Deep Compression of Pre-trained Transformer Models

Naigang Wang **Chi-Chun Liu** **Swagath Venkataramani** **Sanchari Sen**

Chia-Yu Chen **Kaoutar El Maghraoui** **Vijayalakshmi Srinivasan**

Leland Chang
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598, USA
{nwang,cliu,swagath.venkataramani,sanchari.sen,
cchen,kelmaghr,viji,lelandc}@us.ibm.com

Abstract

Pre-trained transformer models have achieved remarkable success in natural language processing (NLP) and have recently become competitive alternatives to Convolution Neural Networks (CNN) and Recurrent Neural Networks (RNN) in vision and speech tasks, respectively. Due to their excellent computational efficiency and scalability, transformer models can be trained on exceedingly large amounts of data at the expense of tremendous growth in model size. As high performance, large-scale, and pre-trained transformer models become increasingly available for users to download and fine-tune for customized downstream tasks, their deployment becomes challenging due to the vast amount of operations and large memory footprint. To address this challenge, we introduce methods to deeply compress pre-trained transformer models across three major application domains: NLP, speech, and vision. Specifically, we quantize transformer backbones down to 4-bit and further achieve 50% fine-grained structural sparsity on pre-trained BERT, Wav2vec2.0, and Vision Transformer (ViT) models to demonstrate 16x compression while maintaining model accuracy. This is achieved by identifying critical initialization strategies for quantization- and sparsity- aware fine-tuning as well as developing novel techniques such as quantizers with a zero-preserving format and scheduled dropout. These hardware-friendly techniques need only to be applied in the fine-tuning phase for downstream tasks, which renders them especially suitable for acceleration and deployment of pre-trained transformer models.

1 Introduction

Since its inception in 2017, the attention-based transformer architecture [1] has excelled in many Natural Language Processing (NLP) tasks and become the preferred building block for state-of-the-art language models. With their excellent capacity for transfer learning, transformer models that are pre-trained on large corpora, such as BERT [2]) and GPT-3 [3], can be fine-tuned for downstream tasks to achieve state-of-the-art performance. Today, fine-tuning a transformer-based pre-trained language model has become the de facto standard for NLP tasks. Inspired by this success in NLP, transformer models are also being actively explored in other application domains and making tremendous progress. For vision applications, transformer models have demonstrated excellent performance — matching or exceeding CNN-based models on image classification [4, 5] and object detection [6] tasks. For speech applications, models with a transformer backbone such as Wav2vec2.0 [7], HuBERT [8], and

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

WavLM [9] show comparable performance to traditional RNN/LSTM-based models, but with much improved computation efficiency and scalability.

Pre-trained transformer models used as a 'Swiss Army knife' for a wide range of machine learning tasks are becoming so-called "Foundation Models", which can potentially change the landscape of deep learning research and development [10]. Today, transformer models are pre-trained on increasingly large amounts of data, which has grown model sizes to unprecedented levels. In the past few years, many high performance, large-scale models with millions to billions parameters have been released for users to download and fine-tune for customized downstream tasks [11]. However, the vast number of computation operations and large memory footprint are becoming key barriers to their practical deployment and adoption in production settings. To address this challenge, techniques such as knowledge distillation [12–16] have been used to pre-train compact transformer-like models. However, standard transformer topologies remain the preferred architecture to achieve high performance on a wide range of tasks.

To compress deep neural networks, quantization has proven to be effective in many application domains and is a common technique to accelerate DNNs on GPU and other AI hardware platforms [17]. Thus far, quantization of transformer models has primarily been studied for NLP applications. 8-bit quantization was first applied to BERT in [18, 19] and showed negligible model accuracy degradation. Later, weight precision was pushed to 4-bit or lower [20, 21], but activations remained in 8-bit due to sensitivity to quantization errors. Recently, knowledge distillation has been shown to be effective in quantizing BERT models down to 4-bit for both weights and activations while maintaining baseline accuracy. Vision transformers (ViT) have been successfully quantized down to 6-bit [22] using post-training quantization (PTQ) techniques. For speech, the authors in [23] quantized a simplified transformer down to 8-bit. To date, vision and speech transformer models have not yet been successfully quantized down to precisions as low as 4-bit while NLP transformer models have only reached this level with complex distillation techniques.

Pruning is another promising method to compress transformer models — motivated by the assumption that the rich features learned during the pre-training phase may be redundant when fine-tuning for downstream tasks and can thus be pruned. In the past few years, pruning has been extensively explored on transformer models for NLP [24], ViT [25], and Speech [26] to achieve 40-50% sparsity without significant accuracy loss. However, most such work focuses on unstructured sparsity, which is challenging to harness in modern efficient hardware using vector- and matrix-based instructions. Hardware vendors are instead introducing techniques such as fine-grained structured sparsity [27], which can achieve hardware efficiency improvements across a wide range of DNN models.

Applying both quantization and pruning techniques to the same model is a particular challenge — due primarily to difficulties in coordinating two separate lossy mechanisms to optimize and recover model accuracy [28] [29]. To our knowledge, [27] is the only work to explore the combination of quantization and sparsity on pre-trained transformer models such as BERT. In those results, for 50% sparsity, INT8 is the lowest precision that achieves reasonable accuracy. To recover model accuracy in the sparse INT8 model, pre-training must be repeated with sparsity before fine-tuning for downstream tasks — a requirement that becomes extremely challenging due to access limitations to pre-training datasets and the significant compute resources needed for today's large-scale models [3].

In this paper, we combine 4-bit quantization and 50% pruning to deeply compress transformer models across multiple application domains, i.e. BERT, Wav2vec2.0, and Vision Transformer (ViT) models. Specifically, transformer backbones are quantized down to 4-bit and combined with 50% fine-grained structural sparsity to achieve up to 16x model compression while maintaining model accuracy. Our method does not require repeating pre-training nor accessing pre-training datasets. Instead, quantization and pruning are performed during the fine-tuning phase on downstream tasks using downloaded pre-trained models. Our techniques are straightforward to implement without the need for complex methods such as knowledge distillation, which may require large memory and compute resource. Our precision format and sparsity configuration are hardware-friendly and can be efficiently implemented in modern deep learning accelerators. Our methodology generalizes across different application domains, which brings an opportunity to greatly simplify the software and hardware-based acceleration and deployment of pre-trained transformer models.

In summary, we make the following contributions towards accurate, sparse, 4-bit transformer models across application domains:

1. We report, for the first time, transformer backbones quantized down to 4-bit precision with 50% fine-grained structural sparsity on pre-trained BERT, Wav2vec2.0, and ViT models. We achieve 1.62%, 0.45% and 0.52% degradation for the question-answering task on SQuAD1.1, the speech recognition task on Librispeech, and the image classification task on ImageNet1k, respectively. Without sparsity, all 4-bit models achieve $< 1\%$ accuracy loss with respect to the baseline.
2. We propose procedures for quantization-aware and sparsity-aware fine-tuning to minimize the impact of these two lossy processes. In particular, we show that proper initialization of the quantized model, quantization scales, and sparse model are critical to minimize accuracy losses.
3. We introduce novel techniques, including a SAWB+ weight quantizer with a symmetric, zero-preserving quantization format and scheduled dropout, to enable the deep compression.

2 Quantization-aware and Sparsity-Aware Fine-Tuning

Performing quantization/sparsity-aware training during the pre-training phase on large amounts of data may offer the best opportunity to optimize and recover model accuracy for extremely compressed transformer models [27]. However, in practice, the pre-training phase is not generally transparent to end users as it is often difficult to access pre-training data and repeat the pre-training phase, which may require hundreds of GPUs and very long training times [11]. On the other hand, the fine-tuning phase for downstream tasks is often lightweight, during which users can directly operate on their own datasets and tune models for only a few epochs [11]. In this work, we thus focus on quantization/sparsity-aware fine-tuning and introduce techniques in this context to enable deep compression of pre-trained transformer models.

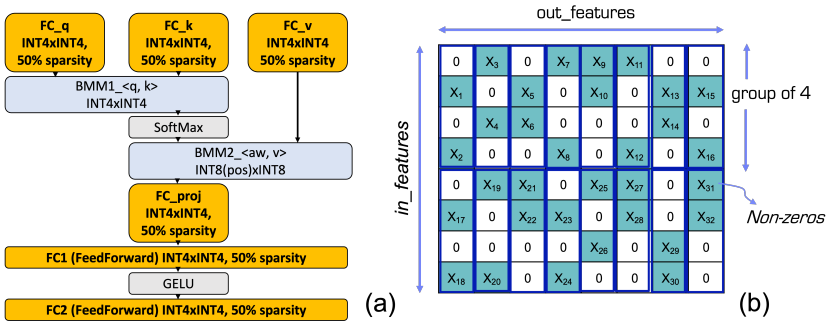

Figure 1: a) The transformer encoder block consisting of FC modules and BMM modules. Gray modules are linear layers where pruning is applied as shown in b), which is a schematic of a fine-grain structurally pruned weight matrix.

Figure 1 shows operations that are quantized and sparsified in a transformer block. Six linear layers, including QKV layers, a projection layer and two feed-forward layers are quantized to 4-bit and their weights are pruned by 50%. All input activations of two batch matrix-matrix product (BMM) layers are quantized. To preserve the fidelity of the probability distribution, we keep BMM2 in 8-bit.

2.1 Quantization

2.1.1 Weight Quantization

SAWB+ is an enhancement of the SAWB (statistics-aware weight binning) quantizer introduced by [30]. SAWB exploits both the first and second moments of the weight distribution to minimize the quantization error. The optimal quantization scale, α_w is calculated by the equation:

$$\alpha_w = c1 * E(w^2) + c2 * E(|w|) \tag{1}$$

where c1 and c2 are coefficients empirically determined offline by fitting six standard distributions at a given precision. The details of extracting c1 and c2 can be found in [31] and the coefficients used in this paper are listed in Appendix-B. SAWB can effectively capture weight distributions to achieve excellent performance on a range of DNN models; however, in the backward path, SAWB can cause instability. As shown in the left of equation 2, where W and W_q represent weight and

Table 1: The impact of zero-alignment on the performance of three sparse INT4 models.

Zero alignment	BERT-base (F1 %)	Wav2vec2.0 (WER[1] %)	ViT (Accuracy%)
Yes	87.07	4.53	83.60
NO	86.06	5.76	83.25

quantized weight respectively, the clamped weights see zero gradients during optimization, which can cause two potential problems: 1) the variance of the weight gradients is reduced by truncation, which slows learning in non-quantized optimization, and 2) larger weights clamped early in training may never be updated, which reduces the representation power of the weight space during training. To overcome this problem, we introduce a simple modification to allow the gradients of clipped weights to pass through during weight updates while keeping the forward pass the same. This modified technique, SAWB+, captures the statistical characteristics of the distribution of weights just as in SAWB, but also has the ability to search the entire weight space during optimization with the same gradient variance per iteration step. In this work, we use SAWB+ for the weight quantization for all transformer models and observe no degradation from weight quantization.

$$\text{SAWB: } \partial W = \begin{cases} \partial W q & |W| \leq \alpha \\ 0 & |W| \geq \alpha \end{cases} \quad \text{SAWB+: } \partial W = \begin{cases} \partial W q & |W| \leq \alpha \\ \partial W q & |W| \geq \alpha \end{cases} \tag{2}$$

Zero alignment is another critical technique to enable pruning at low precision. For integer weights, instead of utilizing the full range of precision levels, i.e. 2^k , we choose to lose one level while keeping the distribution symmetric about zero, resulting in $2^k - 1$ levels. For example, in INT4, the number of integer levels is 15, i.e. [-7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7]. This way, floating point zeros in input will be represented by integer zeros. This zero alignment has significant impact on tuning the sparse model. As shown in Fig. 2, with zero-aligned weights, the BERT-base model converges to a much lower training loss and higher evaluation score in the fine tuning process as compared with the non-aligned case. The intuition is that with zeros in the weights as existing masks, the zero-aligned model requires less effort to learn the rest of the masks to achieve the same amount of sparsity. The same improvement is also observed in Wav2vec2.0 and ViT models, as shown in Table 1. Thus, just through zero alignment, the accuracy of three models can be boosted by 1.01%, 1.23% and 0.35%, respectively. Furthermore, symmetric and uniformly distributed quantization bins with zero alignment are also hardware friendly - allowing for simplified multiplication and accumulation (MAC) design and efficient data-flow [32].

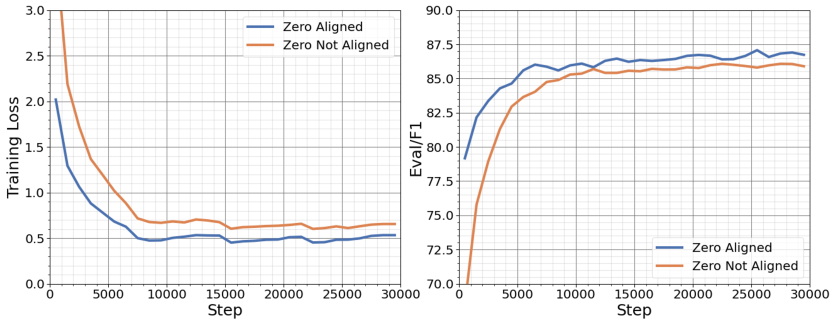

Figure 2: The impact of zero alignment on the convergence of sparse INT4 Bert-base on SQuAD1.1.

2.1.2 Activation Quantization

For activation quantization, we use two well-known quantizers: MinMax and PACT [31]. We extend the original 1-sided PACT to a 2-sided PACT to quantize activations with both positive and negative

[1]WER: Word error rate, lower the better.

values. α and α_n are used to define the dynamic range and the clamped activation output is uniformly quantized to k bits for the dot-product computation.

For the MinMax quantizer, α and α_n take the min and max of the tensors. For PACT, both clipping levels are parameterized and dynamically adjusted via gradient descent-based training. When using PACT, we sample 10 batches of the downstream dataset and initialize α and α_n of each layer based on a given percentile (e.g. 99% and 1%) of the observed activation distribution [30]. Note that we use the same quantizer for both symmetric and asymmetric activation functions such as GeLU. We also preserve activations of zero value by aligning zero to an integer level. Unlike for weights, the full 2^k levels are utilized for activation quantization.

While experimenting with different quantizers, we observed that for all INT8 models, the MinMax quantizer out-performed the PACT quantizer, as shown in Table 2. This observation comes as a surprise, since ideally, learned αs should match the performance of the special MinMax case. As shown in Fig. 3, even when α is initialized to 99.9 or 99.95 percentile, training curves are noisier than when initializing to the max of the sampled activations. Further investigation led us to conclude that a small amount of large-amplitude outliers play an important role in transformer training compared to CNN-based models [33].

INT4 models are very limited in the number of precision levels available and hence become more sensitive to quantization errors as compared to INT8. For this reason, we use the MinMax quantizer for INT8 and sparse INT8 models while using PACT for INT4 models. The details of α initialization conditions are listed in Appendix-B.

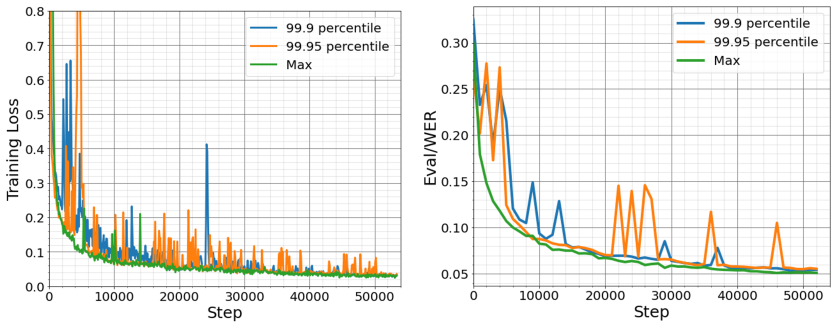

Figure 3: The impact of clipping α initialization on training loss and WER of sparse INT4 Wav2vec2.0 on Librispeech

2.2 Fine-Grained Structured Pruning

Figure 1b depicts the details of the fine-grained structured pruning performed in this work, which is based on the 2:4 sparsity proposed in [27]. We form groups of 4 consecutive weights values along the "in_features" dimension of weights and zero out 2 values to achieve 50% sparsity. The location of the two zero values can vary across the different groups in a layer. The pruned weights can be compressed by storing only the non-zero weight values and their corresponding indices within each group — 2 non-zero values and 2 indices per group. Since indices can only take one of four possible values for a group size of 4, 2 bits are sufficient to represent each index within a group.

We use a mask for weight pruning. The weight pruning mask is extracted only once before the start of the training instead of, e.g. Iterative Magnitude Pruning (IMP) that can be otherwise adopted [28] [29]. At each iteration, the weights are quantized first and the pruning mask is applied on the quantized weights and then fed into matrix computation.

2.3 Quantization- and Sparsity-Aware Fine-Tuning

A key challenge in combining quantization and sparsity is to determine the workflow by which to apply these two lossy processes during fine-tuning, e.g. tuning with quantization followed by pruning, or vice versa. We discover that quantization and sparsity can be jointly fine-tuned in one

Table 2: MinMax vs. PACT quantizers for INT8 models.

Quantizer	BERT-base (F1 %)	Wav2vec2.0 (WER %)	ViT (Accuracy%)
MinMax	88.35	3.85	84.47
PACT	87.89	4.05	83.98

optimization process provided that the process is properly initialized. In this section, we explain the key initialization methods for the compressed model to achieve high accuracy.

2.3.1 Initialization for Quantization

Pre-trained models are generally trained on large datasets in full FP32 precision. The rich features learned during pre-training should provide opportunity for models to accommodate quantization error during the fine-tuning phase. However, when we use the pre-trained model as initialization for quantization-aware fine-tuning, the resulting quantized models consistently show inferior performance as shown in Table 3. For INT4 even without sparsity, the three models studied suffer significant degradation of 2.01%, 0.82%, 1.37%. Even in INT8, the ViT model incurs >1% degradation.

Instead, we can use full precision fine-tuned models to initialize quantization-aware fine-tuning. As shown in Table 3, this significantly boosts the performance of the quantized model both in INT8 and INT4 precisions. Particularly for INT4, all three models achieve accuracy within $< 1\%$ of the baseline. To the best of our knowledge, this is the first time that INT4 transformer models (both weight and activation) achieve iso-accuracy without the teacher-student distillation technique, which requires computation of the full precision teacher models during tuning rather than just using them for initialization. The results also indicate that initialization with pre-trained models is more vulnerable to quantization noise despite redundant features, whereas initialization with the full precision fine-tuned models effectively learn the features most relevant to downstream tasks, which improves the resultant quantized models. When combined with the quantizers described in Section 2.1, quantized model accuracy is significantly improved.

2.3.2 Initialization for Pruning

When adding sparsity, for INT8 precision models, the full precision fine-tuned models are effective as initialization, as shown in table 3. Both the Wav2vec2.0 and ViT models can reach baseline accuracy while the Bert-base model is within 1% of the baseline. Note that we achieve the same accuracy on a sparse INT8 BERT model as reported in [27], but without the need to repeat pre-training.

For sparse INT4 models, however, it becomes difficult to maintain accuracy with this same initialization. As shown in Table 3, BERT and ViT models, in particular, suffer significant degradation $> 2\%$. To overcome this, we propose a different initialization strategy, in which we use a pre-tuned sparse INT8 model as initialization to fine-tune the sparse INT4 model. This is inspired by the observation that INT8 models show no degradation when using MinMax quantizers. Intuitively, a pre-tuned sparse INT8 model is a sparse architecture that can already tolerate low precision, which makes it a strong starting point to further adjust to the larger quantization errors associated with INT4. As shown in Table 3, this initialization method boosts the accuracy significantly and brings Wav2vec2.0 and ViT sparse INT4 models to within 1% of baseline accuracy. Fig. 4 shows the convergence curves for the fine-tuning of sparse INT4 ViT models initialized using FP32 and sparse INT8 fine-tuned models. The training loss starts at a much lower level when using sparse INT8 initialization and the network converges to lower loss and higher accuracy across the entire tuning process.

2.3.3 Scheduled Dropout

For the BERT model, sparse INT8 initialization starts at lower training loss just as in Wav2vec2.0 and ViT, but it does not show immediate improvement in the F1 score due to over-fitting potentially caused by the small size of the SQuAD1.1 dataset. To effectively take advantage of sparse INT8 initialization and avoid over-fitting in the early stages of training, we propose scheduled dropout. Scheduled dropout initially applies a very high dropout rate, e.g. 0.35, and then linearly decreases the rate until the 10k-th iteration, where it finally stays constant, e.g. at 0.2, until the end of the fine-tuning

Table 3: The impact of initialization model on the accuracy of INT8 and INT4 models. FP32 means fine-tuned FP32.

Precision	Initialization	BERT-base (F1 %)	Wav2vec2.0 (WER %)	ViT (Accuracy %)
INT8	Pre-trained	87.95	3.79	83.05
	FP32	88.35	3.85	84.47
INT4	Pre-trained	86.68	5.02	82.75
	FP32	87.86	4.56	83.49
INT8 + 50% sparsity	FP32	87.70	4.21	84.03
INT4 + 50% sparsity	FP32	86.75	5.09	82.66
	Sparse INT8	87.07	4.65	83.60

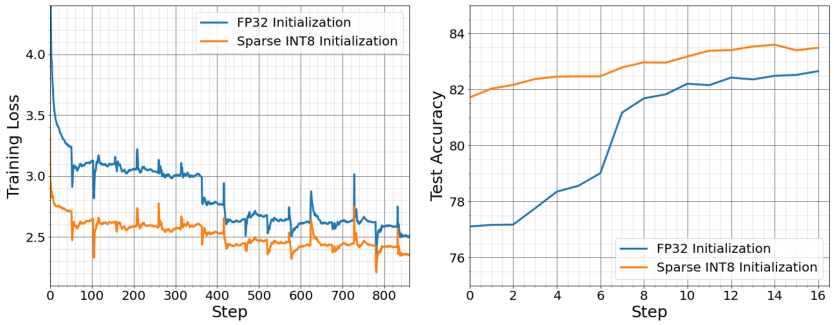

Figure 4: Convergence of sparse INT4 ViT on ImageNet1k using sparse INT8 vs. FP32 initialization.

phase. As shown in Fig. 5, scheduled dropout improves generalization from the beginning of the training, which enables 4-bit models to get closer to baseline F1 scores.

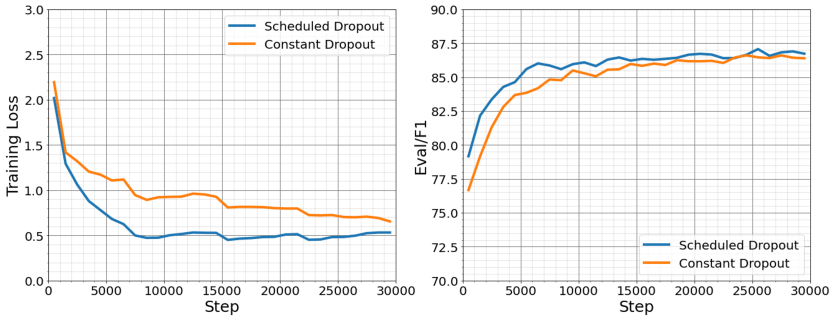

Figure 5: Scheduled vs. constant dropout on the convergence of sparse INT4 BERT on SQuAD1.1

2.4 Hardware Implementation

To evaluate the hardware benefits of quantized and pruned transformers, we consider a systolic-array-based AI hardware accelerator similar to [34] [35]. We enhance the accelerator to further exploit fine-grained structured weight sparsity by skipping computations on zero-valued weights. Specifically, the systolic array is enhanced to read only the non-zero weights in a group and perform a multiply-accumulate operation with the activation value corresponding to the non-zero index.

The primary hardware benefit of sparse transformers is the memory capacity and bandwidth savings obtained from storing their weights in a compressed manner (Section 2.2). These savings are a direct function of the amount of weight sparsity and the precision of weights. Accordingly, for INT8

precision with 50% sparsity, we achieve a memory capacity savings of 1.6× with respect to a dense INT8 baseline and for INT4 precision with 50% sparsity, we achieve a memory capacity savings of 1.33× with respect to a dense INT4 baseline, across all benchmarks. Overall, when compared to a dense FP16 baseline, a sparse INT8 transformer and a sparse INT4 transformer achieves 6.4× and 10.67× savings in memory capacity, respectively. In addition to the memory capacity savings, we also get performance benefits from only performing computations on the non-zero weight values. The overall performance benefits observed for a benchmark depends on the size of different layers in the transformer block and their execution time fraction. By pruning the 6 layers inside each transformer block shown in Figure 1, we affect their execution times while keeping the execution time for all other layers unchanged. We observe that the 6 pruned layers account for 57.7%, 52.4% and 51.1% of the overall execution time for BERT-base, Wav2vec2.0, and ViT respectively in their dense INT8 execution. In dense INT4 execution, these 6 layers account for 40.5%, 35.5% and 34.3% of the overall execution time for BERT-base, Wav2vec2.0 and ViT respectively. Accordingly, we achieve a performance improvement of 1.4×, 1.35× and 1.34× for BERT-base, Wav2vec2.0 and ViT respectively in INT8 with respect to dense models and a performance improvement 1.25×, 1.22× and 1.21× in INT4. When compared to a dense FP16 baseline, the performance improvements of sparse INT8 and INT4 further increase by a factor of 2.76× and 3.78× on an average across all benchmarks.

3 Experimental Setup and Results

We evaluate the proposed methods on three representative pre-trained models and corresponding downstream benchmarks in multiple application domains to demonstrate the effectiveness of our methods. Specifically, we investigate the BERT-base model on the SQuAD1.1 benchmark, the Wav2vec2.0 model on the Librispeech dataset and the ViT-base model on the ImageNet1k benchmark. We implement quantization and pruning on top of Huggingface Transformer [36] for BERT and Wav2vec2.0 models, and Timm packages [37] for the ViT model. We use the recipe published in the original papers to fine-tune the downstream benchmarks and use the results as the baseline to evaluate our method. As shown in Table 4, our baseline matches the published results in the corresponding paper or git repository. All experiments are performed on NVIDIA V100 GPUs. Listed below is basic information on the models and benchmarks — more details are presented in Appendix-A.

Pre-trained BERT-base model on the SQuAD1.1: The BERT-base model has 12 transformer blocks with 12 self-attention heads and a hidden dimension of 768. We use the pre-trained BERT model provided by Huggingface Transformer for fine-tuning SQuAD 1.1, a standard question-answering task for NLP. For fine-tuning, we use batch size of 12 and sequence length of 384. We use the AdamW optimizer with a learning rate of 3e-5 with linear decay. The model is fine-tuned for 2-4 epochs with a dropout probability of 0.1-0.2. For the sparse INT4 model, scheduled dropout is applied as introduced in section 2.3.3.

Wav2vec2.0 large model on the Librispeech: The Wav2vec2.0-large model contains 24 transformer blocks with 16 attention heads and a hidden dimension of 1024 [7]. We use the facebook/wav2vec2-large-lv60 pretrained model on HuggingFace Transformer, which is pre-trained on the audio data from LibriVox. The pretrained model is fine-tuned on Librispeech's 100 hour clean subset using standard Connectionist Temporal Classification (CTC) loss for the Automatic Speech Recognition (ASR) downstream task. The model has a varied token length averaging 600-800. For fine tuning, we use the AdamW optimizer with a learning rate of 3e-4. The learning rate decays linearly after 500 warm-up steps. We use a batch size of 32 and tune the model for 6 epochs.

ViT-base model on ImageNet1k: The ViT-base model contains 12 transformer blocks with 12 attention heads and a hidden dimension of 768 [4]. To clearly understand the impact of quantization/pruning on transfer learning, we use the pre-trained model that is only trained on ImageNet21k and then fine-tune it on ImageNet1k for downstream image classification. For fine-tuning, we use a patch size of 16x16 and fine-tune resolution of 384x384, following the original settings of [4]. The optimizer is SGD with a learning rate of 0.01. We tune the model for 8 epochs using a cosine learning rate schedule, gradient clipping of 1.0, and batch size of 512. For sparse INT8/4 models, we find that it is beneficial to stimulate training by increasing the learning rate to 0.05 and tune for 16 epochs until convergence settles.

Quantization and pruning are applied only on transformer backbones as they contribute $> 99\%$ of the total compute operations. We use a common setting for all three models and benchmarks.

Table 4: Results for deeply compressed BERT-base, Wav2vec2.0, and ViT models. FP32 refers to FP32 fine-tuned on downstream tasks. Sp is short for sparsity.

Precision Sparsity	Weight Quantizer	Activation Quantizer	Initialization Model	BERT-base (F1%)	Wav2vec2.0 (WER %)	ViT (Accuracy %)
FP32	–	–	Pre-trained	88.69	4.20	84.12
INT8	SAWB+	MinMax	FP32	88.35 (-0.34)	3.85 (+0.35)	82.47 (+0.35)
INT8+50%Sp	SAWB+	MinMax	FP32	87.70 (-0.99)	4.21 (-0.01)	84.03 (-0.09)
INT4	SAWB+	PACT	FP32	87.86 (-0.83)	4.53 (-0.33)	83.49 (-0.63)
INT4+50%Sp	SAWB+	PACT	INT8+50%Sp	87.07 (-1.62)	4.65 (-0.45)	83.60 (-0.52)

Specifically, the SAWB+ quantizer is used for weight quantization. The MinMax quantizer is used for the activation quantization for INT8 and sparse INT8 models, and the PACT quantizer is used for INT4 and sparse INT4 models. Full precision fine-tuned models are used for initialization of INT8, sparse INT8 and INT4 models. For the sparse INT4 model, we use a sparse INT8 model for initialization as discussed in section 2.3.2. More details of the training methodology are presented in Appendix-B. Table 4 shows the final results on the three benchmarks with different precision/sparsity settings. The uncertainty analysis of the results is presented in Appendix-B, where table 7 As shown, aside from the sparse INT4 BERT-base model, which shows a small degradation of 1.6%, all other compressed models achieve iso-accuracy within $< 1\%$ of their respective baselines, making them good candidates for efficient deployment.

4 Related Work

Quantization has been extensively studied to compress CNN and RNN models. Over the years, many approaches, such as quantization-aware training [31], post-training-quantizaiton [22], knowledge distillation [38], and quantizers, such as PACT [31], LSQ [39], LSQ+ [40], and SAWB [30], have been developed to improve the performance of quantized models. For NLP, these techniques have been applied to compress transformer models. FullyQT [18] and [41] developed an 8-bit transformer for machine translation. Q8BERT [19] and IBERT [42] quantize BERT and RoBERTa [43] models respectively in 8-bit. Later studies focus on lower precision weight quantization. QBERT [20] quantizes weights down to 4-bit without significant degradation using Hession information. TernaryBERT, BinaryBERT and BiBERT [44–46] pushed weight precision to even lower than 4-bit, while maintaining activations in 8-bit. Recently, KDLSQ-BERT [47] and MKQ-BERT [48] discovered that distillation techniques can be used to effectively quantize activations in the BERT model into 4-bit. Our methods, however, are more straightforward, easier to implement, and can achieve good performance while avoiding some of the limitations of distillation such as the requirement of storing and computing the teacher models. More importantly, our methods and distillation are mutual exclusive and can be combined to further improve the performance of compressed models.

There has been less work in quantizing vision and speech transformers, however, Bie et. al [23] quantize a transformer-based encoder-decoder architecture to 8-bit for the end-to-end ASR task, but the transformer is simplified with only 6 encoder/decoder layers and quantization is only applied to weights. Liu et al. [22] and FQ-ViT [22] quantize vision transformer models such as ViT [49], DeiT [38] and DETR [50] and achieve iso-accuracy at 8-bit precision, but noticeable degradation is observed when the precision is reduced down to 6-bit.

Pruning has been extensively applied to compress transformer models in NLP [51–53], ViT [54–56], and Speech [26]. However, most pruning approaches are unstructured, which can be challenging for hardware-based acceleration. Structured pruning in the form of directly removing attention heads [57, 58] shows promising results, but these methods may not be applicable to transformers in other application domains [55]. To the best of our knowledge, there is no common structural pruning approach that can be broadly applied.

The most relevant work is [27], which explores the combination of quantization and sparsity on transformer models and is the first to propose fine-grained structural pruning. However, the work is only verified on pre-trained BERT models, the lowest precision reached is 8-bit, and the sparse model must repeat pre-training to recover model accuracy. Using our approach, we achieve the same

accuracy while only impacting the fine-tuning phase. We also successfully push precision down to 4-bit and demonstrate our approach across three representative application domains.

5 Conclusion

Pre-trained transformer models have demonstrated state-of-the-art performance in a wide range of applications, but tremendous growth in computational requirements and memory footprint render deployment in practical applications a significant challenge. To address this, we introduce a novel quantization/sparsity-aware fine-tuning methodology to successfully compress pre-trained transformer models by carefully choosing suitable quantizers, data format, initialization, and regularization techniques. Our methods are demonstrated on popular pre-trained transformer models and benchmarks across three application domains. This work is critical as it lays the foundation to deploy large pre-trained transformer models in practical end-user applications including data centers and edge scenarios. Our inference solutions can accelerate ML deployment and provide significant cost and energy savings for AI inference. Our solutions could still be subject to unexpected instabilities, and going forward, will require task-specific robustness studies to prepare these models against adversarial attacks. More details on the broader impact of our work are addressed in Appendix-C.

Acknowledgments and Disclosure of Funding

This work is fully funded by IBM Research. The authors would like to thank I-Hsin Chung, Ming-Hung Chen, Paul Crumley and James Norris for their support on computing infrastructure; Xiao Sun, Kailash Gopalakrishnan, Yuhong Li, Xiaodong Cui, Andrea Fasoli, Jiamin Ni, Derrick Liu and Mauricio Serrano for the technical discussions; Jeffrey Burns and Mukesh Khare for their executive support and feedback on the draft; IBM AI Hardware Center for providing computing resources.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[2] 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.

[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] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[5] Bichen Wu, Chenfeng Xu, Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Masayoshi Tomizuka, Kurt Keutzer, and Peter Vajda. Visual transformers: Token-based image representation and processing for computer vision. *arXiv preprint arXiv:2006.03677*, 2020.

[6] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer, 2020.

[7] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *arXiv preprint arXiv:2006.11477*, 2020.

[8] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. Hubert: Self-supervised speech representation learning by masked prediction of hidden units. 2021. doi: 10.48550/ARXIV.2106.07447. URL `https://arxiv.org/abs/2106.07447`.

[9] Sanyuan Chen, Chengyi Wang, Zhengyang Chen, Yu Wu, Shujie Liu, Zhuo Chen, Jinyu Li, Naoyuki Kanda, Takuya Yoshioka, Xiong Xiao, Jian Wu, Long Zhou, Shuo Ren, Yanmin Qian, Yao Qian, Jian Wu, Michael Zeng, Xiangzhan Yu, and Furu Wei. Wavlm: Large-scale self-supervised pre-training for full stack speech processing. 2021. doi: 10.48550/ARXIV.2110.13900. URL https://arxiv.org/abs/2110.13900.

[10] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models. 2021. doi: 10.48550/ARXIV.2108.07258. URL https://arxiv.org/abs/2108.07258.

[11] B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, ... Neelakantan, A., and D. Amodei. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[12] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

[13] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984*, 2020.

[14] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[15] Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[16] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *arXiv preprint arXiv:2007.14062*, 2020.

[17] et al. Agrawal, Ankur. A 7nm 4-core ai chip with 25.6tflops hybrid fp8 training, 102.4tops int4 inference and workload-aware throttling. *ISSCC*, 2021.

[18] Gabriele Prato, Ella Charlaix, and Mehdi Rezagholizadeh. Fully quantized transformer for improved translation. *CoRR*, abs/1910.10485, 2019. URL http://arxiv.org/abs/1910.10485.

[19] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert: Quantized 8bit bert. *arXiv preprint arXiv:1910.06188*, 2019.

[20] 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. *arXiv preprint arXiv:1909.05840*, 2020.

[21] Wei Zheng, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. Ternarybert: Distillation-aware ultra-low bit bert. *arXiv preprint arXiv:2009.12812*, 2020.

[22] Zhenhua Liu, Yunhe Wang, Kai Han, Siwei Ma, and Wen Gao. Post-training quantization for vision transformer. 2021. doi: 10.48550/ARXIV.2106.14156. URL `https://arxiv.org/abs/2106.14156`.

[23] Alex Bie, Bharat Venkitesh, Joao Monteiro, Md. Akmal Haidar, and Mehdi Rezagholizadeh. A simplified fully quantized transformer for end-to-end speech recognition. 2019. doi: 10.48550/ARXIV.1911.03604. URL `https://arxiv.org/abs/1911.03604`.

[24] Mitchell A. Gordon, Kevin Duh, and Nicholas Andrews. Compressing bert: Studying the effects of weight pruning on transfer learning. *arXiv preprint arXiv:2002.08307*, 2020.

[25] Mingjian Zhu, Yehui Tang, and Kai Han. Vision transformer pruning. *arXiv preprint arXiv:2104.08500*, 2021.

[26] Cheng-I Jeff Lai, Yang Zhang, Alexander H. Liu, Shiyu Chang, Yi-Lun Liao, Yung-Sung Chuang, Kaizhi Qian, Sameer Khurana, David Cox, and James Glass. Parp: Prune, adjust and re-prune for self-supervised speech recognition. 2021. doi: 10.48550/ARXIV.2106.05933. URL `https://arxiv.org/abs/2106.05933`.

[27] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. 2021. doi: 10.48550/ARXIV.2104.08378. URL `https://arxiv.org/abs/2104.08378`.

[28] et al. Ye, Shaokai. Progressive dnn compression: A key to achieve ultra-high weight pruning and quantization rates using admm. *arXiv preprint arXiv: 1903.09769*, 2019.

[29] S. J. Kwon, D. Lee, B. Kim, P. Kapoor, B. Park, and G. Y. Wei. Structured compression by weight encryption for unstructured pruning and quantization. *CVPR*, 2020.

[30] Jungwook Choi, Swagath Venkataramani, Vijayalakshmi Srinivasan, Kailash Gopalakrishnan, Zhuo Wang, and Pierce Chuang. Accurate and efficient 2-bit quantized neural networks. In *Proceedings of the 2nd SysML Conference*, volume 2019, 2019.

[31] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

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

[33] Shuchang Zhou, Yuzhi Wang, He Wen, Qinyao He, and Yuheng Zou. Balanced quantization: An effective and efficient approach to quantized neural networks, 2017.

[34] et al. Venkataramani, Swagath. Rapid: Ai accelerator for ultra-low precision training and inference. *ISCA*, 2021.

[35] et al. Jouppi, Norman P. In-datacenter performance analysis of a tensor processing unit. *ISCA*, 2021.

[36] et al. Wolf, Thomas. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[37] Ross Wightman, Hugo Touvron, and Herve Jegou. Resnet strikes back: An improved training procedure in timm. *arXiv preprint arXiv:2110.00476*, 2021.

[38] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers distillation through attention. *arXiv preprint arXiv:2012.12877*, 2021.

[39] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.

[40] Yash Bhalgat, Jinwon Lee, Markus Nagel, Tijmen Blankevoort, and Nojun Kwak. Lsq+: Improving low-bit quantization through learnable offsets and better initialization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 696–697, 2020.

[41] 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. 2019. doi: 10.48550/ARXIV.1906.00532. URL `https://arxiv.org/abs/1906.00532`.

[42] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. I-BERT: integer-only BERT quantization. *CoRR*, abs/2101.01321, 2021. URL `https://arxiv.org/abs/2101.01321`.

[43] 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. 2019. doi: 10.48550/ARXIV.1907.11692. URL `https://arxiv.org/abs/1907.11692`.

[44] Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jing Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. Binarybert: Pushing the limit of bert quantization. *arXiv preprint arXiv:2012.15701*, 2020.

[45] Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. Ternarybert: Distillation-aware ultra-low bit bert. *arXiv preprint arXiv:2009.12812*, 2020.

[46] Haotong Qin, Yifu Ding, Mingyuan Zhang, Qinghua Yan, Aishan Liu, Qingqing Dang, Ziwei Liu, and Xianglong Liu. Bibert: Accurate fully binarized bert. 2022. doi: 10.48550/ARXIV.2203.06390. URL `https://arxiv.org/abs/2203.06390`.

[47] Jing Jin, Cai Liang, Tiancheng Wu, Liqin Zou, and Zhiliang Gan. Kdlsq-bert: A quantized bert combining knowledge distillation with learned step size quantization. *arXiv preprint arXiv:2101.05938*, 2021.

[48] Hanlin Tang, Xipeng Zhang, Kai Liu, Jianchen Zhu, and Zhanhui Kang. Mkq-bert: Quantized bert with 4-bits weights and activations. 2022. doi: 10.48550/ARXIV.2203.13483. URL `https://arxiv.org/abs/2203.13483`.

[49] Zhenhua Liu, Yunhe Wang, Kai Han, Wei Zhang, Siwei Ma, and Wen Gao. Post-training quantization for vision transformer. *arXiv preprint arXiv:2106.14156*, 2021.

[50] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. *arXiv preprint arXiv:2005.12872*, 2020.

[51] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained bert networks. 2020. doi: 10.48550/ARXIV.2007.12223. URL `https://arxiv.org/abs/2007.12223`.

[52] Mitchell A. Gordon, Kevin Duh, and Nicholas Andrews. Compressing bert: Studying the effects of weight pruning on transfer learning. 2020. doi: 10.48550/ARXIV.2002.08307. URL `https://arxiv.org/abs/2002.08307`.

[53] Xiaohan Chen, Yu Cheng, Shuohang Wang, Zhe Gan, Zhangyang Wang, and Jingjing Liu. Earlybert: Efficient bert training via early-bird lottery tickets. 2021. doi: 10.48550/ARXIV.2101.00063. URL `https://arxiv.org/abs/2101.00063`.

[54] Huanrui Yang, Hongxu Yin, Pavlo Molchanov, Hai Li, and Jan Kautz. Nvit: Vision transformer compression and parameter redistribution. 2021. doi: 10.48550/ARXIV.2110.04869. URL `https://arxiv.org/abs/2110.04869`.

[55] Fang Yu, Kun Huang, Meng Wang, Yuan Cheng, Wei Chu, and Li Cui. Width depth pruning for vision transformers. *AAAI 2022*, 2022.

[56] Mingjian Zhu, Yehui Tang, and Kai Han. Vision transformer pruning. 2021. doi: 10.48550/ARXIV.2104.08500. URL https://arxiv.org/abs/2104.08500.

[57] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? 2019. doi: 10.48550/ARXIV.1905.10650. URL https://arxiv.org/abs/1905.10650.

[58] Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. Dynabert: Dynamic bert with adaptive width and depth. 2020. doi: 10.48550/ARXIV.2004.04037. URL https://arxiv.org/abs/2004.04037.

[59] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016. URL http://arxiv.org/abs/1606.05250.

[60] Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books, 2015.

[61] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: An asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, 2015. doi: 10.1109/ICASSP.2015.7178964.

[62] J. Kahn, M. Riviere, W. Zheng, E. Kharitonov, Q. Xu, P.E. Mazare, J. Karadayi, V. Liptchinsky, R. Collobert, C. Fuegen, T. Likhomanenko, G. Synnaeve, A. Joulin, A. Mohamed, and E. Dupoux. Libri-light: A benchmark for ASR with limited or no supervision. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, may 2020. doi: 10.1109/icassp40776.2020.9052942. URL https://doi.org/10.1109%2Ficassp40776.2020.9052942.

[63] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[64] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Section 2 and 3
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Instructions and code examples are in Appendix
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] in Appendix
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] but key results are averaged results over random seeds
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix-A: Model and Dataset

In this section, we provide details on the datasets, model architectures, and settings used for full precision fine-tuning to replicate the baseline accuracy reported in prior publications.

A.1 BERT-base/SQuAD1.1

SQuAD1.1 is the Stanford Question Answering dataset [59]. It is a reading comprehension dataset, consisting of a collection of 100k question and answer pairs from reading passages, where the answer to every question is a segment of corresponding passage. The task is to predict the answer text span in a passage.

BERT-base model [2] is a transformer model pre-trained on a large corpus of English data, BooksCorpus [60], with 800M words and English Wikipedia of 2500M words. The model was pre-trained in a self-supervised fashion using a masked language modeling (MLM) procedure. The BERT-base model consists of an input embedding, 12 transformer blocks, and an output linear layer, with a total of 100M parameters. The input embedding is a sum of token embeddings, segmentation embeddings and the position embeddings. Each transformer block contains 12 self-attention heads and a hidden size of 768.

To fine-tune BERT-base on the SQuAD1.1 downstream task, we use a batch size of 12 and a sequence length of 384. The experiments are performed on 4 V100 GPUs with a per-gpu batch size of 3. For the full precision fine-tuning baseline, we follow the fine-tuning strategy from [2] and use the AdamW optimizer with a learning rate of 3e-5 with linear decay. The model is fine-tuned for 2 epochs with a

dropout probability of 0.1. We obtain a baseline F1 score of 88.69, which closely matches the F1 score (88.50) published in [2]. Fig. 6a shows the convergence curve of the full precision fine-tuning baseline.

A.2 Wav2vec2.0/Librispeech

Librispeech is a corpus of English speech for the automatic speech recognition (ASR) task [61]. It contains 1000 hours of speech sampled at 16 kHz. The training data is split into 3 partitions of 100hr, 360hr and 500hr sets with 'clean' and 'other' categories. In this work, we use the 100hr-clean data for the downstream task and the clean validation subset for evaluation.

Wav2vec2.0-large is a speech model pre-trained on the audio data from LibriVox (LV-60k) [62] in a self-supervised manner [7]. In this work, we use the Wav2vec2.0-large model, which has a large transformer backbone with 24 transformer blocks. The hidden dimension, inner dimension, and number of attention heads in each transformer block are 1024, 4096 and 16, respectively. Prior to being fed into the transformer backbone, the raw waveform is encoded through multiple 1d-convolution layers followed by layer normalization and GeLU activations.

The pre-trained model is fine-tuned on Librispeech's 100 hour clean subset using standard Connectionist Temporal Classification (CTC) loss. We follow the implementation and settings from HuggingFace Transformer [36] for the fine-tuning. Specifically, for the full precision fine-tuning baseline, we use the AdamW optimizer with betas=(0.9,0.999) and a learning rate of 3e-4. The learning rate decays linearly after 500 warm-up steps. We use 8 V100 GPUs to tune the model for 3 epochs with a total batch size of 32. We achieve a baseline Word Error Rate (WER) of 4.20 %, matching the result provided by HuggingFace Transformer (4.2 %). Fig. 6b shows the convergence curve of the full precision fine-tuning baseline.

A.3 ViT/ImageNet1k

ImageNet1k [63] is an image classification benchmark which consists of 1000-categories of objects with over 1.2M training and 50K validation images.

ViT-base model is a BERT-like transformer encoder model taking images as the input for image classification tasks. The input images are split into fixed-sized patches of 16x16 and linearly embedded. The ViT-base model has 12 transformer blocks with 12 attention heads and a hidden dimension of 768 [4]. In this paper, we use the pre-trained model that is pre-trained on ImageNet21k [63] and then fine-tune it on ImageNet1k for downstream image classification. We use a resolution of 384x384 for fine-tuning following the original settings of [4]. The optimizer is SGD with a learning rate of 0.01. We tune the model for 8 epochs using a Cosine learning rate schedule, gradient clipping of 1.0, and batch size of 512 on 32 V100 GPUs. With these settings, we achieve an accuracy of 84.12, which matches the accuracy (83.97) published in [4]. Fig. 6c shows the convergence curve of full precision fine-tuning baseline.

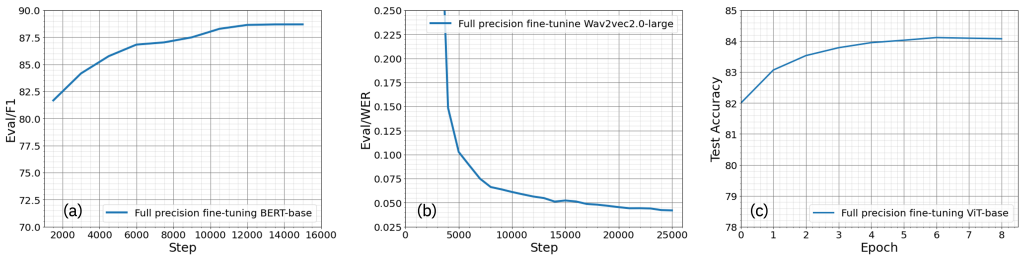

Figure 6: Convergence curves of full precision fine-tuning of (a) BERT-base on SQuAD1.1; (b) Wav2vec2.0-large on Librispeech; and (c) ViT-base on ImageNet1k.

B Appendix-B: Quantization- and Sparsity-aware Fine-tuning Settings

In this section, we provide details on the implementation of quantization and pruning operations, as well as the hyper-parameters used for the fine-tuning of deeply compressed models.

B.1 Quantization/Pruning Implementation

We implement quantization and pruning in the PyTorch framework. For each linear module or batch-matrix-matrix multiplication (bmm) operation, we insert quantization operations to quantize both the activations and weights. Fig. 7 shows screenshot examples of quantized modules with inserted quantization operations from the graph of the quantized BERT-base model. For linear modules, both input activations and weights are quantized, as shown in Fig. 7(a) a query layer in self-attention and Fig. 7(c) an intermediate-dense layer in the feed-forward network (FFN); for bmm operations, both input activations are quantized as shown in Fig. 7(b).

Fig. 8 presents a toy example showing a deeply compressed linear module, i.e. QLinear, running a forward pass with random input. The linear layer is quantized in 4-bit for both weights and activations using SAWB+ and PACT quantizers, respectively. Weights are further pruned with 50% sparsity using a fine-grained group of 4 as discussed in section 2.2. The printout shows the pruning mask tensor, pruned weight tensor, and quantized weight tensor computed during the forward pass.

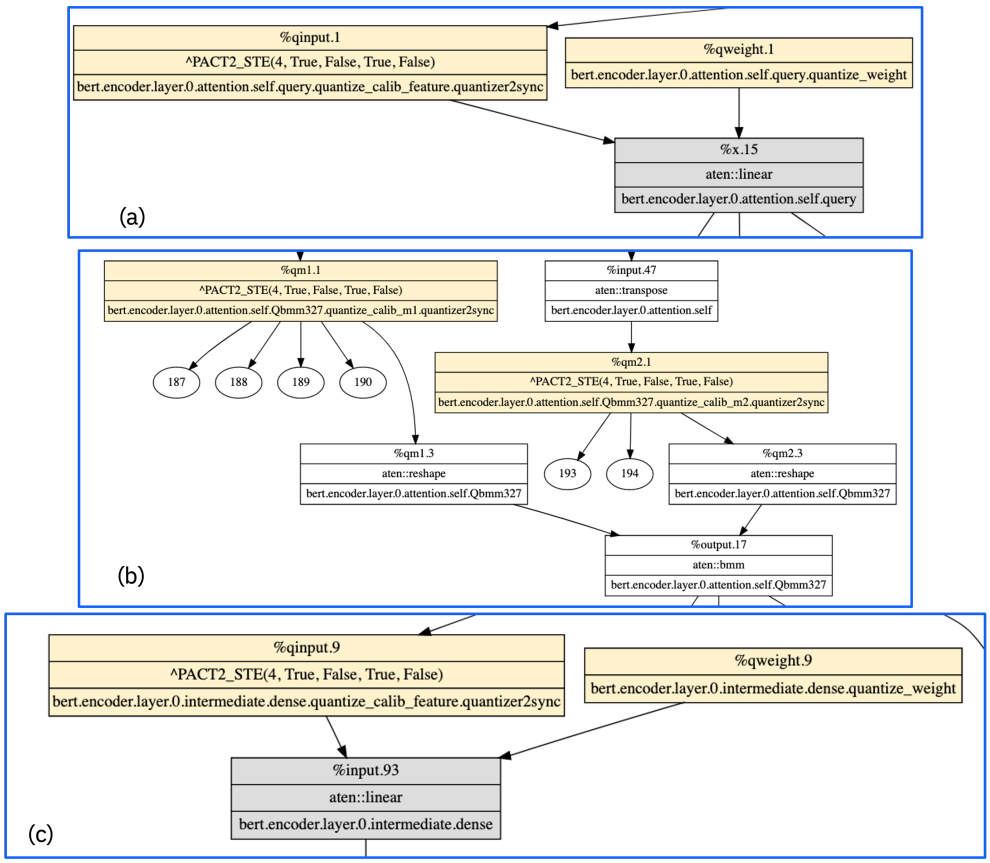

Figure 7: a) Screenshot examples of a quantized graph with implemented weight and activation quantization operations, for (a) a query linear layer; (b) a bmm operation for attention computation; and (c) an intermediate dense linear layer in FFN from layer0 (the first transformer block) of the deeply compressed BERT-base model.

B.2 Fine-tunine Setting

We use a common setting for all three models and benchmarks. Full precision fine-tuned models are used for the initialization of INT8, sparse INT8, and INT4 models. For the sparse INT4 model, we use a sparse INT8 model for initialization as explained in section 2.3. Fig. 9 shows a schematic of the fine-tuning procedures. The SAWB+ quantizer is used for weight quantization for all models as discussed in section 2.1.1. The coefficients used in SAWB+ are listed in Table 5. The MinMax or PACT quantizer is used for the activation quantization. For the PACT quantizer (discussed in section

```python
#a toy example to quantize and prune one Linear layer with dummy inputs
model_q = QLinear(in_features=64, out_features=4, num_bits_feature=4, num_bits_weight=4, \
                  p_group=4, p_ratio = 0.5, qa_mode='pact', qw_mode='sawb+')
#get pruning mask
model_q.get_mask()
output = model_q(torch.rand(1, 4, 64))
```

```
Weights[:,:8]:
 tensor([[ 1.1034e-01,  7.0334e-02, -9.7582e-02,  4.9689e-02, -8.1956e-07,
         -9.7793e-02, -8.3745e-06, -1.0821e-01],
        [-1.1906e-01,  8.3641e-02,  3.2591e-02,  3.1367e-02, -1.5860e-02,
          6.3644e-02, -1.1322e-01, -1.1617e-01],
        [ 2.6805e-02, -3.2602e-02,  1.0721e-01, -3.6630e-02, -1.1574e-01,
          7.5815e-02, -5.7634e-02, -1.1444e-02],
        [ 1.1820e-01,  5.0626e-02, -7.3768e-02,  6.0336e-02, -8.2874e-02,
         -6.0145e-02,  1.0179e-01,  9.6717e-02]])
Mask[:,:8]:
 tensor([[1., 0., 1., 0., 0., 1., 0., 1.],
        [1., 1., 0., 0., 0., 0., 1., 1.],
        [0., 0., 1., 1., 1., 1., 0., 0.],
        [1., 0., 1., 0., 0., 0., 1., 1.]])
Pruned weight[:,:8]:
 tensor([[ 0.1103,  0.0000, -0.0976,  0.0000, -0.0000, -0.0978, -0.0000, -0.1082],
        [-0.1191,  0.0836,  0.0000,  0.0000, -0.0000,  0.0000, -0.1132, -0.1162],
        [ 0.0000, -0.0000,  0.1072, -0.0366, -0.1157,  0.0758, -0.0000, -0.0000],
        [ 0.1182,  0.0000, -0.0738,  0.0000, -0.0000, -0.0000,  0.1018,  0.0967]])
Quantized weight[:,:8]:
 tensor([[ 0.1067,  0.0000, -0.1067,  0.0000,  0.0000, -0.1067,  0.0000, -0.1067],
        [-0.1067,  0.0711,  0.0000,  0.0000,  0.0000,  0.0000, -0.1067, -0.1067],
        [ 0.0000,  0.0000,  0.1067, -0.0356, -0.1067,  0.0711,  0.0000,  0.0000],
        [ 0.1067,  0.0000, -0.0711,  0.0000,  0.0000,  0.0000,  0.1067,  0.1067]])
```

Figure 8: a) A toy example of a quantized and pruned linear module running a forward pass with a random input. The printout shows the pruning mask, pruned weight and quantized weight tensors.

Table 5: Coefficients for SAWB+.

Precision	C1	C2
INT4	12.68,	-12.80
INT8	31.76,	-35.04

2.1.2), three hyper-parameters are used to train α and α_n parameters, i.e. initiation in percentile, learning rate (α_lr) and L2 decay (α_decay). The detailed settings used for three benchmarks are as follows.

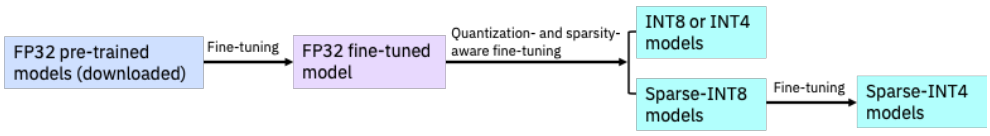

Figure 9: A schematic of the fine-tuning procedures leading to deep compressed models. Quantization- and sparsity- aware fine-tuning are initialized by FP32 fine-tuned models to obtain the INT8, INT4 or sparse-INT8 models. The sparse-INT8 models are further fine-tuned to achieve sparse-INT4 models.

Table 6 lists the settings for BERT-base/SQuAD1.1 benchmark. We use the same baseline optimization methods as described in Appendix 1.1, except that the compressed models are fine-tuned for 4 epochs with a larger dropout (0.2) or a scheduled dropout as introduced in section 2.3.3. Fig. 10 shows the convergence curves of the deep compressed models.

Table 7 lists the settings for Wav2vec2.0-large/Librispeech benchmark. We use the same baseline optimization methods as described in Appendix 1.2, except that the compressed models are fine-tuned for 6 epochs. Fig. 11 shows the convergence curves of the deeply compressed models.

Table 8 lists the settings for ViT-base/ImageNet1k benchmark. For INT8/4 models without pruning, we use the same baseline optimization methods as described in Appendix 1.3. For sparse INT8/4 models, we tune the model for 16 epochs with an initial learning rate of 0.05, keeping the rest

Table 6: Qantization/Sparsity-aware fine-tuning setting for BERT-base on SQuAD1.1. "Sp" indicates sparsity.

Precision Sparsity	Weight Quantizer	Activation Quantizer	Initialization Model	Percentile (%)	α_lr	α_decay	Dropout
INT8	SAWB+	MinMax	FP32	–	–	–	0.2
INT8+50%Sp	SAWB+	MinMax	FP32	–	–	–	0.2
INT4	SAWB+	PACT	FP32	99	1e-3	1e-3	0.2.
INT4+50%Sp	SAWB+	PACT	INT8+50%Sp	99	1e-3	1e-3	Scheduled

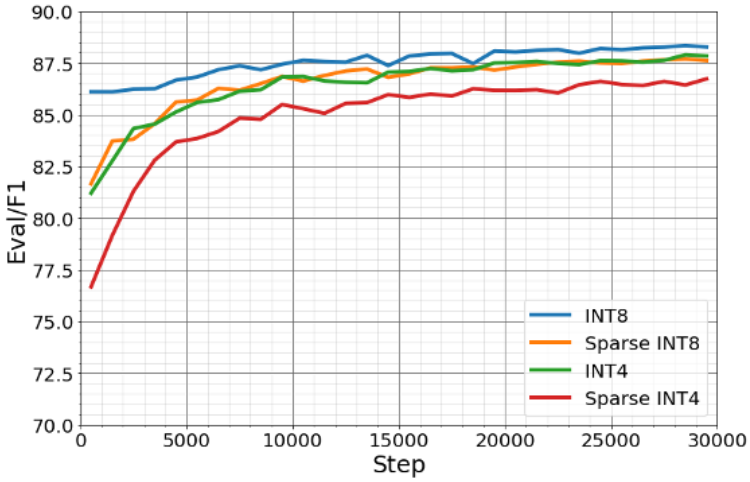

Figure 10: Convergence curves of INT8, sparse INT8, INT4 and sparse INT4 BERT-base models on SQuAD1.1.

of hyper-parameters the same as the baseline. Fig. 12 shows the convergence curves of the deep compressed models.

B.3 Error Bar

To understand the uncertainty of the results, we run five experiments with random initial seeds for all three sparse INT4 models. The means and standard deviations calculated using the five runs are shown in the Table 9. The results are consistent with Table 4.

C Appendix-C: Broader Impact

Dedicated hardware accelerators for DNN inference, including CPUs, GPUs, TPUs and other AI platforms, have powered the deployment of machine learning for real-life applications in both cloud and edge devices. Reduced precision innovations (FP16, FP8 and INT8), together with sparsity,

Table 7: Qantization/Sparsity-aware fine-tuning setting for Wav2vec2.0-large on Librispeech. Sp is short for sparsity.

Precision Sparsity	Weight Quantizer	Activation Quantizer	Initialization Model	Percentile (%)	α_lr	α_decay
INT8	SAWB+	MinMax	FP32	–	–	–
INT8+50%Sp	SAWB+	MinMax	FP32	–	–	–
INT4	SAWB+	PACT	FP32	max	1e-2	7e-3
INT4+50%Sp	SAWB+	PACT	INT8+50%Sp	99.9	1e-2	3e-2

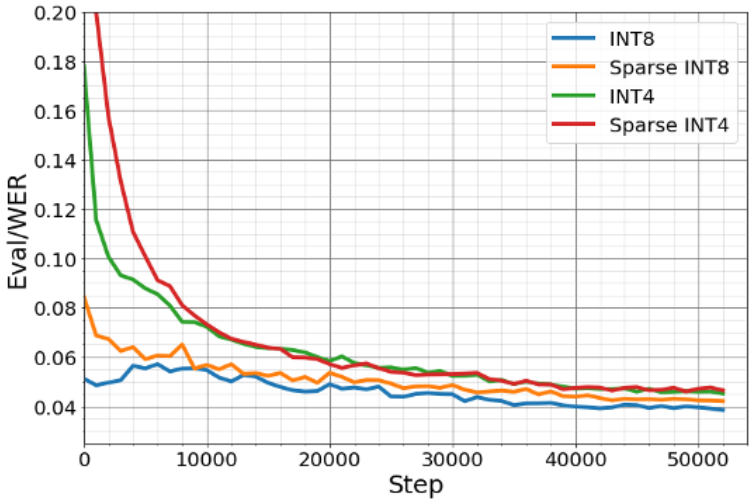

Figure 11: Convergence curves of INT8, sparse INT8, INT4 and sparse INT4 Wav2vec2.0-large models on Librispeech.

Table 8: Qantization/Sparsity-aware fine-tuning setting for ViT-base on ImageNet1k. Sp is short for sparsity.

Precision Sparsity	Weight Quantizer	Activation Quantizer	Initialization Model	Percentile (%)	α_lr	α_decay	Epoch
INT8	SAWB+	MinMax	FP32	–	–	–	8
INT8+50%Sp	SAWB+	MinMax	FP32	–	–	–	8
INT4	SAWB+	PACT	FP32	99.9	1e-2	1e-5	16
INT4+50%Sp	SAWB+	PACT	INT8+50%Sp	99.9	1e-2	1e-6	16

have recently improved the capability of these accelerators by 4-8$\times$ and have dramatically improved energy cost and carbon emissions. Although pre-trained transformers have unlocked the power of transfer learning and are leading to breakthroughs in multiple application domains, the architecture is too complex for many production systems, such as those for edge-computing inference. There are many ongoing efforts to reduce the size of these models while retaining model performance and transferability. Deep compression of transformers, which is presented in this work, aims to push this front aggressively to enable faster and cheaper inference systems for a wide spectrum of deep learning models and domains. We believe that sparse 4-bit inference solutions can accelerate ML deployment and provide significant cost and energy savings for corporations and research institutes — in addition to helping reduce the carbon / climate impact of AI inference. By improving power efficiency by about $4\times$ over current transformers running in FP16 (and $8\times$ vs. default FP32 designs), the carbon footprint for predicting with large DNN models can be significantly reduced [64].

The reduction in computational energy and memory footprint could also enable the inference of large transformer models to be carried out on edge devices (mobile platforms, health care devices,

Table 9: Uncertainty study of three sparse INT4 models. The means and standard deviations are calculated using 5 runs with different initial seeds.

Precision Sparsity	BERT-base (F1%)	Wav2vec2.0 (WER %)	ViT (Accuracy %)
INT4+50%Sp	87.04 ± 0.10	4.63 ± 0.06	83.65 ± 0.12

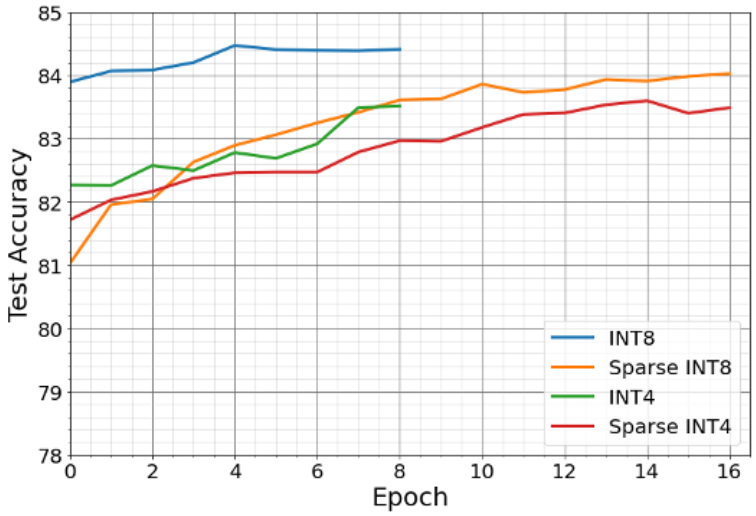

Figure 12: Convergence curves of INT8, sparse INT8, INT4 and sparse INT4 ViT-base models on ImageNet1k.

security cameras, consumer drones, etc.). This, in turn, could alleviate security and privacy concerns of sending data back to the Cloud for prediction tasks.

We would also like to emphasize that although we have shown promising results and limited accuracy loss in comparison to FP32 downstream tasks, deeply compressed transformer models using our solutions could still be subject to unexpected instabilities. This may necessitate a careful examination of these optimization techniques and numerical formats over a wider range of models and perfected alongside the development of ML model research. The risk of using deeply compressed transformer models in real inference applications is most likely higher than full precision dense models and thus requires task-specific robustness studies to prepare these models against adversarial attacks. In addition, More work is also needed to assess the impact of deeply compressed models in fairness and explainability.

In addition, for each application domain, we chose representative models, datasets, and tasks (SQuAD1.1, Librispeech, and ImageNet1k) that have been used widely to evaluate compression algorithms. Going forward, it will be important to study different (simple and complex) downstream tasks across multiple datasets within each domain; this will be the focus of our future work. We will study various datasets for each task to assess the generalization of our proposed techniques.