Quantization-Aware and Tensor-Compressed Training of Transformers for Natural Language Understanding

Zi Yang[1]*, Samridhi Choudhary*[2]*, Siegfried Kunzmann*[2]*, Zheng Zhang*[1]

[1] Department of Electrical & Computer Engineering, University of California, Santa Barbara, CA
[2] Amazon Alexa AI

ziy@ucsb.edu, samridhc@amazon.com, kunzman@amazon.com, zhengzhang@ece.ucsb.edu

Abstract

Fine-tuned transformer models have shown superior performances in many natural language tasks. However, the large model size prohibits deploying high-performance transformer models on resource-constrained devices. This paper proposes a quantization-aware tensor-compressed training approach to reduce the model size, arithmetic operations, and ultimately runtime latency of transformer-based models. We compress the embedding and linear layers of transformers into small low-rank tensor cores, which significantly reduces model parameters. A quantization-aware training with learnable scale factors is used to further obtain low-precision representations of the tensor-compressed models. The developed approach can be used for both end-to-end training and distillation-based training. To improve the convergence, a layer-by-layer distillation is applied to distill a quantized and tensor-compressed student model from a pre-trained transformer. The performance is demonstrated in two natural language understanding tasks, showing up to $63\times$ compression ratio, little accuracy loss and remarkable inference and training speedup.

Index Terms: model compression, tensor decomposition, quantization, natural language understanding

1. Introduction

Transformer models [1] have been widely used for natural language understanding (NLU) [2–4] and automatic speech recognition (ASR) [5–7]. Typically, larger pre-trained transformer models perform better on downstream tasks [2,8–10]. However, these large-size models cannot be deployed directly on edge devices due to the limited computing, memory, and energy resources as well as low latency requirement. As a result, model compression has become an indispensable step to enable efficient deployment of large NLU and ASR models on resource-constraint hardware platforms [11–13]. Existing works have studied NLU and ASR model compression via knowledge distillation [14–17], quantization [18–20] and low-rank matrix factorization [21, 22]. Among these approaches, low-rank matrix compression normally achieves much higher compression ratios.

Meanwhile, studies in the applied math community have shown that tensor decomposition [23] often achieves a much higher compression ratio than matrix compression approaches. As a high-dimensional generalization of matrix decompositions, low-rank tensor decomposition has achieved state-of-the-art results in neural network compression [24–27], including both post-training compression and end-to-end compressed training. Recently, tensor decomposition has also been employed to compress transformer models used in natural language modeling [28]. Since many edge devices (e.g., embedded CPU, embedded GPU and FPGA) support low-precision computation, it is natural to ask if low-precision tensor compression can be used to achieve further cost reduction on edge devices. A previous study [29] investigated low-precision training of tensor-compressed models, but it shows that directly applying existing low-precision training in the tensor-compressed setting can cause a remarkable accuracy drop even on a simple two-layer perceptron network.

In this work, we present a quantization-aware and tensor-compressed training approach for transformers. We first use low-rank tensor train (TT) and tensor-train matrix (TTM) formats to represent the embedding tables and linear layers respectively, which achieve significant parameter reduction. To further reduce memory and computing costs, we apply quantization-aware training with learnable scale factors, which enforces the low-rank tensor factors of transformer models into low precision. Our work uses 2-, 4-, or 8-bit fixed-point uniform quantization. The proposed quantization-aware and tensor-compressed training can be used for both end-to-end training and post-training compression. In order to leverage the information of pre-trained transformer models to save the training cost, we further employ layer-by-layer distillation [15] to match the internal outputs and attention probabilities of the original model and our low-precision tensor-compressed model to maintain the generalization capability. This layer-by-layer distillation can avoid the divergence issue of distilling all layers in a tensor-compressed format. We demonstrate the quantization-aware and tensor-compressed training approach on NLU tasks, ATIS dataset [30] and GLUE benchmark [31]. We perform end-to-end training on the ATIS dataset and compress the BERT-base via layer-by-layer distillation on the GLUE benchmark. In both tasks, our approach reaches ultra-low model size with little performance degradation.

2. Methodology

2.1. Tensor-Compressed Transformer Training

A typical transformer model [1] consists of an embedding table and a set of encoder blocks, where each encoder has one self-attention layer and one feed-forward layer. All self-attention and feed-forward layers are composed of linear layers. The embedding table can be regarded as a special type of linear layer. Tensor-compressed transformer compresses the weight matrices of the linear layers into small tensor cores. We directly train the small tensor cores rather than larger weight matrices.

Consider the linear layer $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$, where $\mathbf{x} \in \mathbb{R}^N$ is the input, $\mathbf{W} \in \mathbb{R}^{M\times N}$ is the weight matrix, and $\mathbf{b} \in \mathbb{R}^M$ is the bias vector. The weight $\mathbf{W}$ is reshaped into a tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{m_1\times\cdots\times m_d\times n_1\times\cdots n_d}$, where $\Pi_{i=1}^d m_i = M$ and $\Pi_{i=1}^d n_i = N$. Then we employ either the tensor-train (TT) for-

mat or tensor-train matrix (TTM) format to reduce the number of model parameters.

TT Compression. The TT format represents tensor $\boldsymbol{\mathcal{W}}$ as a set of small-size tensor cores $\boldsymbol{\mathcal{G}}_1, \ldots, \boldsymbol{\mathcal{G}}_{2d}$, where $\boldsymbol{\mathcal{G}}_i \in \mathbb{R}^{r_{i-1} \times m_i \times r_i}$ for $1 \leq i \leq d$ and $\boldsymbol{\mathcal{G}}_i \in \mathbb{R}^{r_{i-1} \times n_{i-d} \times r_i}$ for $d+1 \leq i \leq 2d$. The tensor $\boldsymbol{\mathcal{W}}$ and the tensor cores $\{\boldsymbol{\mathcal{G}}_i\}_{i=1}^{2d}$ satisfy the following equation

$$\boldsymbol{\mathcal{W}}(i_1, \ldots, i_d, j_1, \ldots, j_d) = \mathbf{G}_1^{i_1} \cdots \mathbf{G}_d^{i_d} \mathbf{G}_{d+1}^{j_1} \cdots \mathbf{G}_{2d}^{j_d},$$

where $\mathbf{G}_k^{i_k} := \boldsymbol{\mathcal{G}}_k(:, i_k, :) \in \mathbb{R}^{r_{k-1} \times r_k}$ and $\mathbf{G}_{k+d}^{j_k} := \boldsymbol{\mathcal{G}}_{k+d}(:, j_k, :) \in \mathbb{R}^{r_{k-1+d} \times r_{k+d}}$. The tuple $(r_0, r_1, \ldots, r_{2d})$ is called the TT rank, with $r_0 = r_{2d} = 1$. The TT-compressed linear layer stores the small tensor cores $\{\boldsymbol{\mathcal{G}}_i\}_{i=1}^{2d}$ rather than the large matrix $\mathbf{W}$. After compression, the number of model parameters is reduced to $\sum_{i=1}^{d}(r_{i-1} m_i r_i + r_{i-1+d} n_i r_{i+d})$ from $MN = m_1 \cdots m_d n_1 \cdots n_d$. For fixed ranks, the reduction is roughly $O(m_1 \cdots m_d n_1 \cdots n_d) \rightarrow O(\sum_{i=1}^{d}(m_i + n_i))$. The compression ratio is determined by the TT rank. For the convenience of discussions and experiments, we fix the TT rank before training, but the TT ranks can also be determined automatically in the training process [25]. The matrix-vector multiplication in TT format can be done efficiently with fewer arithmetic operations than standard matrix-vector products [32].

TTM Compression. The TTM decomposition represents tensor $\boldsymbol{\mathcal{W}}$ as d tensor cores $\{\boldsymbol{\mathcal{F}}_i \in \mathbb{R}^{p_{i-1} \times m_i \times n_i \times p_i}\}_{i=1}^{d}$. The tensor cores satisfy

$$\boldsymbol{\mathcal{W}}(i_1, \ldots, i_d, j_1, \ldots, j_d) = \mathbf{F}_1^{i_1, j_1} \cdots \mathbf{F}_d^{i_d, j_d},$$

where $\mathbf{F}_k^{i_k, j_k} := \boldsymbol{\mathcal{F}}_k(:, i_k, j_k, :) \in \mathbb{R}^{p_{i-1} \times p_i}$.

The matrix-vector product using TT format is faster TTM format since the contraction order for TT format is optimized as in [32]. The TTM compression is more suitable for weight matrices with unbalanced rows and columns. In our tensor-compressed transformer, all linear layers in encoder blocks are trained in the TT format for efficient computation, and the embedding table is trained in the TTM format since the number of rows is much larger than the number of columns.

Assume that the weights and embedding tables $\{\mathbf{W}_j\}_{j=1}^{M}$ of a transformer are represented with a set of small tensor cores $\{\boldsymbol{\mathcal{G}}_i\}_{i=1}^{N}$. The training variables in the tensor-compressed model are the tensor cores $\{\boldsymbol{\mathcal{G}}_i\}_{i=1}^{N}$. Suppose that the tensor-compressed model parameterized by the tensor cores is $f(\mathbf{x}|\{\boldsymbol{\mathcal{G}}_i\}_{i=1}^{N})$. The full-precision end-to-end tensor-compressed training is to minimize the model loss:

$$\min_{\{\boldsymbol{\mathcal{G}}_i\}_{i=1}^{N}} \sum_k \text{loss}\left(\text{target}_k, f(\mathbf{x}_k|\{\boldsymbol{\mathcal{G}}_i\}_{i=1}^{N})\right).$$

2.2. End-to-End Quantization-Aware Training

With TT/TTM compression, we further reduce the model size by quantization-aware training with learnable scale factors. The goal is to obtain ultra low-bit representation for all tensor cores used for compressing a transformer model.

Assume that the tensor cores are represented with b-bit quantization $\{Q(\boldsymbol{\mathcal{G}}_i, \delta_i, b)\}_{i=1}^{N}$ to save the computing and memory cost on edge devices. The quantization-aware tensor-compressed training computes the tensor cores $\{\boldsymbol{\mathcal{G}}_i\}_{i=1}^{N}$ and scales $\{\delta_i\}_{i=1}^{N}$ via solving the following optimization problem:

$$\min_{\{\boldsymbol{\mathcal{G}}_i, \delta_i\}_{i=1}^{N}} \sum_k \text{loss}\left(\text{target}_k, f(\mathbf{x}_k|\{Q(\boldsymbol{\mathcal{G}}_i, \delta_i, b)\}_{i=1}^{N})\right).$$

We observe that the tensor cores are well centered around 0, thus a symmetric quantization with scaling is employed. The quantization function Q is defined as

$$Q(x, \delta, b) := \delta \,\text{round}\left(\text{clip}(\frac{x}{\delta}, -2^{b-1}, 2^{b-1} - 1)\right),$$

where $\text{round}(a)$ rounds a to its nearest integer and the function $\text{clip}(a, v_{\min}, v_{\max})$ clips x into the range $[v_{\min}, v_{\max}]$. The quantization $Q(x, b, \delta)$ maps x into $\texttt{INT}_b$. The range of weights and tensor cores may differ dramatically before and after training. Thus, we set the scaling factor δ as a learnable variable that can be automatically determined during training. The quantization function Q is not differentiable, but we can compute the fake gradients to $\boldsymbol{\mathcal{G}}_i$ and δ_i using straight-through estimators. According to [33], the fake gradient of $Q(x, \delta, b)$ with respect to δ and x is

$$\frac{\partial Q(x, \delta, b)}{\partial \delta} := \begin{cases} \frac{Q(x,\delta,b) - x}{\delta} & \text{if } -2^{b-1} \leq \frac{x}{\delta} \leq 2^{b-1} - 1 \\ -2^{b-1} & \text{if } \frac{x}{\delta} < -2^{b-1} \\ 2^{b-1} - 1 & \text{if } \frac{x}{\delta} > 2^{b-1} - 1 \end{cases} \tag{1}$$

$$\frac{\partial Q(x, \delta, b)}{\partial x} := \begin{cases} 1 & \text{if } -2^{b-1} \leq \frac{x}{\delta} \leq 2^{b-1} - 1 \\ 0 & \text{otherwise} \end{cases}. \tag{2}$$

The scaling factors are layer dependent, i.e., different linear layers have different scaling factors. All tensor cores in the same linear layer share the same scaling factor since they typically have a similar range. During quantization-aware training, the inputs of the linear layers are quantized into $\texttt{INT}_8$, and all intermediate tensor contractions are computed in $\texttt{INT}_8$ to reduce computation costs further.

2.3. Layer-by-Layer Distillation of Low-Precision Tensor-Compressed Transformers

Our quantization-aware tensor-compressed training performs very well on end-to-end tasks. However, in some practical NLU tasks, an end-to-end training from scratch can be expensive. Instead, one may want to learn a small-size model from a pre-trained large model. Therefore, this subsection presents a layer-by-layer distillation to learn a low-precision tensor-compressed student model from a fine-tuned teacher model.

Suppose the teacher model is a transformer with an embedding and L encoders. The low-precision tensor-compressed student model has the same model architecture as the teacher model, while the weights are represented by low-precision low-rank tensor cores. Let $\mathbf{y}_{\text{emb}}$ be the output of the embedding table, $\mathbf{y}$ be the predicted soft label, and $\mathbf{y}_i, \mathbf{attn}_i$ be the output and attention probability matrix of the ith encoder block of a transformer. The superscript t and s indicate a teacher model and a student model, respectively. Existing works [16, 18, 19] use the following distillation loss for compression:

$$\begin{aligned} \mathcal{L}_{\text{all}} :=& \text{MSE}(\mathbf{y}_{\text{emb}}^t, \mathbf{y}_{\text{emb}}^s) + \text{COS}(\mathbf{y}_{\text{emb}}^t, \mathbf{y}_{\text{emb}}^s) \\ &+ \sum_{i=1}^{L}(\text{MSE}(\mathbf{y}_i^t, \mathbf{y}_i^s) + \text{COS}(\mathbf{y}_i^t, \mathbf{y}_i^s)) \\ &+ \sum_{i=1}^{L} \text{CE}(\mathbf{attn}_i^t, \mathbf{attn}_i^s)) + \text{CE}(\mathbf{y}^t/T, \mathbf{y}^s/T), \end{aligned} \tag{3}$$

where MSE is the mean squared error, COS is the cosine similarity, CE is the cross entropy loss, and T is the temperature of soft labels. The above distillation loss matches soft labels, the internal outputs, and attention probabilities to increase the generalization property of a student model.

Table 1: *Tensor-compression setting for ATIS dataset.*

	format	linear shape	tensor shape	rank
embedding	TTM	(800,768)	(15,20,16,16,8)	30
attention	TT	(768,768)	(24,32,32,24)	10
feed-forward	TT	(768,3072)	(32,24,48,64)	10
classification	TT	(768,768)	(24,32,32,24)	10

Address the Convergence Issue in Tensor-Compressed Settings. Most existing works [14, 16, 18, 19] reuse pretrained weights in the teacher model to initialize the student model by making the two models similar at the beginning of training. Thus, the distillation loss containing outputs and attention probabilities of all layers performs well. However, the pretrained weight matrices may not have low-rank structures, thus they cannot be directly used to initialize tensor-compressed distillation. In fact, the initial tensor-compressed transformer is very different from the teacher model, causing the distillation loss of all layers to fail in tensor-compressed training. Motivated by this observation, we use the layer-by-layer distillation proposed in [15], which matches the outputs and attention probabilities from top layers to bottom layers. The layer-by-layer distillation starts from the embedding table with loss $\mathcal{L}_0 := \mathrm{MSE}(\mathbf{y}^t_{\mathrm{emb}}, \mathbf{y}^s_{\mathrm{emb}}) + \mathrm{COS}(\mathbf{y}^t_{\mathrm{emb}}, \mathbf{y}^s_{\mathrm{emb}})$. Then, the loss for the ith encoder block is

$$\begin{aligned}\mathcal{L}_i \quad := \quad & \mathcal{L}_{i-1} + \mathrm{MSE}(\mathbf{y}^t_i, \mathbf{y}^s_i) + \mathrm{COS}(\mathbf{y}^t_i, \mathbf{y}^s_i) \\ & + \mathrm{CE}(\mathbf{attn}^t_i, \mathbf{attn}^s_i)).\end{aligned}$$

The loss $\mathcal{L}_i$ aims to match the outputs and attention of the first i encoder blocks. We train the tensor-compressed model using the losses $\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_L$ sequentially. Finally, the soft labels are added to the loss $\mathcal{L}_L$, and the loss becomes $\mathcal{L}_{\mathrm{all}}$ in (3).

3. Experiments

We use two natural language understanding (NLU) benchmarks to test our quantization-aware tensor-compressed training framework. Specifically, we test end-to-end and distillation-based training on the ATIS dataset [30] and the GLUE benchmark [31], respectively. For the GLUE benchmark, we use the fine-tuned BERT [2] model as a teacher model for our quantization-aware and tensor-compressed distillation.

3.1. ATIS Dataset for End-to-End Training

The airline travel information system (ATIS) dataset [30] is an NLU dataset containing utterances related to queries for flight reservations. For each utterance, we need to detect its intent and the slot annotation for each word in the utterance. On this dataset, we perform **end-to-end** quantization-aware tensor-compressed training.

The transformer model for this task has one embedding table, two encoders, and two classification heads, where one head is for intent classification and the other one is for slot filling. We use its full-size and full-precision model as a baseline. We compress the embedding table and the two encoders into quantized tensor cores. The first linear layer of each classification is compressed into full-precision tensor cores. All other layers are kept in the original form. Table 1 lists the compressed tensor shapes and ranks. We use batch size 32 and the Adam optimizer [34] with $\beta_1 = 0.9, \beta_2 = 0.98$, and learning rate 10^{-3}. For each model, we train 40 epochs and report the result in Table 2.

The intent classification task is measured by accuracy, and the slot filling is measured by F1-score. The test results are reported in Table 2. Our full-precision tensor-compressed model reaches $19\times$ size reduction with almost the same performance compared with the full-precision full-size baseline. The $\mathtt{INT}_8$ and $\mathtt{INT}_4$ models perform similarly to the baseline and the $\mathtt{FP}_{32}$ tensor-compressed model, with less than 1% accuracy and F1-score drop. The intent accuracy of the $\mathtt{INT}_2$ model drops marginally. The $\mathtt{INT}_4$ and $\mathtt{INT}_2$ models have almost the same size, because the low-precision tensor cores consumes negligible memory and the uncompressed layers and parameters (e.g., layer normalization and bias vectors) dominate the storage cost. We can conclude that the quantized tensor-compressed transformer can reach around $60\times$compression ratio with less than 2% accuracy drop on this dataset.

Table 2: *Tensor-compressed training of Transformer on ATIS dataset in precisions $\mathtt{INT}_2$, $\mathtt{INT}_4$, $\mathtt{INT}_8$, and $\mathtt{FP}_{32}$*

	intent	slot	size (MB)
Full-size full-precision	95.2	97.0	63 ($1\times$)
Tensor-compressed $\mathtt{FP}_{32}$	96.0	96.2	3.3 ($19\times$)
Tensor-compressed $\mathtt{INT}_8$	95.5	96.1	1.4 ($45\times$)
Tensor-compressed $\mathtt{INT}_4$	94.3	96.2	1.1 ($57\times$)
Tensor-compressed $\mathtt{INT}_2$	93.6	95.0	**1.0** (**63**$\times$)

3.2. GLUE Benchmark for Distillation

The General Language Understanding Evaluation (GLUE) benchmark [31] is a collection of multiple natural language understanding tasks. It is widely used to evaluate the performance of natural language models. Four datasets in GLUE are chosen to test the proposed quantization-aware tensor-compressed **distillation method** described in Section 2.3. Among them, MNLI and QNLI have the largest size, SST-2 is moderate, and MRPC is the smallest. These datasets cover common natural language understanding tasks.

BERT-base is a large model containing one large embedding table and twelve encoders. One classification consisting of two linear layers is attached to the end of BERT. The embedding table, all linear layers in encoders, and the first linear layer in the classification are compressed via quantization-aware tensor-compressed training. We first fine-tune BERT-base on each dataset and use the fine-tuned BERTs as the teacher models for layer-by-layer distillation. Table 3 shows the detailed compression setting. In the experiments, we use batch size 32 and the Adam optimizer [34] with $(\beta_1, \beta_2) = (0.9, 0.98)$. The learning rate is 10^{-3} for the losses $\mathcal{L}_0, \ldots, \mathcal{L}_{12}$ and is 5×10^{-5} for the last loss $\mathcal{L}_{\mathrm{all}}$. For each loss, we run 3, 5, 10, and 20 epochs for MNLI, QNLI, SST-2, and MRPC, respectively.

Test results are reported in Table 4. All tasks are measured by **accuracy**. We test two different ranks 30 and 50. The full precision tensor-compressed training of rank 50 maintains the most performance of BERT-base with only $1\% - 2\%$ accuracy drop on every task. The accuracy slightly drops when decreasing the precision to $\mathtt{INT}_8$ and $\mathtt{INT}_4$ while the compression ratio increases to $17\times$ and $35\times$ from $4\times$. The $\mathtt{INT}_4$ model is only

Table 3: *Tensor-compression setting for BERT-base.*

	format	linear shape	tensor shape
embedding	TTM	(30522,768)	(64,80,80,60)
attention	TT	(768,768)	(24,32,32,24)
feed-forward	TT	(768,3072)	(32,24,48,64)
classification	TT	(768,768)	(24,32,32,24)

Table 4: *Distillation-based tensor-compressed training results on development split of the GLUE benchmark. The $\texttt{INT}_8$ tensor-compressed model has the same number of operations as $\texttt{FP}_{32}$, but those operations are cheap fixed-point operations.*

	precision	size (MB)	FLOPs (G)	MNLI	QNLI	SST-2	MRPC
BERT-base [2]	$\texttt{FP}_{32}$	423 (1×)	20.3 (1×)	83.4	91.2	92.8	87.7
DistilBERT [14]	$\texttt{FP}_{32}$	254 (1.7×)	10.1 (2×)	82.2	89.2	91.3	87.5
BinaryBERT [19]	$\texttt{INT}_1$	16.5 (26×)	3.1 (7×)	84.2	91.5	92.6	85.5
LadaBERT-4 [22]	$\texttt{FP}_{32}$	42 (10 ×)	—	75.8	75.1	84.0	—
Rank 50	$\texttt{FP}_{32}$	99 (4×)	3.8 (5×)	82.1	89.1	90.0	86.5
	$\texttt{INT}_8$	24.3 (17×)	3.8 (5×)	80.7	88.1	89.6	85.8
	$\texttt{INT}_4$	12.1 (35×)	1.9 (11×)	79.7	87.9	89.2	85.5
Rank 30	$\texttt{FP}_{32}$	39 (11×)	1.8 (11×)	80.1	88.1	89.3	85.1
	$\texttt{INT}_8$	9.5 (45×)	1.8 (11×)	78.3	87.2	89.2	85.0
	$\texttt{INT}_4$	**4.8** (**88**×)	**0.9** (**23**×)	77.4	86.9	88.3	84.8

12.1MB, suitable for inference on middle resource-constrained edge devices. All results of rank 30 are slightly worse than rank 50 because of the smaller model size. The rank 30 model in $\texttt{INT}_4$ is only 4.8MB while still having acceptable accuracy. The tiny model is suitable for edge devices with strictly limited memory. The tensor-compressed training can easily adjust the model size by tuning the tensor rank in the model. It makes the quantized tensor-compressed transformer work for a wide range of devices with various resource budgets. In practice, we can also use rank-adaptive training [25] to automatically determine the tensor ranks in both end-to-end training and distillation-based training.

The 4th column of Table 4 shows the estimated computational FLOPs at inference for each model. Here, we only count the FLOPs for matrix-vector/tensor-vector multiplications in encoders to simplify the computation. Other operations, like layer normalization and bias addition, only take a very small amount of computation compared to matrix-vector/tensor-vector multiplications. For $\texttt{FP}_{32}$ operations and quantized operations, FLOPs stand for the number of floating-point operations and fixed-point operations, respectively. We follow [19] to count the quantized operations, i.e., the multiplication between an m-bit number and an n-bit number roughly needs $\frac{mn}{64}$ fixed point operations. The full-precision tensor-compressed model saves $5\times$ and $11\times$ FLOPs for ranks 50 and 30, respectively. The $\texttt{INT}_8$ tensor-compressed model has the same number of operations as $\texttt{FP}_{32}$, but those operations are cheap fixed-point operations. After reducing the precision to $\texttt{INT}_4$, the saving of FLOPs further increases to $11\times$ and $23\times$ for ranks 50 and 30, respectively.

Compared to DistilBERT [14], BinaryBERT [19], and LadaBERT [22], our quantization-aware tensor-compressed approach reaches the highest compression ratio ($88\times$) with little accuracy drop and has more flexibility to handle the trade-off between model performance and model size by tuning tensor ranks and model precisions.

We demonstrate the reduced computation of tensor-compressed training and inference by end-to-end training on the MNLI dataset training split with 393,000 sentences on an RTX-3090 GPU with 24G memory. Table 5 shows that the tensor-compressed training and inference are $1.8\times$ faster than the uncompressed training. The time reduction ratio is less than the FLOPs reduction ratio in Table 4 because some small tensor contractions in tensor-vector multiplication are not parallelized on GPU. We expect the runtime reduction ratio to be similar to the FLOPs reduction ratio after optimizing the parallelization of the tensor contractions.

Table 5: *Inference and training time for one epoch on MNLI training split with batch size 128.*

	inference	training
uncompressed	8.8min	26min
tensor-compressed	**4.7min** (**1.8**×)	**14min** (**1.8**×)

4. Conclusions and Remarks

To compress transformer-based NLU models, we have proposed a quantization-aware and tensor-compressed method for both end-to-end training and distillation-based training. The embedding table and linear layers are compressed into small tensor cores, thereby substantially reducing the total number of model parameters. Besides that, we have applied quantization to each tensor core, further reducing memory costs. Quantization-aware training with trainable scaling factors has been used to learn the quantized tensor cores. To learn a compact NLU and speech recognition model from a pre-trained large transformer model, we have proposed to use layer-by-layer distillation method. This method outperforms the distillation that combines all layer outputs which typically leads to divergence in tensor-compressed training. We have evaluated our quantization-aware tensor-compressed training for two NLU tasks, where our compressed models have achieved high compression ratios with minimal accuracy drop. The quantized tensor-compressed models can have vastly different model sizes for various combinations of tensor ranks and precision. The experiment has demonstrated that our approach could maintain good accuracy even for extremely low ranks and precision. Our method allows additional deployment flexibility on devices with varying resource constraints.

We would like to remark that our method can be applied to all transformer-based models for compression, not only limited to BERT. For instance, our approach has the potential to highly compress the transformer part of wav2vec2 [10], a pre-trained transformer-based model for speech recognition.

5. Acknowledgements

The authors would like to thank Ershad Banijamali, Clement Chung, Athanasios Mouchtaris, and Hieu Nguyen from Amazon for their fruitful suggestions and comments!

6. References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need,"

Advances in neural information processing systems, vol. 30, 2017.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proc. Association for Computational Linguistics: Human Language Technologies*, 2019, pp. 4171–4186.

[3] W. Antoun, F. Baly, and H. Hajj, "AraBERT: transformer-based model for arabic language understanding," in *Proceedings of the 4th Workshop on Open-Source Arabic Corpora and Processing Tools, with a Shared Task on Offensive Language Detection*, 2020, pp. 9–15.

[4] M. H. Radfar, A. Mouchtaris, and S. Kunzmann, "End-to-end neural transformer based spoken language understanding," in *Interspeech*, 2020, pp. 866–870.

[5] N. Moritz, T. Hori, and J. Le, "Streaming automatic speech recognition with the transformer model," in *Proc. Intl. Conf. Acoustics, Speech and Signal Processing*, 2020, pp. 6074–6078.

[6] S. Zhang, E. Loweimi, P. Bell, and S. Renals, "On the usefulness of self-attention for automatic speech recognition with transformers," in *IEEE Spoken Language Technology Workshop (SLT)*, 2021, pp. 89–96.

[7] S. Kim, A. Gholami, A. E. Shaw, N. Lee, K. Mangalam, J. Malik, M. W. Mahoney, and K. Keutzer, "Squeezeformer: An efficient transformer for automatic speech recognition," in *Advances in Neural Information Processing Systems*, 2022.

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

[9] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *Advances in neural information processing systems*, vol. 32, 2019.

[10] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," *Advances in neural information processing systems*, vol. 33, pp. 12 449–12 460, 2020.

[11] K. Mysore Sathyendra, S. Choudhary, and L. Nicolich-Henkin, "Extreme model compression for on-device natural language understanding," in *Proc. Intl. Conf. Computational Linguistics: Industry Track*, Dec. 2020, pp. 160–171.

[12] A. Saade, A. Coucke, A. Caulier, J. Dureau, A. Ball, T. Bluche, D. Leroy, C. Doumouro, T. Gisselbrecht, F. Caltagirone *et al.*, "Spoken language understanding on the edge," in *Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*. IEEE, 2019, pp. 57–61.

[13] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "MobileBERT: a compact task-agnostic BERT for resource-limited devices," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 2158–2170.

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

[15] G. Aguilar, Y. Ling, Y. Zhang, B. Yao, X. Fan, and C. Guo, "Knowledge distillation from internal representations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 7350–7357.

[16] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "TinyBERT: Distilling BERT for natural language understanding," in *Findings of the Association for Computational Linguistics: EMNLP*, 2020, pp. 4163–4174.

[17] L. Hou, Z. Huang, L. Shang, X. Jiang, X. Chen, and Q. Liu, "Dynabert: Dynamic bert with adaptive width and depth," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 9782–9793.

[18] W. Zhang, L. Hou, Y. Yin, L. Shang, X. Chen, X. Jiang, and Q. Liu, "Ternarybert: Distillation-aware ultra-low bit bert," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 509–521.

[19] H. Bai, W. Zhang, L. Hou, L. Shang, J. Jin, X. Jiang, Q. Liu, M. Lyu, and I. King, "Binarybert: Pushing the limit of bert quantization," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2020, pp. 4334–4348.

[20] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Q-bert: Hessian based ultra low precision quantization of bert," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 8815–8821.

[21] H. Saghir, S. Choudhary, S. Eghbali, and C. Chung, "Factorization-aware training of transformers for natural language understanding on the edge," in *Interspeech*, 2021.

[22] Y. Mao, Y. Wang, C. Wu, C. Zhang, Y. Wang, Q. Zhang, Y. Yang, Y. Tong, and J. Bai, "LadaBERT: Lightweight adaptation of BERT through hybrid model compression," in *Proceedings of the 28th International Conference on Computational Linguistics*, 2020, pp. 3225–3234.

[23] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, Aug. 2009.

[24] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Advances in Neural Information Processing Systems 28*, 2015, pp. 442–450.

[25] C. Hawkins, X. Liu, and Z. Zhang, "Towards compact neural networks via end-to-end training: A bayesian tensor approach with automatic rank determination," *SIAM Journal on Mathematics of Data Science*, vol. 4, no. 1, pp. 46–71, 2022.

[26] C. Hawkins and Z. Zhang, "Bayesian tensorized neural networks with automatic rank selection," *Neurocomputing*, vol. 453, pp. 172–180, 2021.

[27] A. Tjandra, S. Sakti, and S. Nakamura, "Compressing recurrent neural network with tensor train," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 4451–4458.

[28] X. Ma, P. Zhang, S. Zhang, N. Duan, Y. Hou, M. Zhou, and D. Song, "A tensorized transformer for language modeling," *Advances in neural information processing systems*, vol. 32, 2019.

[29] K. Zhang, C. Hawkins, X. Zhang, C. Hao, and Z. Zhang, "On-fpga training with ultra memory reduction: A low-precision tensor method," in *ICLR Workshop on Hardware Aware Efficient Training*, 2021.

[30] C. T. Hemphill, J. J. Godfrey, and G. R. Doddington, "The atis spoken language systems pilot corpus," in *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27*, 1990.

[31] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, 2018, pp. 353–355.

[32] Z. Liu, X. Yu, and Z. Zhang, "TT-PINN: a tensor-compressed neural PDE solver for edge computing," *arXiv preprint arXiv:2207.01751*, 2022.

[33] S. Jain, A. Gural, M. Wu, and C. Dick, "Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks," in *Proceedings of Machine Learning and Systems*, vol. 2, 2020, pp. 112–128.

[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2014.