HOLOFORMER: DEEP COMPRESSION OF PRE-TRAINED TRANSFORMS VIA UNIFIED OPTIMIZATION OF N:M SPARSITY AND INTEGER QUANTIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

In recent years, large pre-trained Transformer networks have demonstrated dramatic improvements in many Natural Language Processing (NLP) tasks. However, the huge size of these models brings significant challenges to fine-tuning and online deployment due to latency and cost constraints. Recently, hardware manufacturers have released new architectures that support efficient N:M sparsity and low-precision integer computation for fast inferencing. In contrast to unstructured sparsity, N:M sparsity specifies that out of each chunk of N contiguous weight parameters, exactly M parameters are non-zero. Moreover, these architectures also support processing data with reduced precision, such as INT8. Prior work often considers inducing N:M sparsity and integer quantization in isolation or as independent pieces of a compression pipeline. However, there lacks a systematic investigation towards how N:M sparsity and integer quantization can be effectively combined to exploit the maximum degree of redundancy and enable even faster acceleration for pre-trained Transformer networks.

In this work, we propose a unified, systematic approach to learning N:M sparsity and integer quantization for pre-trained Transformers using the Alternating Directions Method of Multipliers (ADMM). We show that both N:M sparsity and integer quantization and their combinations can be framed as non-convex constrained optimization problems and solved in a unified manner. When evaluated across the GLUE suite of NLP benchmarks, our approach outperforms baselines that consider each of these problems independently, retaining 99.4% accuracy of the dense baseline while being able to execute on newly released hardware effectively.

1 INTRODUCTION

Large-scale Transformer-based models, such as BERT (Devlin et al., 2019) and GPT (Brown et al., 2020), have achieved outstanding performance for a wide variety of natural language tasks, such as natural language inference (Yang et al., 2019; Raffel et al., 2019), question answering (Liu et al., 2019), and others. These large-scale models often employ a transfer learning paradigm: Pre-train the model on massive open-domain data followed by a fine-tuning stage to adapt it to a specific domain with task-specific data. In recent years, the size of these models has been growing at an unprecedented speed, from hundreds of millions of parameters (Devlin et al., 2019; Radford et al., 2019) to over multi-billions (Rajbhandari et al., 2019; Raffel et al., 2020; Brown et al., 2020), and recent studies (Kaplan et al., 2020) show that the performance of these models also continue to improve as their sizes increase. As a result, we expect this trend to continue.

Despite showing excellent improvements in accuracy, it becomes excessively expensive and slow to train these models even on large GPU/TPU clusters. Furthermore, the trained model also raises significant challenges during deployment due to latency and cost constraints. These challenges have motivated diverse techniques to compress and accelerate these models, including but not limited to knowledge distillation (Sanh et al., 2019; Wen et al., 2016a; Jiao et al., 2020; Sun et al., 2020; Mao et al., 2020), sparsification (Chen et al., 2020; Guo et al., 2019a), and quantization (Zafrir et al., 2019a; Shen et al., 2020; Bai et al., 2020). While achieving high compression ratios, many methods (e.g., unstructured sparsity) fail to improve the computation efficiency on modern hardware and require custom accelerators (Liu et al., 2021).

Recently, Sparse Tensor Cores have entered the realm of DNN since NVIDIA released the Ampere architecture in its commodity hardware (nvi, 2020). Sparse Tensor Cores not only support N:M sparsity but also carries the capability of accelerating fine-grained sparse matrix multiplication with low-bit computation. N:M sparsity is a form of semi-structured sparsity that allows a model to preserve M parameters out of N consecutive parameters. This relatively weak constraint on sparsification allows for sparse representations similar in flexibility to those of unstructured approaches. On the other hand, N:M sparsity offers performance improvements similar to structured sparsification techniques that remove parameters in groups (e.g., rows or columns), e.g., by introducing a small set of multiplexers that select values from the input matrix corresponding to the retained values in the weight matrix (Mishra et al., 2021). The 4:2 sparsity in A100 GPU enables a 2X acceleration of regular matrix multiplication in DNNs. On top of that, reducing weights and activations through low-precision integer data (e.g., from FP32 to INT8) would yield even more compute, bandwidth, memory, and power savings.

Previous works exist that aim to compress models to better leverage Sparse Tensor Cores. ASP (Mishra et al., 2021) proposes training the dense network until convergence and then using single-shot magnitude-based pruning (i.e., with a fixed mask) to obtain sparsity conformant to the N:M constraints. Zhou et al. (2021) propose to train a model from scratch with an N:M mask, using a sparse-refined straight-through estimator (SR-STE). Both methods focus on learning N:M sparsity for common cases. However, their approaches pose challenges for compressing pre-trained Transformer networks. ASP requires a second costly sparse pre-train of the model and the single-shot magnitude-based pruning might hurt the knowledge transferability to different downstream tasks. SR-STE, on the other hand, sparsifies from the random model initialization, which avoids the costly sparse retraining but also necessitates performing the pre-training process with only a sparse representation in mind. Researchers have also looked into compressing Transformers through quantization. Similar to SR-STE, Q8BERT (Zafrir et al., 2019b) applies quantization-aware training (QAT) to the BERT model by maintaining a dense copy of the weights and quantizes the model in each iteration while keeping the gradients oblivious to that process using STE. More recently, Shen et al. (2020) proposed to quantize Transformer models based on the sensitivity obtained using the Hessian spectrum. These methods can effectively reduce the bits for model weights and activations, but they do not consider both quantization and N:M sparsity and miss opportunities to further reduce the computational cost.

Given that the new hardware supports both N:M sparsity and integer quantization, it naturally prompts the question, *how can we obtain N:M sparsified models with quantized weights efficiently and effectively?* In this work, we formulate the compression with N:M sparsity and integer quantization as a non-convex optimization problem with combinatorial constraints. We then present a principled and unified method, HoloFormer, for learning Transformer networks with N:M sparsity and low-bit quantized weights simultaneously, which accelerates the inference of pre-trained Transformer networks on the newly released Sparse Tensor Core hardware using the technique of Alternating Direction Method of Multipliers, a proven method for solving large-scale constrained optimization problems. We conduct comprehensive experiments and demonstrate that HoloFormer retains 99.4% accuracy on a wide range of downstream tasks. Finally, we perform a detailed analysis of our approach and shed light on how HoloFormer affects fine-tuning accuracy for downstream tasks.

2 Background and Related Work

Compression Techniques for Pre-trained Transformers. Many work are dedicated to compressing large-scale pre-trained language models. In particular, sparsification has been demonstrated to be a successful technique for achieving a high compression ratio for computer vision tasks and convolution networks (Han et al., 2016). However, sparsifying pre-trained language models turns out to be a challenging task because the semi-supervised models tend to have much less intrinsic sparsity than CNN-based models (Gordon et al., 2020). Chen et al. (2020) extend the Lottery Ticket Hypothesis (Frankle & Carbin, 2019) to pre-trained models, finding that winning tickets for the pre-training tasks do transfer universally to downstream tasks. Guo et al. (2019b) show that a proximal pruning strategy achieves higher accuracy than competing lasso regularization methods and iterative magnitude pruning. However, most of these studies still focus on unstructured sparsity, which encounters difficulty in obtaining large speedups on modern hardware due to their irregularity in weight patterns. As a structured technique, Michel et al. (2019) observe that many transformer heads themselves are redundant and can be pruned with minimal accuracy loss. Despite having a structured weight pattern,

this technique provides limited acceleration benefits since much of the compute time is spent on the intermediate layers rather than the attention heads. In addition to parameter sparsification, pre-trained Transformers, such as BERT, can also realize compression gains with model quantization (Zafrir et al., 2019b; Shen et al., 2020; Dong et al., 2020; Kim et al., 2021). Since Transformer networks often have a high memory footprint and require heavy compute and bandwidth during inference, quantization would not only reduce the model size but also accelerate inference time with reduced memory bandwidth consumption. Prior work shows that it is possible to even push the model to have only binary weights in the extreme case (Bai et al., 2020). However, extremely low-bit quantization would require custom hardware support and have not become a mainstream way for running inference. Sparsification and quantization can be combined to achieve more effective overall compression, as in (Han et al., 2016). However, so far for pre-trained Transformer models, existing work considers them in isolation. In contrast, we aim at providing a systematic investigation and approach on unifying these techniques for compressing pre-trained Transformers.

Sparse Tensor Core. Recently, NVIDIA announced the A100 GPU, which contains Sparse Tensor Cores that are able to accelerate N:M sparsified models with INT8 quantized weights. There have been some studies on how to compress DNN models to better leverage Sparse Tensor Core. For example, SR-STE uses an extended Straight-through Estimator and a sparse-refined term to induce N:M sparsity. SR-STE is evaluated on image classification and neural machine translation tasks and demonstrates improved accuracy over NVIDIA's ASP (Mishra et al., 2021). However, SR-STE is designed for training from random initialization, not a pretrained model.

NVIDIA has demonstrated with its cuSPARSElt (Mishra et al., 2021) sparse linear algebra library that NxM semi-structured sparsity may provide performance improvements of between 1.4-1.6x for typical layer dimensions in a BERT model. Models with larger layer dimensions (depth of model is largely irrelevant for these calculations) can even approach the theoretical limit of 2x speedup. Since the performance improvement moving from 16-bit floating point implementations (either fp16 or bf16) is nearly linear to 8-bit integer quantization, the combination of integer quantization and semi-structured sparsity can be expected to accelerate the core GEMMs by more than 2.5x for smaller models and approach four times the throughput for large models.

ADMM for Neural Networks. Prior work uses the Alternating Direction Method of Multipliers (ADMM) primarily for compressing convolutional neural networks. Ye et al. (2018b) explore an iterative unstructured pruning approach to ADMM that shortcuts full ADMM convergence with a masked retraining operation. Ren et al. (2019) use ADMM for both unstructured sparsification and quantization of the model while combining the technique with a heuristic for determining whether speedup will be achieved given the achieved compression of a layer. Ye et al. (2018a; 2019) propose to use ADMM for both unstructured weight pruning and weight clustering/quantization. Ma et al. (2020) use domain knowledge of CNN filter structure to perform a structured pruning operation for mobile-hardware efficient compression. Niu et al. (2020) propose to use ADMM for pattern- and connectivity-based weight pruning. While the above techniques all use ADMM as the sparsifying and/or quantization mechanism, none examine in-depth its applicability to pre-trained language models for jointly learning N:M sparsity and integer quantization.

3 HoloFormer: Joint N:M Sparsification and Integer Quantization

This section first formulated model compression with N:M sparsity and quantization as a joint constrained optimization problem. We then describe our optimization methodology, including the specific aspects of pre-trained Transformer networks that define the optimization pipeline.

3.1 Problem Formulation

Consider a Transformer-based language model Φ with a collection of weights $W = \{\mathbf{W}_i\}_{i=1}^{L}$. The model is first pre-trained on open-domain data to get $W^0 = \{\mathbf{W^0}_i\}_{i=1}^{L}$ and then adapts to $\{W'_i\}_{i=1}^{L}$ with a domain-specific dataset D for a natural language understanding task, such as sentiment analysis, entailment, question-answering, etc. D is composed of input (e.g., text phrases) and target pairs: $\{x_i, y_i\}_{i=1}^{N}$. The goal of HoloFormer is to load W^0 and fine-tune it on D to W' such that each ${W'}_i \in W'$ satisfies two constraints simultaneously: (1) C_i^1 = $\{{W'}_i|$ at most M weight parameters having non-zero values out of N consecutive weights in ${W'}_i\}$, and (2) C_i^2 = $\{{W'}_i|$ the remaining non-zero weights in ${W'}_i$ are quantized to a finite set of integer values$\}$. Finally, $\Phi(W')$ should achieve

similar performance in comparison to fine-tuning the task-specific objective function $f(\{\mathbf{W}_i\}_{i=1}^L)$ using W_0 but without constraints.

3.2 DECOUPLING THE OPTIMIZATION PROBLEM INTO SUBPROBLEMS WITH ADMM

The above optimization problem contains combinatorial constraints and is non-convex, which is difficult to solve directly through gradient-based methods such as SGD or Adam (Kingma & Ba, 2015). On the other hand, the alternating direction method of multipliers (ADMM) has been proven to be a powerful tool for solving large-scale constrained optimization (Ouyang et al., 2013). To apply ADMM, we first define indicator functions to incorporate the combinatorial constraints:

$$g_i(\mathbf{W}_i) = \begin{cases} 0 & \text{if } \mathbf{W}_i \in \mathbf{C_i^1} \\ \infty & \text{otherwise} \end{cases} \quad h_i(\mathbf{W}_i) \begin{cases} 0 & \text{if } \mathbf{W}_i \in \mathbf{C_i^2} \\ \infty & \text{otherwise} \end{cases} \tag{1}$$

We then introduce auxiliary variables Z_i and modify the fine-tuning objective function as:

$$\min_{\{\mathbf{W}_i\}} f(\{\mathbf{W}_i\}_{i=1}^L) + \sum_{i=1}^{L} h_i \cdot g_i(\mathbf{Z}_i) \text{ subject to } \mathbf{W}_i = \mathbf{Z}_i, i = 1, \ldots, L \tag{2}$$

Decomposing into subproblems. Through ADMM (Boyd et al., 2011), we can rewrite the optimization in Equation 2 with augmented Lagrangian, which decomposes Equation 2 into two subproblems on W and Z. The overall problem then can be solved by solving subproblems iteratively until convergence. The first problem is:

$$\min_{\{\mathbf{W}_i\}} f(\{\mathbf{W}_i\}_{i=1}^L) + \sum_{i=1}^{L} \frac{\rho}{2} \|\mathbf{W}_i - \mathbf{Z}_i^k + \mathbf{U}_i^k\|_F^2 \tag{3}$$

which consists of two terms: The first term is the standard differentiable loss function for the fine-tuning task; The second term is a quadratic regularization term, which is convex and differentiable. This subproblem can be solved (e.g., via ADAM) with the same complexity of training $f(\cdot)$. U_i^k in the second term is the dual variable, dynamically updated in each ADMM iteration. We have have a second subproblem, which is:

$$\min_{\{\mathbf{Z}_i\}} \sum_{i=1}^{L} h_i \cdot g_i(\mathbf{Z}_i) + \sum_{i=1}^{L} \frac{\rho}{2} \|\mathbf{W}_i^{k+1} - \mathbf{Z}_i + \mathbf{U}_i^k\|_F^2 \tag{4}$$

Since $g_i(\cdot)$ and $h_i(\cdot)$ are the indicator function of C_i^1 and C_i^2, the analytical solution of this subproblem is

$$\mathbf{Z}_i^{k+1} = \Pi_{S_i}(\mathbf{W}_i^{k+1} + \mathbf{U}_i^k) \tag{5}$$

where $\Pi_{S_i}(\cdot)$ is the Euclidean projection of $\mathbf{W}_i^{k+1} + \mathbf{U}_i^k$ onto the constraints C_i^1 and C_i^2. We will discuss more details about the Euclidean projection later in the next section.

After solving the subproblems, we update the dual variable U as $\mathbf{U}_i^k := \mathbf{U}_i^{k-1} + \mathbf{W}_i^k - \mathbf{Z}_i^k$ to guarantee that the dual feasibility condition is satisfied in each ADMM iteration. During training, we perform the above steps in an alternating manner, i.e., performing some number of steps with Adam to solve the first subproblem, solving the Euclidean projection for each weight matrix for the second sub-problem, and finally updating the dual variable. The optimization proceeds until the $\mathbf{W}$ and $\mathbf{Z}$ variables have converged, at which point we have a network compliant with both N:M sparsity (C_i^1) and integer quantization (C_i^2) constraints.

Practical consideration. We note that not all weights in pre-trained Transformer models need to satisfy the aforementioned constraints. For N:M sparsity and integer quantization, we focus on considering computationally intensive layers in Transformer blocks. As in (Devlin et al., 2019), BERT and similar pre-trained models typically consist of three major components: embedding layer, Transformer blocks, and output layer (See Figure 1). Each Transformer block further consists of 6 fully connected sub-layers: the query Q, key K, value V layers, the attention output layer $Attn.$, and two feed-forward networks $FFN1$ and $FFN2$. Each of the fully connected layers can take advantage of N:M semi-structured sparsity and quantization; furthermore, these layers constitute the

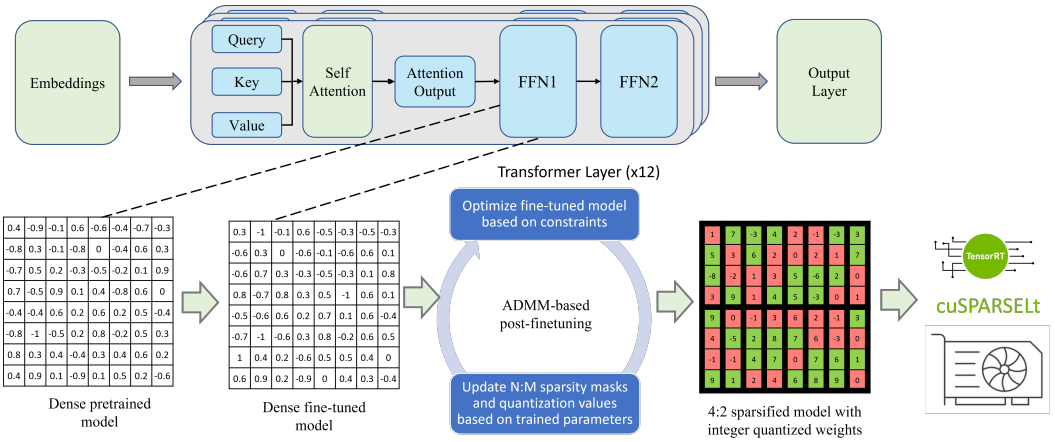


Figure 1: Overview. HoloFormer provides unified optimizations to induce N:M sparsity sparsity and integer quantization for compressing pre-trained Transformer networks. The resulting compressed model can be trivially converted to the deployment format for compatible hardware.

vast majority of inference wall-time. We exclude the embedding layer since the lookup operation associated with the embedding layers is incompatible for acceleration with Tensor Cores. The output layer consists of a fully connected layer, which is randomly initialized at the beginning of fine-tuning. We find that compressing this layer will unnecessarily harm accuracy. Since the execution of this layer is often less than 2% of the total execution time, we also exclude it from compression.

Solving N:M sparsity constraints. For the N:M sparsity constraint, the solution to Equation 5 is to retain M values of the contiguous group of N values in the Euclidean projection of $\mathbf{W}_i^{k+1} + \mathbf{U}_k$. Since ADMM does not explicitly provide a way to select those M values, we consider two strategies, both of which can be solved optimally in linear time. (i) We rely on the magnitude to determine whether a weight value should be pruned or not, i.e., we retain the M largest values out of N values. This is similar to magnitude-based pruning commonly used for DNN sparsification (Frankle & Carbin, 2019). (ii) We take a product of the absolute value of a parameter gradient momentum and the weight to measure individual parameter importance. This is similar to how prior work estimate filter importance for convolution networks (Molchanov et al., 2019). The intuition behind this strategy is that parameter importance should rely on not just magnitude but also statistics that reflect how these parameters are updated. We take the momentum (i.e., the exponential moving average of the gradient) instead of the stochastic gradient with each mini-batch for calculation because we would like to find weights that reduce the error consistently, rather than just for the most recent minibatch prior to the ADMM update. In our experiments, we observe that the magnitude-based strategy works surprisingly well across many tasks (See Section 4 for detailed analysis results).

Solving integer quantization constraints. For uniform integer quantization, we specifically formulate a quantized matrix as one whose values are contained can be represented by the following,

$$x_d = s_i * (x_q + o_i) \tag{6}$$

where x_d is the encoded value, s_i is the per-layer scaling factor, o_i is the per-layer offset, and x_q is the encoded integer representation. Under this scheme, an arbitrary floating point value may be quantized as in Equation7. The floating-point representation is divided by the encoding scale and then clamped to the range of the integer representation we are converting.

$$x_q = quantize(x, b, s, o) = clamp(round(x/s), -2^{b-1} + o, 2^{b-1} - 1 + o) \tag{7}$$

While this flexible definition of quantization introduces the least quantization error, using asymmetric quantization (where the offset is non-zero) introduces additional runtime overheads (Jacob et al., 2018). To avoid this cost, we use uniform symmetric quantization (Zafrir et al., 2019b) where the encoded integer range is evenly distributed around 0. In addition to reducing runtime overhead, this

approach also reduces ADMM optimization time, as it is only necessary to solve for the quantization scale s_i per-layer (Equation 8).

$$\min_{s_i, o_i} \|quantize(\mathbf{W}_i^{k+1} + \mathbf{U}_k, b, s_i, 0) - x\|_F^2 \tag{8}$$

For symmetric quantization, this can be easily solved with a linear scan beginning with a minimum viable scale since the optimization surface is smooth and convex, (The Appendix A.1 includes more details on the optimization surface for quantization).

It is also worth noting that to maximize inference performance, we quantize not only weights but also activations. For activations, we directly apply uniform symmetric quantization on activation tensors, solving for the scale by setting it to the smallest possible value such that the largest weight parameter can still be represented.

ADMM-base post-finetuning. For pre-trained language models, the primary purpose of fine-tuning is to adapt the model for a specific downstream task. Given that our primary goal of introducing ADMM is to optimize the network while satisfying constraints, we choose to perform the standard fine-tuning adaptation first to promote the model to achieve the desired accuracy for a task. Then we perform ADMM-based optimization as a post-finetuning step to specifically optimize the fine-tuned network to conform to the constraints. We find that when the constraints are complicated (e.g., the combinatorial constraints in our case), the post-finetuning optimization strategy provides better accuracy than directly applying ADMM-based optimizations while the model is still adapting to downstream tasks. We will include a more detailed analysis in Section 4.

Fused vs. unfused constraints. There is an additional degree of flexibility when performing the Euclidean projection with two sets of constraints. A specific weight can either (i) satisfy both N:M sparsity and quantization constraints simultaneously (i.e., fused constraints) in one ADMM iteration, such that a weight has the freedom to be either projected to zero or to the closest quantization value in one ADMM iteration, or we can (ii) let ADMM optimize these two constraints in two separate stages (i.e., unfused constraints), e.g., let HoloFormer perform N:M sparsification first, and then optimize for quantization on the remaining, non-zero weights. We investigate both of these two strategies. In practice, we observe that both strategies are able to achieve comparable accuracy. The unfused constraints take a longer time to optimize but are able to provide slightly better accuracy for some tasks. On the other hand, the fused constraints have a simpler optimization pipeline and faster optimization speed to achieve the same accuracy. See Section 4 for more analysis results.

4 EVALUATION

In this section, we evaluate HoloFormer and show its effectiveness in compressing pre-trained Transformer networks over a wide range of NLP tasks.

Implementation. HoloFormer is implemented as a PyTorch (Paszke et al., 2019) compatible library for unified optimization of Transformer models with N:M sparsity and integer quantization. Furthermore, we also implement a HuggingFace Transformers (Wolf et al., 2020) compatible Trainer to enable easy integration with their model collection and training scripts. Our approach supports different N:M sparse patterns (e.g., 4:1, 8:4) so long as weight's input dimension is a multiple of N, as well as different integer quantization bits (e.g., INT8, INT4). For evaluation, we focus on evaluating 4:2 sparsity and INT8 quantization since they are supported in commodity hardware. We use the pre-trained model checkpoint for BERT[1], provided by the HuggingFace model repository. All models were fine-tuned on an Intel Xeon 2630 v4 server with NVIDIA Titan V GPU running Ubuntu 18.04. PyTorch version 1.7.1 was used alongside Transformers 4.3.2. Fine-tuning these models required between 5 minutes (RTE) and 5 hours (MNLI), depending on task size.

Dataset. We evaluate HoloFormer and our baselines using the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018), a collection of NLP tasks varying in data availability and complexity. We report the Spearman correlation for STS-B, the F1 score for MRPC, Matthews correlation for CoLA, and accuracy for all remaining tasks. The reported average is the arithmetic mean of reported scores.

[1] `https://huggingface.co/bert-base-uncased`

4.1 MAIN RESULTS

In this section, we apply HoloFormer to GLUE and evaluate the effectiveness of the proposed approach by comparing the following schemes.

- **BERT** (Devlin et al., 2019): This is the $\text{BERT}_{\text{base}}$ model from publicly available checkpoint.
- **ASP + PTQ**: We first follow the practice in ASP (Mishra et al., 2021), by performing one-shot magnitude-based masked pruning on the pre-trained model to obtain a N:M sparsified model. Then we apply post-training quantization (PTQ) on the sparsified model to quantize model weights and activations to INT8.
- **ASP + QAT**: Similar to the above configuration, where we first perform ASP to learn N:M sparsity. Then we apply quantization-aware training (QAT) (Zafrir et al., 2019b) on the sparsified model to quantize model weights and activations to INT8.
- **HoloFormer**: This is our method to learn N:M sparsity plus INT8 quantized weights as described in Section 3.

Hyperparameters. In Devlin et al. (2019), the authors only report the development results on a few tasks. Therefore, we produce the BERT baseline results. We fine-tune BERT for 5 epochs on each downstream task, and the best-observed result on the validation set is reported. Our BERT baseline results are comparable to the results reported in the original paper. We perform a grid search of batch sizes 16 and 32, learning rates {1e-5, 3e-5, 5e-5, 7e-5, 9e-5} for all configurations. For HoloFormer, we additionally tune the penalty coefficient $\rho = \{$4e-4, 1e-3, 6e-3, 1e-2$\}$. We follow Devlin et al. (2019) to set all other training hyperparameters.

Table 1: The dev set results on the GLUE benchmark. The results show that HoloFormer is able to achieve higher accuracy than ASP + PTQ and ASP + QAT for integer quantized N:M sparsity, especially when the downstream tasks have low data resources.

	Task								Average
	MNLI (m/mm)	SST-2	QNLI	QQP	CoLA	STS-B	MRPC	RTE	
Samples	392k	67k	108k	368K	8.5k	5.7k	3.5k	2.5k	
Baseline ($\text{BERT}_{\text{base}}$)	84.5/84.8	92.5	90.9	90.9	57.5	89.0	89.8	66.7	82.6
ASP + PTQ	82.9/83.5	91.5	89.8	90.7	49.2	85	83.1	59.9	79.5
ASP + QAT	83.1/83.5	**92.6**	90	90.7	51.2	87	89.6	64.9	81.4
HoloFormer	**83.5/83.6**	92.4	**90.8**	**91.1**	**52.3**	**88.8**	**89.7**	**67.1**	**82.1**

We report the evaluation results for BERT in Table 1 and make the following observations. Existing techniques to induce N:M sparsity and integer quantization often result in accuracy losses for natural language understanding tasks. ASP + PTQ results in 3.1% accuracy drop as compared to the uncompressed model ($\text{BERT}_{\text{base}}$). We analyze the results and identify two reasons for this accuracy loss. First, ASP sparsifies the pre-trained Transformer model but cannot fully close the accuracy gap between the sparsified model and the uncompressed baseline model (shown later in the ablation analysis section). Second, PTQ quantizes the sparsified model weights and activations directly without further training involved. Therefore the model is executed without any adaption to the quantization process, which hurts the accuracy. On the other hand, by performing ASP + QAT, we are able to obtain better results than ASP + PTQ. This is because in PTQ the quantization operations are inserted into the model before fine-tuning. Therefore, the model is able to adapt to the quantized weights and activations while fine-tuning. Despite showing improved performance, ASP + QAT still leads to a drop of 1.2 points in accuracy of the uncompressed model.

HoloFormer consistently outperforms the baseline methods on every GLUE task and achieves an average score of 82.1, outperforming ASP + PTQ and ASP + QAT by 2.6 and 0.7 points, respectively, indicating that HoloFormer can effectively induce N:M sparsity and integer quantization. Notably, HoloFormer achieves 0.8 points higher accuracy for MNLI, 1.1 points higher accuracy for CoLA, 1.8 points higher accuracy for STS-B, and 2.2 points higher accuracy for RTE. Overall, HoloFormer retains 99.4% of the accuracy of the uncompressed model. These results suggest that a principled and unified optimization approach that takes both the N:M sparsity constraints and integer quantization constraints into account would yield high accuracy results. More importantly, our method can accelerate inference by effectively leveraging the underlying Sparse Tensor Core.

4.2 ABLATION STUDIES

In this section, we perform detailed ablation studies of HoloFormer to investigate its effectiveness.

Comparison to pure sparsification. We compare HoloFormer with methods that learn sparsity only. (i) $\text{ADMM}_{\text{Unstructured}}$: We create a baseline that uses ADMM but induces unstructured sparsity at 50% per-layer sparsity. (ii) **ASP**: We perform one-shot magnitude-based masked pruning for N:M sparsity on the pretrained model (Mishra et al., 2021). (iii) HoloFormer (N:M sparsity only): We use ADMM to induce N:M sparsity (i.e., with just the constraint set C^1 in Section 3.1).

Table 2: Comparison among different sparsification methods.

	MNLI (m/mm)	SST-2	QNLI	CoLA	STS-B	MRPC	RTE	Avg
$\text{ADMM}_{\text{Unstructured}}$	84.0/84.7	92.5	91.0	57.5	89.6	90.5	68.2	82.1
ASP	83.5/83.6	91.9	90.6	51.7	88.7	88.1	63.9	80.0
HoloFormer (N:M sparsity only)	**84.1/84.2**	**92.3**	**90.9**	**55.3**	**89.3**	**90.8**	**68.6**	**81.7**

Table 2 compares the accuracy of different sparsification methods. First, unstructured pruning (i.e.,$\text{ADMM}_{\text{Unstructured}}$) sparsifies weights of Transformer blocks but does not explicitly optimize against the underlying hardware constraints, e.g., the N:M sparsity. As a result, although it preserves the highest accuracy on downstream tasks (82.1 on average), the obtained model has a weight structure that consists of random non-zero weights. Such an irregular pattern makes it very inefficient to execute in modern hardware systems. As a result, the performance benefit with unstructured sparsity-based approaches is negligible, even when the pruning rate is high (e.g., 95%) (Wen et al., 2016b). Second, HoloFormer with N:M sparsity only achieves an average score of 81.7, outperforming ASP by 1.7 points. HoloFormer offers higher sparsification accuracy than ASP, because it explicitly optimizes the model while taking the hardware sparsity constraints into account. Thus, HoloFormer sparsification achieves 99.5% of the accuracy of the unstructured sparsity. Different from $\text{ADMM}_{\text{Unstructured}}$, which suffers from expensive irregular memory accesses, our method allows effective use of the underlying Sparse Tensor Core to achieve inference speedups.

Comparison with pure quantization. In this part, we compare HoloFormer with methods that perform integer-quantization only. Existing methods could compress models to use fewer bits such as ternary or even binary weights (Shen et al., 2020; Bai et al., 2020). However, lower-bit quantized execution is not supported in commodity hardware. Therefore, we focus on the evaluation of INT8 quantization results. Table 3 reports accuracy from using different methods for quantization. We quantize our baseline models using quantization-aware training (QAT) (Zafrir et al., 2019b), where we use fake quantization to quantize the weights and activations to INT8 for inference, and during back-propagation we use full precision weights. We then use our method to induce INT8 quantized weights (i.e., with just the constraint set C^2 in Section 3.1) and activations (HoloFormer INT8 Only). The results show that the 8-bit quantization from QAT can achieve comparable accuracy as the baseline uncompressed model on some tasks as MNLI, QNLI, QPP, and STS-B. However, the accuracy degradation can be large on some small datasets such as CoLA, MRPC, and RTE. In contrast, our ADMM-based quantization method is able to obtain higher accuracy than QAT on 6 out of 8 tasks, achieving 0.8 points higher accuracy on average (82.2 vs. 81.4) and retaining 99.5% of the baseline model accuracy. HoloFormer (INT8 only) achieves better accuracy because it can also be viewed as a variant of quantization aware training. Different from basic QAT, it explicitly takes quantization constraints into account when optimizing the model and takes a dynamic regularization procedure, where the regularization target is dynamically updated in each iteration.

Table 3: Comparison among different integer quantization methods.

	MNLI (m/mm)	SST-2	QNLI	QQP	CoLA	STS-B	MRPC	RTE	Avg
Baseline ($\text{BERT}_{\text{base}}$)	84.5	92.5	90.9	90.9	56.7	89.2	90.6	65.3	82.6
QAT	**84**	91.5	**90.9**	90.6	52.8	**88.9**	88.3	64.2	81.4
HoloFormer (INT8 only)	83.9	**92.2**	90.8	**91.1**	**53.4**	**88.9**	**89.4**	**68.2**	**82.2**

Effect of different importance estimation for N:M sparsity. As discussed in Section 3, we compare the impact of two importance estimation strategies for solving N:M sparsity constraints: i) magnitude-based vs. ii) a gradient-based method that takes the product of the absolute value of weight and momentum as an estimate of importance. Table 4 shows that the two importance estimation strategies lead to comparable accuracy. In fact, the magnitude-based strategy works surprisingly well across many tasks and leads to slightly better performance than the gradient-based method. For this reason, we keep the magnitude-based ranking method as default but tuning this strategy might lead to some improvements (e.g., QQP). More advanced techniques such as learning-based methods or methods based on higher-order curvature information could be used to improve further the performance, which we leave for future exploration.

Impact of direct vs. post ADMM-based fine-tuning. In Section 3.2, we discussed the workflow of ADMM-based post-finetuning. In this part, we evaluate the effect of performing ADMM-based optimization directly against a pre-trained Transformer network vs. as a post-finetuning step against fine-tuned models. While both strategies could yield models that satisfy hardware constraints, Table 5 shows that performing ADMM against fine-tuned checkpoints after the pre-trained model has led to consistently better accuracy on downstream tasks.

Table 4: Evaluation results of different importance ranking strategies.

	Magnitude-based	Grad-based
MNLI-m	84	83.3
MNLI-mm	84.4	84
SST2	92.3	92.2
QNLI	91.1	90.6
QQP	91	91.1
Avg	**88.5**	88.2

Table 5: Evaluation results of direct vs. post ADMM-based finetuning.

	Direct-finetuning	Post-finetuning
MNLI-m	82.3	84.1
MNLI-mm	83.4	84.2
SST2	92.3	92.3
QNLI	90.4	90.9
QQP	90.9	91.1
Avg	87.8	**88.5**

Effect of fused vs. unfused constraints. Table 6 shows the comparison results of HoloFormer with fused constraints (HoloFormer-fused) vs. unfused constraints (HoloFormer-unfused). For the unfused case, we first perform ADMM-based fine-tuning for 5 epochs until the model converges to learn the N:M sparsity. Then we apply ADMM-based integer quantization for another 5 epochs to learn the integer quantized sparse model. We observe that HoloFormer-fused and HoloFormer-unfused achieve comparable accuracy for most tasks. However, overall HoloFormer-unfused achieves slightly better accuracy on MNLI-m, QNLI, QQP, and STS-B, but it also takes a longer time to converge. On the other hand, HoloFormer-fused converges faster while achieving similar results on several tasks.

Table 6: Performance of HoloFormer with fused vs. unfused constraints.

	MNLI (m/mm)	SST-2	QNLI	QQP	CoLA	STS-B	Avg
HoloFormer-Fused	83.1/83.8	92.4	90.7	90.9	52.3	88.3	83.0
HoloFormer-Unfused	83.5/83.6	92.2	90.8	91.1	52.3	88.8	83.2

5 CONCLUSION

In this work, we present a unified optimization method to learn N:M sparsity and integer quantization using ADMM for compressing pre-trained Transformer networks. Our results over a wide range of NLP tasks indicate that HoloFormer is an effective method for obtaining models with N:M sparsity and quantization, outperforming existing methods that perform sparsification or quantization individually. The model learned through HoloFormer can effectively improve runtime resource efficiency without large penalties in model accuracy. Finally, the simplicity of our method is also worth reemphasizing. HoloFormer has only one additional hyperparameter to optimize for both N:M sparsity and integer quantization and works sufficiently well on the benchmark datasets with the optimization of the penalty coefficient and learning rate.

REFERENCES

Nvidia a100 tensor core gpu architecture, 2020. URL https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf.

Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jing Jin, Xin Jiang, Qun Liu, Michael R. Lyu, and Irwin King. Binarybert: Pushing the limit of BERT quantization. *CoRR*, abs/2012.15701, 2020.

Stephen P. Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1):1–122, 2011.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

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.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://www.aclweb.org/anthology/N19-1423.

Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. HAWQ-V2: hessian aware trace-weighted quantization of neural networks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

Mitchell Gordon, Kevin Duh, and Nicholas Andrews. Compressing BERT: Studying the effects of weight pruning on transfer learning. In *Proceedings of the 5th Workshop on Representation Learning for NLP*, pp. 143–155, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.repl4nlp-1.18. URL https://www.aclweb.org/anthology/2020.repl4nlp-1.18.

Fu-Ming Guo, Sijia Liu, Finlay S. Mungall, Xue Lin, and Yanzhi Wang. Reweighted proximal pruning for large-scale language representation. *CoRR*, abs/1909.12486, 2019a.

Fu-Ming Guo, Sijia Liu, Finlay S. Mungall, Xue Lin, and Yanzhi Wang. Reweighted proximal pruning for large-scale language representation. *CoRR*, abs/1909.12486, 2019b. URL http://arxiv.org/abs/1909.12486.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark Horowitz, and Bill Dally. Deep compression and EIE: efficient inference engine on compressed deep neural network. In *2016 IEEE Hot Chips 28 Symposium (HCS), Cupertino, CA, USA, August 21-23, 2016*, pp. 1–6. IEEE, 2016.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 4163–4174, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.372. URL https://www.aclweb.org/anthology/2020.findings-emnlp.372.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020. URL https://arxiv.org/abs/2001.08361.

Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. I-BERT: integer-only BERT quantization. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 5506–5518. PMLR, 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015. URL http://arxiv.org/abs/1412.6980.

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. *CoRR*, abs/1907.11692, 2019.

Zejian Liu, Gang Li, and Jian Cheng. Hardware acceleration of fully quantized BERT for efficient natural language processing. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*, pp. 513–516. IEEE, 2021. doi: 10.23919/DATE51398.2021.9474043.

Xiaolong Ma, Fu-Ming Guo, Wei Niu, Xue Lin, Jian Tang, Kaisheng Ma, Bin Ren, and Yanzhi Wang. Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):5117–5124, Apr. 2020. doi: 10.1609/aaai.v34i04.5954. URL https://ojs.aaai.org/index.php/AAAI/article/view/5954.

Yihuan Mao, Yujing Wang, Chufan Wu, Chen Zhang, Yang Wang, Quanlu Zhang, Yaming Yang, Yunhai Tong, and Jing Bai. Ladabert: Lightweight adaptation of BERT through hybrid model compression. In Donia Scott, Núria Bel, and Chengqing Zong (eds.), *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020*, pp. 3225–3234. International Committee on Computational Linguistics, 2020.

Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/2c601ad9d2ff9bc8b282670cdd54f69f-Paper.pdf.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks, 2021.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pp. 11264–11272. Computer Vision Foundation / IEEE, 2019.

Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. Patdnn: Achieving real-time DNN execution on mobile devices with pattern-based weight pruning. In James R. Larus, Luis Ceze, and Karin Strauss (eds.), *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020*, pp. 907–922. ACM, 2020.

Hua Ouyang, Niao He, Long Tran, and Alexander G. Gray. Stochastic alternating direction method of multipliers. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pp. 80–88. JMLR.org, 2013.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019. URL http://arxiv.org/abs/1910.10683.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. ArXiv, October 2019.

Ao Ren, Tianyun Zhang, Shaokai Ye, Jiayu Li, Wenyao Xu, Xuehai Qian, Xue Lin, and Yanzhi Wang. Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, pp. 925–938, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362405. doi: 10.1145/3297858.3304076. URL https://doi.org/10.1145/3297858.3304076.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108, 2019.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. Q-BERT: hessian based ultra low precision quantization of BERT. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 8815–8821. AAAI Press, 2020.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices. In *ACL (2020)*, 2020. URL https://arxiv.org/abs/2004.02984.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel 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*, pp. 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5446. URL https://www.aclweb.org/anthology/W18-5446.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 2074–2082, 2016a.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 2074–2082, 2016b.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 5754–5764, 2019.

Shaokai Ye, Tianyun Zhang, Kaiqi Zhang, Jiayu Li, Jiaming Xie, Yun Liang, Sijia Liu, Xue Lin, and Yanzhi Wang. A unified framework of DNN weight pruning and weight clustering/quantization using ADMM. *CoRR*, abs/1811.01907, 2018a.

Shaokai Ye, Tianyun Zhang, Kaiqi Zhang, Jiayu Li, Kaidi Xu, Yunfei Yang, Fuxun Yu, Jian Tang, Makan Fardad, Sijia Liu, Xiang Chen, Xue Lin, and Yanzhi Wang. Progressive weight pruning of deep neural networks using ADMM. *CoRR*, abs/1810.07378, 2018b. URL http://arxiv.org/abs/1810.07378.

Shaokai Ye, Xiaoyu Feng, Tianyun Zhang, Xiaolong Ma, Sheng Lin, Zhengang Li, Kaidi Xu, Wujie Wen, Sijia Liu, Jian Tang, Makan Fardad, Xue Lin, Yongpan Liu, and Yanzhi Wang. Progressive DNN compression: A key to achieve ultra-high weight pruning and quantization rates using ADMM. *CoRR*, abs/1903.09769, 2019.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8BERT: quantized 8bit BERT. *CoRR*, abs/1910.06188, 2019a.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8BERT: quantized 8bit BERT. *CoRR*, abs/1910.06188, 2019b. URL http://arxiv.org/abs/1910.06188.

Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning n:m fine-grained structured sparse neural networks from scratch. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=K9bw7vqp_s.

A Additional Analysis Results

A.1 Integer Quantization Surface

In Section 3, we mention that for quantization constraints, we can solve through a linear scan. Figure 2 plots the optimization surface, which is smooth and convex. Given this property, we can easily solve the quantization constraints with a linear scan.

A.2 Effect of penalty coefficient ρ.

Another advantage of HoloFormer is that the algorithm only introduces one additional hyperparameter ρ. Our experiments suggest that, in most cases, HoloFormer can achieve satisfactory performance with the optimization of ρ and the learning rates alone. Figure 3 shows the effect of the ADMM penalty coefficient ρ and learning rates on downstream task MNLI and RTE, which correspond to the domains that have a relatively high resource (392K) vs. low resources (2.5K). In both cases, we observe that the algorithm performs optimally when $\rho = 4e-4, 1e-3$. Small tasks appear to require a slightly larger ρ. This result is not too unsurprising. For smaller datasets, the model may be subject more to overfitting. Therefore, a larger ρ introduces a stronger regularization effect that helps HoloFormer generalize better while satisfying constraints. Increasing ρ further by order of magnitude (e.g., $1e-2$) doe not leads to better accuracy. Therefore, we recommend on an empirical basis that the user prioritizes the parameter search for ρ in the $[1e-4, 1e-3]$ range.

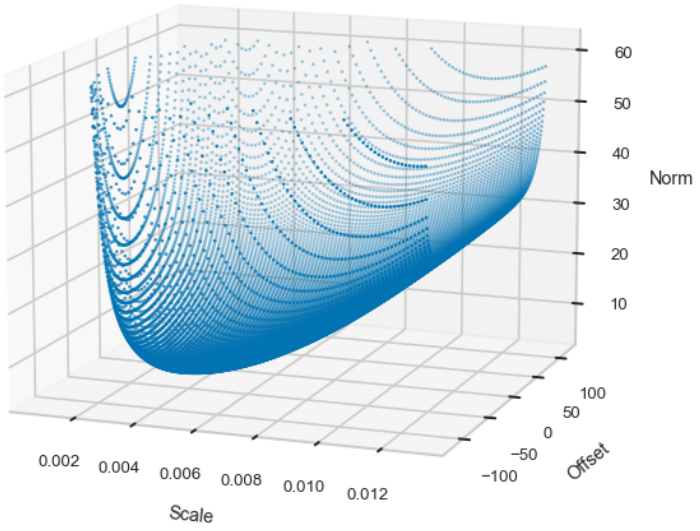
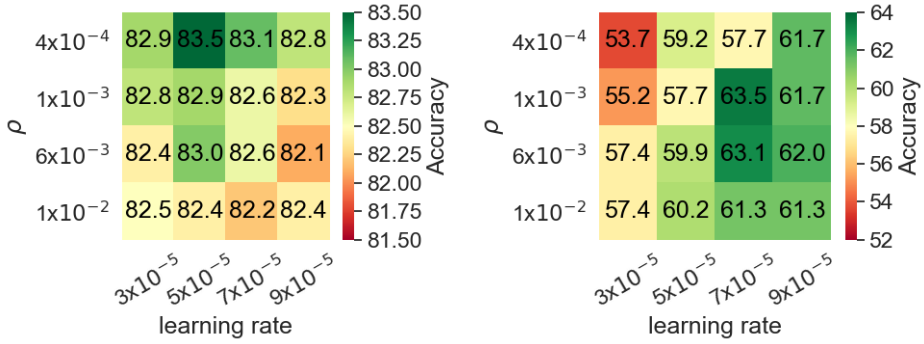


Figure 2: Visualization of optimization surface under asymmetric quantization constraints. Along both the offset and scale axes, we have smooth parabolas.



(a) MNLI-m (b) RTE

Figure 3: Effect of the ADMM penalty coefficient ρ.