

In our proposed method, a b-bit dynamic fixed-point mapping strategy is presented and the effect of varying bit-width for different parts of network such as weights, gradients, and activations is studied. We show that low-bit integer weights and activations are enough to fine-tune the transformer-based models. Our b-bit integer mapping strategy is closely related to floating-point number format and hence is used alongside floating-point number formats in a mixed-precision training. In our proposed strategy, the arithmetic of all the compute intensive layers in forward and back propagation are performed in integer format while other components of the model, such as nonlinear activation functions and the weight update are kept in FP32 number format. Our experimental results show that int8 weights with int12 activations are enough for fine tuning transformer-based models.

To this end, this paper makes the following contributions:

• Integer fine-tuning method for transformer-based models

Remark 3 discusses that convergence behaviour of our integer fine-tuning is directly related to the variance of dynamic fixed-point mapping and is controlled by the bit-width b.

• We fine-tuned transformer-based models with int8 weights and int12 activations. Our integer fine-tuning method shows an average of 2.3 and 4.0 points of score drop for BERT on GLUE and SQuAD datasets and 0.3 and 1 points for ViT on CIFAR-10 and CIFAR-100.

The rest of this paper is structured as follows. Related works discusses previous work on quantization of transformer-based models as well as its differences with our integer fine-tuning method. Background section clarifies the dynamic fixed-point mapping functions that facilitate the conversion of values between dynamic fixed-point and the floating-point number formats. In the Methodology section, our proposed dynamic b-bit fixed-point mapping is described. Finally, the integer fine-tuning experiments on transformer-based models are presented.

Related Works

Compression of deep learning models can be divided into two major categories, low-bit inference and low-bit training. Most of the transformer-based models are compressed for inference. However, to the best of our knowledge, low-bit integer fine-tuning of transformed-based models has not been studied in the literature.

Low-bit Inference

The goal of quantization the inference is to speed up the model by quantizing values for the forward propagation. This category is itself divided into quantization-aware training (QAT) and post-training quantization (PTQ) methods.

In QAT, quantization is performed during the training or fine-tuning, allowing the model parameters to adapt to the quantization noise. QAT relies on high-precision floating-point gradients to train the model and adapt it to quantization noise. For instance, (Zafrir et al. 2019) proposed QSBERT which quantizes the inference computations of all linear and embedding layers of BERT to 8-bit integers and updates the quantization scale with a moving average. Similarly, (Shen et al. 2020) suggested Q-BERT which requires the computation of hessian matrix for each group of parameters to be used in mixed-precision fine-tuning with different bit-widths. (Kim et al. 2021) proposed I-BERT that uses a uniform quantization scheme to quantize input activations and weights of various components of BERT. In I-BERT, the quantizations’ scaling factors are computed based on the
distribution of the training data. (Li et al. 2022) introduced Q-ViT, which is a training-aware quantization method with a scale that is learned during training. Q-ViT quantizes all the linear layers of ViT’s forward propagation using 3-bit integer values.

Unlike QAT that performs quantization during training, Post-Training Quantization (PTQ) methods apply quantization on the parameters only when the training or fine-tuning is completed. Thus, they require extra calibration or parameter tuning to adapt the model to the quantized parameters. As an example, (Bhandare et al. 2019) quantized the matrix multiplications of the original transformer architecture from (Vaswani et al. 2017) to 8-bit integer values. Moreover, the quantization is done only for the forward propagation and requires extra calibration using validation data to tune the boundaries of the quantization function. (Zadeh et al. 2020) introduced GOBO which compresses the fine-tuned weights of BERT by grouping them into two categories of Gaussian and outlier. The outlier weights are kept in FP32, while the Gaussian weights are quantized to lower bits. GOBO also proposes a specialized hardware accelerator to exploit its compression scheme to speed up the inference (Zadeh et al. 2020). (Liu et al. 2021) applied a ranking-aware quantization method that solves an optimization problem to keep the rank of attention mechanism after quantization. FQ-ViT, introduced by (Lin et al. 2021), quantizes the weights and activations of all the layers in ViT using 8-bit integer after training. In FQ-ViT, the quantization function is calibrated by a search algorithm, using a subset of training and evaluation datasets. PTQ4ViT is another similar post-training quantization method proposed by (Yuan et al. 2021) that finds the optimal bit-width to quantize the ViT layers using a search method. For lower bit-width regimes, TernaryBERT and BinaryBERT are able to push the quantization to 2 and 1 bits respectively. They both rely on methods such as data augmentation and knowledge distillation to adapt the model to the low-bit weights. They also keep the layer-norm and softmax layers in FP32 to maintain metric performance (Zhang et al. 2020a; Bai et al. 2020).

**Low-bit Training**

Recent researches in low-bit training try to perform the gradient computation using low-bit arithmetic in addition to the forward propagation computations. Using lower bits for gradients reduces the model’s ability to adapt the parameters to the quantization noise, while it increases the throughput and reduces the memory footprint.

For instance, (Zhang et al. 2020b) quantized the input activations, gradients and parameters of the linear layers for various convolutional and recurrent neural networks. The quantization error is measured during training and used to adjust the quantization scale, making the method dependent on the training data distribution. (Zhu et al. 2020) applied a similar quantization scheme for CNNs architectures while also utilizing “direction sensitive gradient clipping” and learning rate scaling to control the quantization error of gradients. (Zhao et al. 2021) adapted the quantization parameters by detecting the distribution of the gradients in the channel dimension. Finally, (Ghaffari et al. 2022) used a dynamic fixed-point number format for integer training. Their proposed integer training method neither depends on the distribution of the data nor relies on gradient clipping or learning rate adjustment. They also perform all the batch-norm computations in forward propagation and back-propagation using integer arithmetic.

**Background**

Here we briefly review the mapping functions introduced by (Ghaffari et al. 2022). We use these mapping functions to convert numbers between floating-point and dynamic fixed-point formats. A linear fixed-point mapping function is used to map floating-point numbers to the integer numbers, while a non-linear inverse mapping converts them back to floating-point if needed.

**Linear Fixed-point Mapping**

Unlike other common quantization schemes, the linear fixed-point mapping utilizes the inherent properties of the IEEE 754 floating-point standard (Zuras et al. 2008), to obtain the dynamic fixed-point representation.

The linear fixed-point mapping takes a floating-point tensor \( F \) and unpacks it to the sign, exponent, and mantissa tensors. These tensors contain \((s_i, e_i, m_i)\) for each floating-point number \( f_i \). Note that for FP32 data type, exponent and mantissa values are 8-bit and 24-bit integers respectively. For mantissa tensor, the mapping method includes the implicit hidden bit to the 23-bit IEEE standard mantissa tensor. In the next step, linear fixed-point mapping calculates the shared scale of the tensor as \( \epsilon_{\text{scale}} = \max(e_1, \ldots, e_i, \ldots, e_n) \). Each \( m_i \) is then shifted to right by \( \epsilon_{\text{scale}} - e_i \) bits. Finally, to obtain 8-bit dynamic fixed-point numbers, each \( m_i \) should be rounded to 7 bits, reserving one bit for sign.

**Non-linear Inverse Mapping**

The non-linear inverse mapping takes the integer tensor, \( m \), and its scale \( \epsilon_{\text{scale}} \), and converts it to floating-point format.

First, the exponent tensor \( e \) is created with the same shape as \( m \), and then filled with \( \epsilon_{\text{scale}} \). In the next step, the integer mantissas in \( m \) are normalized. To conform with IEEE standard (Zuras et al. 2008), a normalized mantissa starts with a \((1)_2\) as its most significant bit. To do so, the sign of each \( m_i \) is extracted and the unsigned \( m_i \) is shifted until its 24\textsuperscript{th} bit is \((1)_2 \). To maintain the values of each number, for every shift of mantissa \( m_i \), the exponent \( e_i \) is adjusted by one.

**Methodology**

We use the linear mapping and non-linear inverse mapping functions described previously to fine-tune the transformer-based models using integer arithmetic. In our proposed method, we use \( b \)-bit dynamic fixed-point mappings in order to convert floating-point numbers to integer values. The main difference of our work and what proposed by (Ghaffari et al. 2022) is that \( b \)-bit dynamic fixed-point mapping allows to improve the fine-tuning convergence of integer models by controlling the variance induced by the mapping functions. Furthermore, the \( b \)-bit integer arithmetic is used for compute
intensive layers, while the layers that need more precision (e.g. SoftMax in the attention mechanism) are kept in FP32. Note that using integer arithmetic for certain operations is similar to mixed-precision training in which FP16 is used alongside the higher precision FP32 data type [Mickey et al. 2017]. Moreover, using integer arithmetic leads to better throughput when compared with the floating-point data type, see Figure 1.

**b-bit Dynamic Fixed-point Mapping**

We use b-bit dynamic fixed-point mapping to explore the effect of various bit-widths for components of the transformer-based models. As depicted in Figure 2 to map FP32 values to the b-bit dynamic fixed-point numbers, the rounding function is manipulated to control the bit-width of the generated integer. In this method, the bit-width of input activations and parameters of the integer layers is imposed by adjusting $b$ in the dynamic fixed-point mapping. The motivation of having $b$-bit dynamic fixed-point is to control the resolution of integer tensors, and consequently limiting the variance induced by the linear mapping. This phenomenon is analysed in the next section. Our experiments show that using b-bit dynamic fixed-point with $b \geq 10$ achieves the floating-point baseline performance. Figure 3 demonstrates the F1 score of fine-tuning BERT using $b$-bit gradients, and activations on SQuAD v2.0 dataset. For the 8-bit and 9-bit fixed-point bit-widths, we use 12-bit input activations. The reason for using higher bit-width input activations is that we observed lower-bit activations, such as 8-bit, dramatically reduces the F1 score. Figure 4 shows the effect of input activation bit-width on the F1 score when the weight bit-width is set to 8-bit. Observe that having 8-bit input-activation results in an F1 score of 50. Furthermore, note that changing the bit-width of the input activation from 8-bit to 12-bit significantly increases the F1 score. Increasing the bit-width of the input activations beyond 12-bit has a negligible effect on the F1 score, confirming that 12-bit input activation is enough for this application.
Here, we study the effect of simplified assumptions. Let us assume the following simplified mapping on the stochastic gradient descent method under simplify on the variance of b-bit dynamic fixed-point.

**Remark 3**

SQuAD v2.0 dataset. Note that \( \hat{M} \) is the added variance of b-bit dynamic fixed-point gradient and \( q = \| \nabla L(w) \| \). We can relate \( \hat{a}_{ij} \) and \( \hat{d}_{ii} \) with an error term \( \delta \) such as \( \hat{a}_{ij} = a_{ij} + \delta_{ij} \). For a linear layer \( \hat{Y} = XW \), the computation of the b-bit dynamic fixed-point gradients in the back-propagation is

\[
\hat{C} = \frac{\partial L}{\partial \hat{W}} = \frac{\partial Y}{\partial \hat{W}} \frac{\partial L}{\partial Y} = X^\top \frac{\partial L}{\partial \hat{Y}} = X^\top \hat{G}. \tag{4}
\]

It is of interest to find the relation between \( \hat{C} = X^\top \hat{G} \) in the integer back-propagation and the true gradients \( C = X^\top G \). We can derive the variance for each element \( \hat{c}_{ij} \) by expanding the error terms \( \delta \).

\[
V\{\hat{c}_{ij}\} = V\left\{ \sum_{n=1}^{N} x_{ni} \tilde{g}_{nj} \right\}
= V\left\{ \sum_{n=1}^{N} (x_{ni} + \delta_{ni})(g_{nj} + \delta_{nj}) \right\}
\leq V\left\{ \sum_{n=1}^{K} x_{ni} g_{nj} \right\} + \sigma_{G}^2 E\{||X^\top||_2^2\} + \sigma_{G}^2 E\{||G||_2^2\} + N \sigma_{X}^2 \sigma_{G}^2.
\tag{5}
\]

In inequality (5), \( \sigma_{G}^2 = \max_{ij} \{E\{||G_j||_2^2\}\} \) and \( \sigma_{X}^2 = \max_{ij} (E\{||X^\top||_2^2\}) \). Also note that \( ||X^\top||_2^2 = \sum_{i}^J x_{ni}^2 \) denotes the squared L-2 norm of the \( i^{th} \) row of \( X^\top \) and \( ||G||_2^2 = \sum_{i}^J g_{ij}^2 \) denotes the squared L-2 norm of the \( j^{th} \) column of \( G \). Furthermore, by defining

\[
\begin{cases}
M := \sigma_{G}^2 (E\{||X^\top||_2^2\}) + N \sigma_{X}^2 \\
M^\prime := \sigma_{X}^2
\end{cases}
\tag{6}
\]

we have

\[
V\{\hat{c}_{ij}\} \leq V\{c_{ij}\} + M \sigma_{\hat{G}}^2 + M^\prime \sigma_{\hat{G}}^2
\tag{7}
\]

that closely matches Assumption 2. Equation (6) shows that \( M^\prime \) depends on variance of b-bit dynamic fixed-point mapping for input activations and gradients while \( M_\prime \) only depends on b-bit dynamic fixed-point gradients variance.

**Proposition 1.** With b-bit dynamic fixed-point representation of tensor \( \hat{A} \), the variance of error \( \delta A \) satisfies the following inequality

\[
V\{\delta A\} \leq 2^{\epsilon_{\text{scale}} - b - 2}. \tag{8}
\]
Proof. Using $b$-bit dynamic fixed-point mapping, the error $\epsilon^A$ satisfies the following bound

$$-2^{e_{\text{scale}}^A} (0.000001)_b \leq \epsilon^A \leq 2^{e_{\text{scale}}^A} (0.000001)_b - 2^{b-1}$$

$$-2^{e_{\text{scale}}^A} b^{b+2} \leq \epsilon^A \leq 2^{e_{\text{scale}}^A} b^{b+2}.$$  \hfill (9)

Thus, the inequality (8) is obtained by using Popoviciu’s inequality on variances

$$\forall \{\epsilon^A\} \leq 1 \left(2^{e_{\text{scale}}^A} b^{b+2} - \left(2^{e_{\text{scale}}^A} b^{b+2}\right)^2\right) \leq 2^{2(2^{e_{\text{scale}}^A} b^{b+2}).}$$ \hfill (10)

**Remark 3.** Inequality (8) shows that increasing bit-width $b$ in dynamic fixed-point mapping reduces the variance of the error. This confirms our experimental results on SQuAD v2.0 dataset that for $b > 10$, F1 score can match FP32 baseline, see Figure 3. Also note in equation (9), both $M^v$ and $M^p$ depend on $b$-bit dynamic fixed point mapping variance of input activation $\sigma^2_X$. Hence, increasing $b$ for input activations while keeping weights in 8-bit format must improve the convergence behaviour. This phenomenon is also confirmed by our experimental results on SQuAD v2.0 dataset demonstrated in Figure 5.

**Integer-only Layers**

The integer fine-tuning of transformer-based models is enabled by using integer-only versions of compute-intensive layers. $b$-bit dynamic fixed-point versions of linear, convolutional, layer-norm, and embedding layers are used where both forward propagation and back-propagation are performed using integer-only arithmetic. In our proposed integer layers, $b$-bit dynamic fixed-point allows to control the variance induced by the linear fixed-point mapping. Also note that using $b$-bit dynamic fixed-point, the input activations and the parameters of our integer layers can accept different bit-widths when needed.

Our proposed $b$-bit dynamic fixed-point mapping can be used alongside FP32 number format in a mixed-precision training setup. Note that mixed-precision training of deep learning models with FP32 and FP16 has been proposed previously by (Micikevicius et al. 2017). However, our method is more advantageous, since as shown in Figure 2 integer arithmetic is more energy efficient and faster than floating-point arithmetic.

Figure 2 shows the implementation of an integer linear layer. First, the parameter and input activation tensors are mapped to $b$-bit dynamic fixed-point number format. Each float tensor is then converted to an integer tensor and a corresponding scale. The integer tensors are sent to an integer matrix multiplication module to compute the output integer tensor. Furthermore, the parameter and the input activation scales are added together to compute the output scale. The output integer tensor and scale are sent to a non-linear inverse mapping function to generate the floating-point output if needed.

In a similar procedure, the gradient computation during the back-propagation is also computed using integer-only arithmetic.

**Experimental Results**

**Experimental Setup**

We fine-tuned BERT (Devlin et al. 2018) and ViT (Dosovitskiy et al. 2020) models on a series of downstream tasks to compare the performance of our integer fine-tuning method with the FP32 baseline. The BERT-Base model is fine-tuned on selected tasks of GLUE benchmark (Wang et al. 2018), along with the Stanford Question Answering Datasets, i.e. SQuAD v1.1 and SQuAD v2.0 (Rajpurkar et al. 2016). We also fine-tuned ViT-Base model for image classification on CIFAR-10 and CIFAR-100 datasets (Krizhevsky, Hinton et al. 2009).

Our integer fine-tuning setups use the same hyper-parameters as the FP32 baseline and are fine-tuned for the same number of epochs. Each reported metric is the average of five runs with five different seeds to mitigate the effects of random variation of the results. The fine-tuning experiments are performed based on the fine-tuning scripts of the Hugging Face library (Wolf et al. 2019).

For GLUE experiments the fine-tuning is performed for 5 epochs and the learning rate is set to $5 \times 10^{-5}$. Also, the per-device fine-tuning batch-size is set to 32. Fine-tuning BERT on SQuAD datasets is done for 2 epochs and the learning rate is set to $5 \times 10^{-5}$ and the per-device fine-tuning batch-size is 12. Fine-tuning ViT on both CIFAR-10 and CIFAR-100 is done for 4 epochs and the learning rate is $5 \times 10^{-5}$ and the per-device fine-tuning batch size is set to 64. Moreover, all experiments are run on eight NVIDIA V100 GPUs with 32 gigabytes of VRAM.

**Results**

BERT: The results of fine-tuning BERT-Base on GLUE benchmark and SQuAD datasets are presented in Table 1 and Table 2 respectively. GLUE benchmark contains a series of downstream tasks, designed to evaluate a diverse set of language understanding abilities of NLP models. SQuAD v1.1 and SQuAD v2.0 datasets contain a series of text passages accompanied by a question and the task is to predict the span of the answer in the passage. Using 16-bit dynamic fixed-point format, BERT is able to either match or outperform the FP32 baseline performance for all tasks. Despite the fact that the 8-bit dynamic fixed-point has half the dynamic range of its 16-bit counterpart, it exhibits an average of 2.3 point drop on GLUE benchmark and 4.0 point drop for SQuAD datasets. Our experiments using 10-bit and 12-bit dynamic fixed-point format show average score drops of 0.2 and 0.6 points for GLUE tasks, and 0.2 and 1.0 points for SQuAD datasets respectively.

ViT: We fine-tuned ViT-Base model for image classification on CIFAR-10 and CIFAR-100 datasets. As presented in Table 3 for CIFAR-10 dataset the 8-bit dynamic fixed-point has a 0.3% drop in the accuracy, while bit-widths more than 8 are able to match the FP32 baseline. For CIFAR-100 dataset, using the 8-bit dynamic fixed-point drops the accuracy by 1%. The 10-bit and 12-bit fixed-point setups match the FP32 baseline, and the 16-bit dynamic fixed-point outperforms the FP32 baseline by 0.1%.

**Loss Trajectory:** Figure 5 shows the loss trajectory of integer fine-tuning BERT on SQuAD v2.0 dataset using 16-bit and 8-bit dynamic fixed-point formats, along with FP32 baseline. The fine-tuning loss trajectory of 16-bit dynamic fixed-point BERT closely follows that of the FP32 baseline. When using dynamic fixed-point with 8-bit parameters and 12-bit input activations, the loss trajectory is slightly shifted, but follows the same trend of the baseline.

**Conclusion**

We proposed a novel integer fine-tuning method for transformer-based models, based on dynamic fixed-point format. We use $b$-bit dynamic fixed-point format to represent parameters, input activations and gradients of BERT and ViT models. As a result, the proposed fine-tuning method uses integer arithmetic for the forward propagation and back-propagation of the linear, convolutional, layer-norm and embedding layers of BERT and ViT models. With our proposed $b$-bit dynamic fixed-point format, we demonstrated that increasing $b$ reduces the variance of the fixed-point mapping. Furthermore, we have shown that the added variance of fixed-point can be controlled by $b$-bit dynamic fixed-point mapping method. In our experiments on BERT model, the 16-bit dy-
Table 1: Metric performance of integer fine-tuning of BERT on selected GLUE tasks. The reported metric for QQP and MRPC is accuracy and F1 score, for QNLI, MNLI, RTE and SST-2 is accuracy, and for CoLA is the Matthews correlation. The number of sentences for each task is reported according to (Wang et al. 2018).

| Task      | QQP | QNLI | MNLI | SST-2 | RTE | MRPC | CoLA |
|-----------|-----|------|------|-------|-----|------|------|
| Number of Sentences | 364k | 105k | 393k | 67k   | 2.5k| 3.7k | 8.5k |
| FP32 Baseline       | 91.0/88.0 | 91.1 | 84.2 | 92.5  | 63.8| **82.5/87.8** | 57.2 |
| 16-bit              | 91.0/88.0 | 91.2 | **84.2** | 92.5  | **64.5** | 82.3/87.6 | 57.7 |
| 12-bit              | 90.9/88.0 | 91.2 | 84    | **92.6** | 63.5| 81.3/87.4 | 56.7 |
| 10-bit              | 90.8/87.8 | 91   | 84    | 92.5  | 62.7| 78.4/85.8 | 57.6 |
| 8-bit               | 90.1/86.8 | 90.8 | 83.7  | 92.3  | 61.8| 76.8/84.7 | 55.0 |

Table 2: Metric performance of fine-tuning BERT on SQuAD v1.1 and v2.0 datasets. For both datasets the exact match metrics and F1 scores are reported.

| Task      | SQuAD v1.1 | SQuAD v2 |
|-----------|------------|----------|
| FP32 Baseline       | 80.5/88.0 | 70.6/73.8 |
| 16-bit              | 80.7/88.0 | 70.6/73.9 |
| 12-bit              | 79.8/87.6 | 70.5/73.8 |
| 10-bit              | 78.4/86.6 | 69.8/73.2 |
| 8-bit               | 75.6/84.5 | 65.5/69.2 |

Table 3: Accuracy of integer fine-tuning ViT on CIFAR-10 and CIFAR-100 datasets.

| Task      | CIFAR-10 | CIFAR-100 |
|-----------|----------|-----------|
| FP32 Baseline       | 98.9 | 91.1 |
| 16-bit              | 98.9 | 91.2 |
| 12-bit              | 98.9 | 91.1 |
| 10-bit              | **98.9** | 91.1 |
| 8-bit               | 98.6 | 90.1 |

Figure 5: Integer fine-tuning loss trajectory of BERT on SQuAD v2.0 dataset for 2750 iterations.

References
Bai, H.; Zhang, W.; Hou, L.; Shang, L.; Jin, J.; Jiang, X.; Liu, Q.; Lyu, M.; and King, I. 2020. Binarybert: Pushing the limit of bert quantization. arXiv preprint arXiv:2012.15701.
Bhandare, A.; Sripathi, V.; Karkada, D.; Menon, V.; Choi, S.; Datta, K.; and Saletore, V. 2019. Efficient 8-bit quantization of transformer neural machine language translation model. arXiv preprint arXiv:1906.00532.
Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.
Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929.
Ghaffari, A.; Tahaei, M. S.; Tayaranian, M.; Asgharian, M.; and Nia, V. P. 2022. Is Integer Arithmetic Enough for Deep Learning Training? arXiv preprint arXiv:2207.08822.
Acknowledgments