The Optimal BERT Surgeon: Scalable and Accurate Second-Order Pruning for Large Language Models

Eldar Kurtic* [1], Daniel Campos[2,3], Tuan Nguyen[2], Elias Frantar[1], Mark Kurtz[2], Benjamin Fineran[2], Michael Goin[2], and Dan Alistarh[1,2]

[1]*Institute of Science and Technology Austria*

[2]*Neural Magic Inc.*

[3]*Department of Computer Science, University of Illinois Urbana-Champaign*

Abstract

In this paper, we consider the problem of sparsifying BERT models, which are a key building block for natural language processing, in order to reduce their storage and computational cost. We introduce the *Optimal BERT Surgeon* (oBERT), an efficient and accurate pruning method based on approximate second-order information, which we show to yield state-of-the-art results for compression in both stages of language tasks: pre-training and fine-tuning. Specifically, oBERT extends existing work on second-order pruning by allowing for pruning blocks of weights, and is the first such method that is applicable at BERT scale. Second, we investigate *compounding* compression approaches to obtain highly compressed but accurate models for deployment on edge devices. These models significantly push boundaries of the current state-of-the-art sparse BERT models with respect to all metrics: model size, inference speed and task accuracy. For example, relative to the dense $\text{BERT}_{\text{BASE}}$, we obtain 10x model size compression with $< 1\%$ accuracy drop, 10x CPU-inference speedup with $< 2\%$ accuracy drop, and 29x CPU-inference speedup with $< 7.5\%$ accuracy drop. Our code, fully integrated with Transformers and SparseML, is available at `https://github.com/neuralmagic/sparseml/tree/main/research/optimal_BERT_surgeon_oBERT`.

1 Introduction

Pre-trained Transformer models (Vaswani et al., 2017; Devlin et al., 2019) provide robust language representations which can be specialized on various tasks. Given their massive growth (Radford et al., 2019; Smith et al., 2022), techniques for reducing their computational overheads have become popular. One classic technique is Knowledge Distillation (KD) (Hinton et al., 2015), which transfers knowledge from a larger teacher to a smaller student model. Other work has leveraged lower-precision representations to produce quantized models. An orthogonal approach, which is our primary focus, has been to apply unstructured and block pruning, i.e. removing individual weights, to produce compressed but accurate language models. Figure 1 provides a comparative overview of state-of-the-art results for unstructured pruning.

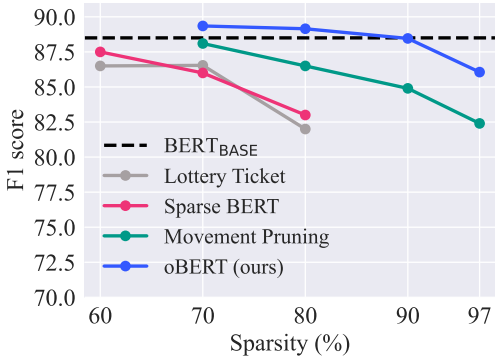

Figure 1: Performance overview relative to state-of-the-art unstructured downstream pruning methods Chen et al. (2020), Xu et al. (2021), Sanh et al. (2020), in this order, of the $\text{BERT}_{\text{BASE}}$ model on the SQuADv1.1 task.

In this paper, we introduce a method for improved unstructured and semi-structured (block) pruning, by leveraging the second-order approach pioneered by the Optimal Brain Surgeon framework (LeCun et al., 1989; Hassibi and Stork, 1992), which we scale for the first time to LLMs. Further, we put our results in the context of a compound compression approach, which combines several compression techniques to obtain sparse models which we execute on a sparsity-aware CPU-based runtime (NeuralMagic, 2021), showing order-of-magnitude speedups at low accuracy loss.

In summary, our contributions are as follows:

- We perform a thorough exploration of weight pruning approaches applied to LLMs,

* Corresponding author: eldar.kurtic@ist.ac.at.

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 4163 - 4181

including lottery-ticket, movement pruning, magnitude and second-order pruning.

- We introduce a general second-order pruning method called *Optimal BERT Surgeon* (oBERT), which supports unstructured and block pruning, and is the first second-order method to be both highly-accurate and scalable to the dimensionality of BERT models.
- We illustrate the benefits of oBERT by significantly improving upon existing state-of-the-art pruning methods, in both stages of language tasks: pre-training and fine-tuning. For illustration, when pruning $\text{BERT}_{\text{BASE}}$, oBERT outperforms Movement Pruning (MvP), the most accurate prior approach, by more than 2% absolute F1 score at the same sparsity, and can match the accuracy of MvP models with 3x fewer parameters.
- We investigate the applicability of this pruning method in a framework which *compounds* popular compression approaches for LLMs, i.e. applying pruning in combination with layer dropping and/or quantization. In this context, we show that our resulting sparse models provide order-of-magnitude improvements compared to other compound compressed models, and that they can be easily deployed for CPU inference.

2 Background and Related Work

Transformer LLMs are usually built using multiple transformer layers with self-attention (Vaswani et al., 2017). Each transformer has a variation of two sub-components: multi head attention (MHA) and fully connected feed forward network (FFN). Given the massive size of well-performing models, there has been growing interest in LLM compression. They have been shown to be fragile as minor perturbations can lead to model collapse (Kovaleva et al., 2021). Pruning schemes are motivated by weight saliency metrics which represent the loss in accuracy due to pruning. It is common to prune in iterative steps, each of which removes weights until a desired sparsity level is reached. Now, we briefly overview existing approaches.

Structured pruning for LLMs focuses on reducing the number of layers and/or attention heads, and requires structural understanding of the model. Michel et al. (2019) and Voita et al. (2019) demonstrated that for some tasks nearly 40% of attention heads can be removed without major impact on accuracy. Other work has focused on removing layers (Sridhar and Sarah, 2020), and on the order in which they are removed (Sajjad et al., 2020). In some of our experiments, we apply standard "direct" layer dropping in conjunction with pruning.

Semi-structured pruning is an intermediate approach, by which smaller groups, e.g. rectangular sets of weights (Lagunas et al., 2021), are set to zero. This approach has recently gained in popularity thanks to efficient computational support. We extend the second-order pruning formulation to such groupings, and show results for a specific grouping supported by a CPU-inference engine.

Unstructured pruning removes individual weights by setting them to zero. Gradual Magnitude Pruning (GMP) is a classic approach, which makes use of weight magnitudes as a saliency metric for pruning (Han et al., 2015; Gale et al., 2019).

First-order pruning methods use a gradient based formulation of the saliency metric. A popular method is **Movement Pruning (MvP)** (Sanh et al., 2020), specifically designed for pruning in the fine-tuning stage. Intuitively, it removes weights that are moving towards zero. The resulting models were the first to achieve high sparsity with tolerable accuracy loss. Methods such as PLATON (Zhang et al., 2022) attempt to capture the uncertainty of weights importance scores by upper confidence bound estimation. Prior to our work, MvP and PLATON approaches set state-of-the-art results for unstructured pruning.

Second-order pruning methods (LeCun et al., 1989; Hassibi and Stork, 1992; Singh and Alistarh, 2020; Frantar et al., 2021) were developed in the context of image classification, and leverage complex approximations of the loss curvature. However, second-order pruning methods require an approximation of the inverse Hessian, which is expensive to store and compute with for LLM parameter counts. The approach we propose is similar to WoodFisher/M-FAC methods (Singh and Alistarh, 2020; Frantar et al., 2021), but is the first to work accurately at LLM scale. Specifically, the WoodFisher approach is infeasible at BERT scale, as it requires storing gradients for inverse Fisher calculation in memory at the point of pruning. The M-FAC approach scales, but we show that its parametrization yields worse pruning results (Appendix Figure 3). This is because

M-FAC performs full-matrix (non-blocked) inversion by default, which is inherently noisy. In addition, we extend the theoretical OBS approach to semi-structured (block) compression. We also show that our method can be applied during LLM pre-training and fine-tuning, yielding state-of-the-art results in both regimes.

Knowledge Distillation (Hinton et al., 2015) trains a smaller student model against outputs of a larger teacher model by adding a loss component which minimizes the KL-divergence between the two output distributions, which is the approach we adopt in our setup too. A hardness parameter is used to control the mixture of regular and distillation loss, and a temperature parameter to control softness of the distribution. Contrary to this, approaches like DistilBERT (Sanh et al., 2019), TinyBERT (Jiao et al., 2020), MobileBERT (Sun et al., 2020a), and MiniLM (Wang et al., 2020) utilize more complex distillation schemes, based on transferring knowledge from intermediate model's representations. Our sparse models provide order-of-magnitude improvements upon some of these methods.

Quantization represents weights and activations in lower precision (Courbariaux et al., 2016), and was used to obtain models such as Q8BERT (Zafrir et al., 2019) and TernaryBERT (Zhang et al., 2020).

Shen et al. (2020) uses information about the Hessian spectrum to choose quantization bit-widths, whereas Yu et al. (2022) uses an approximation of the Hessian trace for structured pruning. These Hessian-based approaches are different from the one we propose, as we use completely different inverse-Hessian approximations to guide pruning decisions. The focus of our work is on *weight pruning*, and on computational speedups achievable on commodity CPUs. As such, the methods we investigate are orthogonal to quantization. Moreover, it is impossible to directly compare to low-bitwidth quantized models as most inference frameworks do not support such custom formats. Therefore, we will only make use of the standard Quantization-Aware Training (QAT) to 8-bit weights, which is well-supported on Intel CPUs, and showcase the resulting speedups in conjunction with layer dropping and weight pruning.

Downstream compression methods attempt to compress directly while fine-tuning on a specific task. MvP method is specially designed for this setup. **Upstream compression** methods compress during the pre-training phase, reducing the need for task-specific pruning. Chen et al. (2020) examined the "Lottery Ticket" strategies (Frankle and Carbin, 2018) which, as we illustrate later, incur huge accuracy loss even at moderate sparsities. Recent work "Prune Once for All" (Prune OFA) by Zafrir et al. (2021) showed that well-tuned magnitude pruning can be competitive with downstream methods like MvP.

We first examine the performance of prior pruning methods, notably MvP, Prune OFA, and Lottery Tickets, relative to the new second-order oBERT method. The approach we propose consistently improves upon all these prior methods, both in pre-training (upstream) and fine-tuning (downstream) stages, and can be compounded with other compression techniques to obtain models that are smaller, faster and more accurate than models like DistilBERT, TinyBERT, and block MvP.

Additional approaches for efficient inference of LLMs exist, like token-pruning and early-exiting. These approaches are *orthogonal* to ours; therefore we discuss them in Appendix A.2.

3 The Optimal BERT Surgeon (oBERT)

3.1 Generalized Second-Order Block Pruning

The pruning problem starts from a well-optimized dense model $\mathbf{w}^* \in \mathbb{R}^d$, and aims to find a sparse version of $\mathbf{w}^*$, where many of the weights are set to zero, and the remaining weights may be updated accordingly in order to preserve the loss. It is common for this process to occur gradually, i.e. by progressively removing the weights. A classic approach (LeCun et al., 1989; Hassibi and Stork, 1992) for "optimal" pruning of weights from $\mathbf{w}^*$ at a step is to expand the loss function $\mathcal{L}$ locally around $\mathbf{w}^*$ with respect to a sparse 0/1 weight mask $\mathbf{M}$. If we denote by $\mathbf{w}_M = (\mathbf{M} \odot \mathbf{w}^*)$, the model resulting from the Hadamard (element-wise) product between $\mathbf{M} \in \{0,1\}^d$ and $\mathbf{w}^*$, we can use the Taylor expansion at $\mathbf{w}_M$ to obtain:

$$\mathcal{L}(\mathbf{w}_M) \simeq \mathcal{L}(\mathbf{w}^*) + (\mathbf{w}_M - \mathbf{w}^*)^\top \nabla \mathcal{L}(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w}_M - \mathbf{w}^*)^\top \mathbf{H}_\mathcal{L}(\mathbf{w}^*)(\mathbf{w}_M - \mathbf{w}^*).$$

Given that $\mathbf{w}^*$ is well-optimized, it is reasonable in practice to assume that $\nabla \mathcal{L}(\mathbf{w}^*) \approx \mathbf{0}$. Then, the change in loss incurred by pruning a subset of

weights can be expressed as

$$\delta\mathcal{L}(\delta\mathbf{w}) \simeq \frac{1}{2}\delta\mathbf{w}^\top \mathbf{H}_\mathcal{L}(\mathbf{w}^*)\delta\mathbf{w} \tag{1}$$

where $\delta\mathcal{L}(\delta\mathbf{w}) \coloneqq \mathcal{L}(\mathbf{w}_M) - \mathcal{L}(\mathbf{w}^*)$ and $\delta\mathbf{w} \coloneqq \mathbf{w}_M - \mathbf{w}^*$. A popular way of approximating the Hessian at $\mathbf{w}^*$ is via a dampened empirical Fisher information matrix (Hassibi and Stork, 1992):

$$\mathbf{H}_\mathcal{L}(\mathbf{w}) \simeq \widehat{\mathbf{F}}(\mathbf{w}) = \lambda\mathbf{I}_d + \frac{1}{m}\sum_{i=1}^{m}\nabla\mathcal{L}_i(\mathbf{w})\nabla\mathcal{L}_i^\top(\mathbf{w}), \tag{2}$$

where $\lambda \geq 0$ is a small dampening constant, $\mathbf{I}_d \in \mathbb{R}^{d\times d}$ identity matrix and m is the number of gradient outer products used to approximate the Hessian. Given the positive-definiteness of (2), the quadratic form (1) is always nonnegative which is why we will refer to $\delta\mathcal{L}(\delta\mathbf{w})$ as a *loss increase* incurred by pruning.

Returning to our pruning problem, assume we wish to identify a block of weights Q of a given shape whose removal by zero-masking would incur minimum increase in loss. This leads to the following constrained optimization problem:

$$\begin{aligned} \min_{\delta\mathbf{w}} \quad & \frac{1}{2}\delta\mathbf{w}^\top\widehat{\mathbf{F}}(\mathbf{w}^*)\delta\mathbf{w} \\ \text{s.t.} \quad & \mathbf{e}_k^\top\delta\mathbf{w} + w_k = 0, \quad \forall k \in \mathbf{Q} \end{aligned} \tag{3}$$

where $\mathbf{e}_k \in \mathbb{R}^d$ stands for the k-th canonical basis vector. Here, we will provide a generalized solution, which applies to general Q. First, for convenience, we express the system of $|\mathbf{Q}|$ equality constraints in matrix-equation form as $\mathbf{E}_\mathbf{Q}\delta\mathbf{w} + \mathbf{E}_\mathbf{Q}\mathbf{w}^* = \mathbf{0}$, where $\mathbf{E}_\mathbf{Q} \in \mathbb{R}^{|\mathbf{Q}|\times d}$ is a matrix composed of the corresponding canonical basis vectors $\mathbf{e}_k$ $(\forall k \in \mathbf{Q})$ arranged in rows. This optimization problem can be solved with the method of Lagrange multipliers. Specifically, we wish to find stationary points of the Lagrangian $L(\delta\mathbf{w}, \boldsymbol{\lambda})$, where $\boldsymbol{\lambda} \in \mathbb{R}^{|\mathbf{Q}|}$ denotes a vector of Lagrange multipliers. Solving the system of equations $\frac{\partial L(\delta\mathbf{w},\boldsymbol{\lambda})}{\partial\delta\mathbf{w}} = \mathbf{0}$ and $\frac{\partial L(\delta\mathbf{w},\boldsymbol{\lambda})}{\partial\boldsymbol{\lambda}} = \mathbf{0}$ yields the following optimal weight update:

$$\delta\mathbf{w}^* = -\widehat{\mathbf{F}}^{-1}(\mathbf{w}^*)\mathbf{E}_\mathbf{Q}^\top\left(\mathbf{E}_\mathbf{Q}\widehat{\mathbf{F}}^{-1}(\mathbf{w}^*)\mathbf{E}_\mathbf{Q}^\top\right)^{-1}\mathbf{E}_\mathbf{Q}\mathbf{w}^*$$

which prunes a set of weights Q and updates the remaining weights to preserve the loss. Now, the corresponding loss increase incurred by the optimal weight update $\delta\mathbf{w}^*$ can be expressed as the saliency score of weights Q, which we denote by:

$$\rho_\mathbf{Q} = \frac{1}{2}\left(\mathbf{E}_\mathbf{Q}\mathbf{w}^*\right)^\top\left(\mathbf{E}_\mathbf{Q}\widehat{\mathbf{F}}^{-1}(\mathbf{w}^*)\mathbf{E}_\mathbf{Q}^\top\right)^{-1}\mathbf{E}_\mathbf{Q}\mathbf{w}^*.$$

We use this saliency/importance score to rank groups of weights for pruning. As a sanity check, if we prune a single weight w_j at a time, our derivations will yield the standard formulas of (Hassibi and Stork, 1992). The full version of Singh and Alistarh (2020) provided a slightly less general derivation for the blocked case, under additional assumptions.

3.2 An Efficient Implementation

Directly implementing the previously described approach for LLMs, where number of weights $\mathbf{w} \in \mathbb{R}^d$ is huge, is infeasible. In particular, this is due to the dependence on the inverse of the empirical Fisher information matrix $\widehat{\mathbf{F}}^{-1}(\mathbf{w}) \in \mathbb{R}^{d\times d}$, appearing in formulations of the saliency score and of the optimal weight update. We now describe how to circumvent these issues.

3.2.1 Pruning the optimal set of weights

Assume a gradual pruning setup, in which at each pruning step we wish to prune a model to a target sparsity $s \in (0, 1]$, effectively zeroing out $s \times d$ weights, in groups of size $|Q|$. Typically $s \times d \gg |Q|$, meaning that we want to remove multiple groups at the same time. Finding the optimal set of $\frac{s\times d}{|Q|}$ groups is an intractable combinatorial problem, due to all possible correlations between them, given by the binomial coefficient $\binom{n}{k}$, where $n = \frac{d}{|Q|}$ and $k = \frac{s\times d}{|Q|}$. This problem can be alleviated by ignoring correlations between different groups of weights Q, and solving only for correlations between the weights within the same group. In practice, this boils down to evaluating the saliency score $\rho_\mathbf{Q}$ for each group Q, and pruning the $\frac{s\times d}{|Q|}$ groups with the lowest score. As pruning many weights in the same step can make the Taylor approximation of the loss function less accurate, one can consider pruning with multiple smaller sub-steps with recomputations of the Hessian approximation in between (without intermediate fine-tuning). While this can further improve the quality of the pruning step (Frantar et al., 2021), we do not implement this additional optimization since the competing methods do not utilize recomputations.

3.2.2 Inverse empirical Fisher computation

The key space and time complexity cost of the above procedure is computing products with the inverse empirical Fisher. A direct approach would be to perform a block-wise diagonal approximation of this matrix (which we detail next), and perform direct block inversion. However, we found experimentally that this approach is too expensive in terms of time, and quite numerically-sensitive. As an alternative, we rely on the fact that the matrix we wish to invert is a sum of rank-1 matrices, and employ the Woodbury/Sherman-Morrison (WSM) inversion formula. Specifically, given a sum $(\mathbf{A} + \mathbf{u}\mathbf{v}^\top)$ of an invertible matrix $\mathbf{A}$ and an outer product of vectors $\mathbf{u}$ and $\mathbf{v}$ with compatible dimensions, the inverse $(\mathbf{A} + \mathbf{u}\mathbf{v}^\top)^{-1}$ can be exactly calculated as $\mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^\top\mathbf{A}^{-1}}{1+\mathbf{v}^\top\mathbf{A}^{-1}\mathbf{u}}$. Placing the expression of the empirical Fisher in the WSM formula, we obtain the following recursive formulation, where m is the number of gradients employed in the approximation:

$$\widehat{\mathbf{F}}^{-1}(\mathbf{w}) = \widehat{\mathbf{F}}_m^{-1}(\mathbf{w}) = \left(\widehat{\mathbf{F}}_{m-1}(\mathbf{w}) + \frac{1}{m}\nabla\mathcal{L}_m(\mathbf{w})\nabla\mathcal{L}_m^\top(\mathbf{w})\right)^{-1}.$$

Unrolling the recursion with $\widehat{\mathbf{F}}_0^{-1}(\mathbf{w}) = \frac{1}{\lambda}\mathbf{I}_d$, we can obtain an iterative formula to exactly calculate the inverse of the empirical Fisher matrix as

$$\widehat{\mathbf{F}}^{-1}(\mathbf{w}) = \widehat{\mathbf{F}}_m^{-1}(\mathbf{w}) = \frac{1}{\lambda}\mathbf{I}_d - \sum_{i=1}^{m} \frac{\left(\widehat{\mathbf{F}}_{i-1}^{-1}(\mathbf{w})\nabla\mathcal{L}_i(\mathbf{w})\right)\left(\widehat{\mathbf{F}}_{i-1}^{-1}(\mathbf{w})\nabla\mathcal{L}_i(\mathbf{w})\right)^\top}{m+\nabla\mathcal{L}_i^\top(\mathbf{w})\widehat{\mathbf{F}}_{i-1}^{-1}(\mathbf{w})\nabla\mathcal{L}_i(\mathbf{w})}.$$

The iterative formulation enjoys a number of computational advantages over the direct implementation. The most notable ones are 1) avoiding explicit calls to the expensive and dampening-sensitive matrix inversions, and 2) allowing successive updates of the inverse as new gradients are computed, never needing to store all m gradients of size d and thus significantly reducing memory requirements.

3.3 Memory and run-time complexity

Computing and storing the inverse empirical Fisher $\widehat{\mathbf{F}}^{-1}(\mathbf{w}) \in \mathbb{R}^{d\times d}$ is prohibitively expensive for modern LLMs, which have hundreds of millions of parameters, due to the quadratic complexity on the number of weights d. However, Singh and Alistarh (2020) have shown that a diagonal block-wise approximation of the empirical Fisher matrix can be very accurate for pruning of convolutional neural networks. We adapt the same approach here, in the context of LLMs. Thus, for blocks of width B along the main diagonal, memory requirements for the computation of the inverse Fisher matrix are reduced from the quadratic $\mathcal{O}(d^2)$ to a linear $\mathcal{O}(Bd)$ dependence on the number of weights d. At the same time, run-time complexity relaxes from $\mathcal{O}(md^2)$ to $\mathcal{O}(mBd)$. As we will show, this computation can be efficiently and accurately performed for moderate values of m and B.

Another alternative we investigated was the matrix-free approach of Frantar et al. (2021), which does not require a block-wise approximation and has complexity $\Theta(dm)$. However, our investigation showed that this approach required high values of m to be accurate (Appendix Figure 3), which leads to excessive memory cost in the case of BERT models.

3.4 Efficient and scalable implementation

On the practical side, we have identified general hyper-parameters $B = 50$ for the block size, and $m = 1024$ for the number of gradients which produce state-of-the-art results for all analyzed BERT models (for more details please see Appendix A.4), while still being able to fit on the 24GB RTX 3090 GPU. We reflect upon the computational costs in more detail in Appendix A.3. Moreover, for these parameter values, the block-wise approximation of $\widehat{\mathbf{F}}^{-1}(\mathbf{w})$ can be implemented very efficiently on modern accelerators. Specifically, we take advantage of the fact that such hardware favors batched matrix operations, and that the blocks of size $B \times B$ in $\widehat{\mathbf{F}}^{-1}(\mathbf{w})$ are independent. With $N_B = \frac{d}{B}$ we refer to the total number of blocks, i.e. the batch-dimension. The procedure works as follows. First, we compute batched matrix-vector products $\widehat{\mathbf{F}}_{i-1}^{-1}(\mathbf{w})\nabla\mathcal{L}_i(\mathbf{w}) \in \mathbb{R}^{N_B\times B}$ and scalar denominators $m + \nabla\mathcal{L}_i^\top(\mathbf{w})\widehat{\mathbf{F}}_{i-1}^{-1}(\mathbf{w})\nabla\mathcal{L}_i(\mathbf{w}) \in \mathbb{R}^{N_B}$. Then, we update the inverse Fisher for each block by computing the scalar-scaled outer products $\left(\widehat{\mathbf{F}}_{i-1}^{-1}(\mathbf{w})\nabla\mathcal{L}_i(\mathbf{w})\right)\left(\widehat{\mathbf{F}}_{i-1}^{-1}(\mathbf{w})\nabla\mathcal{L}_i(\mathbf{w})\right)^\top$ of shape $\mathbb{R}^{N_B\times B\times B}$.

4 Experimental Validation

To ease reproducibility, we conduct our experiments in modified versions of the popular open-source libraries: Transformers (Wolf et al.,

2020), and SparseML (Kurtz et al., 2020). All of our experiments are using publicly available datasets via Lhoest et al. (2021) and focus on the $\text{BERT}_{\text{BASE}}$ model (Devlin et al., 2019), one of the most commonly used LLMs, composed of 12 transformer layers with 110M parameters. Following community standards, we prune encoder's weights (85M) and report sparsities relative to this number. All of our models, compression recipes and the full implementation will be made public.

4.1 Downstream Unstructured Pruning

We first revisit the accuracy-compression trade-off for pruning on downstream tasks.

Goals and setup. We compare existing approaches, notably Movement Pruning (MvP) (Sanh et al., 2020) and Lottery Ticket (LT-BERT) (Chen et al., 2020), against the gradual unstructured oBERT method, introduced in Section 3. Our experiments evaluate performance on a variety of downstream (English) tasks commonly used to evaluate model compression: question answering SQuAD v1.1 (Rajpurkar et al., 2016), sentence classification Quora Duplicate Query Dataset QQP (Shankar et al., 2017), and natural language inference MNLI (Williams et al., 2018).

Comparison with MvP. For a fair comparison with MvP, we consider the 10-epoch gradual pruning setup used to obtain the best results by Sanh et al. (2020). Specifically, we start from the $\text{BERT}_{\text{BASE}}$ model and perform 2 epochs of fine-tuning, followed by 6 epochs of pruning, and 2 further epochs of fine-tuning of the compressed model. We impose a global sparsity distribution over all layers, prune with oBERT two times per epoch, and use KD from the fine-tuned $\text{BERT}_{\text{BASE}}$ teacher. For oBERT pruning we use $m = 1024$ gradients, block size $B = 50$, and dampening $\lambda = 10^{-7}$ to approximate the inverse Hessian matrix. In all of our runs, the first pruning step prunes 70% of weights and then follows the cubic interpolation (Zhu and Gupta, 2018) to the target sparsity. This large first pruning step gives more time to recover from the later pruning steps, which impose higher sparsities. All hyper-parameters are described in detail in Appendix A.5, and the results are given in Table 1 (in the *10 Epochs* section).

We observe that Optimal BERT Surgeon outperforms Movement Pruning by a significant margin, more than 2 points of F1 score at the same sparsity. Remarkably, the model pruned with oBERT to 97% sparsity has similar accuracy to MvP-pruned model at 90% sparsity, which has roughly 3x more weights. This reinforces the effectiveness of second-order information for pruning.

Table 1: Downstream tasks dev-set performance of pruned $\text{BERT}_{\text{BASE}}$ models. (* approximate results as the exact numbers are not available.)

Task	BERT BASE	Spars.	Soft MvP	oBERT (ours)	LT-BERT	oBERT (ours)
	Epochs		10 Epochs		30 Epochs	
SQuAD F1	88.54	80%	-	-	86.54	**89.04**
		90%	84.90	**87.98**	68.00*	**88.31**
		97%	82.30	**84.65**	-	**85.98**
MNLI m-acc	84.54	80%	-	-	82.60	**84.32**
		90%	81.20	**83.20**	75.00*	**83.79**
		97%	79.50	**81.00**	-	**81.77**
QQP Acc	91.06	80%	-	-	90.30	**91.57**
		90%	90.20	**90.89**	90.00	**91.35**
		97%	89.10	**90.23**	-	**90.87**

Extended pruning and fine-tuning. Next, we examine effects of extending the gradual schedule to 30 epochs, matching the setup used for LT-BERT (Chen et al., 2020). The only difference compared to our 10 epoch setup is that we now prune with oBERT every four epochs, and rewind learning rate after each pruning step. The extended setup leaves more time to recover from pruning, which reflects in the improved results in Table 1 (*30 Epochs* section). We report the mean over three runs. For additional evaluation metrics and standard deviations please see Tables 12 and 15 in the Appendix. The results show a clear accuracy difference between oBERT and LT-BERT, especially at high sparsities. This difference is justified since the LT based approach attempts to mainly transfer *network connectivity*, whereas the oBERT can also benefit from the weight values. Finally, we examined the impact of extended setup with Soft MvP on SQuAD, targeting 90% sparsity (not shown in the Table), leading to an (F1, EM) combination of $(87.42, 79.83)$ for MvP. The F1 gap in favor of oBERT is lower than at 10 epochs, suggesting that extended finetuning helps all methods; yet, it is far from negligible.

4.2 Upstream Unstructured Pruning

An appealing alternative to downstream pruning is to compress models upstream, on the semi-supervised pre-training task (Zafrir et al., 2021). Given the upstream pruned model, computational

requirements for obtaining downstream fine-tuned models are significantly reduced, as only fine-tuning of the remaining weights is necessary.

Goals and setup. To compare with existing approaches, notably Prune OFA (Zafrir et al., 2021) and LT-BERT (Chen et al., 2020), we gradually prune with oBERT directly at upstream datasets, BookCorpus and English Wikipedia, and then fine-tune the remaining unpruned weights on the subset of GLUE tasks.

Teacher preparation. Following Liu et al. (2019), we start with the HuggingFace $\text{BERT}_{\text{BASE}}$ uncased model, and fine-tune it for additional 10 epochs only on the masked language modeling task.

Pruning at upstream. Once the distillation teacher is trained, we gradually prune and fine-tune the $\text{BERT}_{\text{BASE}}$ model for 3 epochs, using KD from the dense teacher. We prune four times per epoch, and rewind learning rate to the initial value after each pruning step. Hyper-parameters for oBERT are the same as for downstream pruning in 4.1; a full description can be found in Appendix A.6.

Sparse-transfer to downstream. To evaluate the resulting upstream-pruned models, we finetune the unpruned weights on downstream tasks with KD from the fine-tuned $\text{BERT}_{\text{BASE}}$ model. For a fair comparison with Prune OFA, we fine-tune for 8 epochs. The results in Table 2 show that sparse models produced by oBERT outperform state-of-the-art methods by significant margins. We report the mean over four runs. For additional evaluation metrics and standard deviations please see Appendix Tables 13 and 16. It is worth emphasizing that in contrast to Prune OFA, which performed extensive hyper-parameter tuning for sparse-transfer, our recipe is simple and general across downstream tasks: 8 epochs of fine-tuning with linearly decaying learning rate. This suggests that sparse pre-trained models found by oBERT constitute a strong starting point for sparse transfer learning, which can be further improved by task-specific hyper-parameter tuning.

4.3 Compound Compression for CPUs

To probe the potential practical impact of our approach, we specialize the technique for deployment on CPUs, corresponding to "edge" deployments. Specifically, we tailor our sparse models to the DeepSparse (NeuralMagic, 2021) sparsity-aware runtime, by compounding unstructured pruning with additional compression techniques.

Table 2: Sparse-transfer dev-set performance of upstream-pruned $\text{BERT}_{\text{BASE}}$ models. (* approximate results as the exact numbers are not available.)

Task	BERT BASE	Sparsity	LT-BERT	Prune OFA	oBERT (ours)
SQuAD F1	88.54	90% 97%	68.00* -	87.25 -	**88.49** 84.92
MNLI m-acc	84.54	90% 97%	75.00* -	81.45 -	**83.40** 80.91
QQP Acc	91.06	90% 97%	90.00 -	90.93 -	**90.99** 90.33
SST-2 Acc	93.01	90%	85.00*	90.88	**92.20**
QNLI Acc	91.25	90%	80.00*	89.07	**89.97**

Direct layer dropping. The competitive results obtained at high sparsities in sections 4.1 and 4.2 suggest that $\text{BERT}_{\text{BASE}}$ may be overparameterized for downstream tasks. To improve compression ratio and inference speed, we apply "direct" layer dropping: we initially drop all but 3 or 6 of the BERT's 12 layers. We drop layers from our upstream teacher, and, following (Turc et al., 2019), fine-tune them with KD in the same setup used to prepare the upstream teacher. These 3 and 6 layer models are used as starting points for downstream pruning. More sophisticated layer dropping techniques (Fan et al., 2019), could bring further accuracy gains; we leave this for future work.

Block pruning and QAT. High-performance inference usually benefits more from (semi) structured sparsity patterns than from the unstructured ones. Hence, we employ the generalized oBERT formulation introduced in the section 3 and prune weights in the 4-block pattern, meaning that contiguous blocks of 4 weights are either set to zero or kept dense. Both pruning types, unstructured and 4-block, can be leveraged for computational speedups with the DeepSparse runtime, but 4-block pruning coupled with INT8 quantization can provide further performance gains. For quantization, we apply standard quantization-aware training (QAT) (Jacob et al., 2018) on top of the 4-block models (see Appendix A.7 for a full description).

Compounding for deployment. To determine the impact of different compression schemes, we investigate unstructured and 4-block pruning of the 3, 6, and 12-layer models. For all runs, we use the same set of hyper-parameters from the extended pruning

Table 3: F1 score of the 3, 6, and 12-layer models compound-compressed on the SQuADv1.1.

Layers	Sparsity	Unstructured	4-block	+QAT
12	0%	89.48	89.48	89.06
	80%	89.04	88.57	87.89
	90%	88.31	87.57	86.68
6	0%	88.32	88.32	87.94
	80%	88.20	87.00	86.10
	90%	86.78	85.34	84.59
3	0%	84.66	84.66	84.25
	80%	84.08	82.79	82.04
	90%	82.50	80.69	79.66

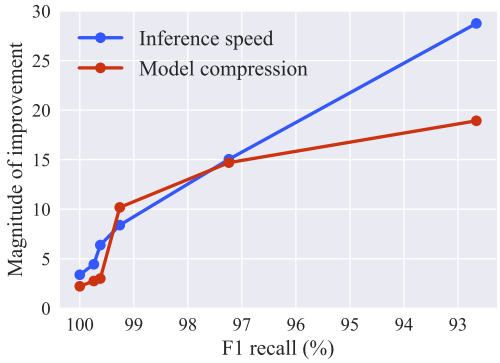

Figure 2: F1 recall on the SQuADv1.1 task relative to improvements in CPU-inference speed and model size.

and fine-tuning setup in Section 4.1. The results are given in Table 3, where we also report accuracy of the corresponding dense models (0% sparsity) in the same setup. For additional evaluation metrics, please see Table 14. The results indicate that compression methods can be combined without model collapse, although the accuracy drops do compound. The fact that the layer-dropped models are also highly compressible suggests that structured and fine-grained (unstructured) compression are complementary. We find it remarkable that our 6-layer unstructured oBERT-pruned model is competitive with the 12-layer MvP-pruned model when both are pruned to 90% sparsity.

Practical trade-offs. We now benchmark these models in end-to-end fashion, both in terms of model size and inference speed. For model size, we report size of the checkpoint in MB after standard gzip compression. For inference speed, we report number of items per second (throughput) on the well-established SQuAD v1.1 CPU-inference benchmark with a sequence length of 128 and a batch size of 32. Figure 2 depicts relative accuracy versus magnitude of improvement in speed and model size. As baseline for full recovery, we follow the community-standard e.g. (Sanh et al., 2020), and adopt the dense $\text{BERT}_{\text{BASE}}$ model with 88.54 F1 score. The baseline for inference speed is dense $\text{BERT}_{\text{BASE}}$ inference with DeepSparse, which matches the industry-standard ONNX Runtime inference engine. Results suggest a roughly-linear trade-off between compression and accuracy loss, with a compression jump around 1% accuracy drop, due to quantization being applied. Specifically, we observe 8.4x higher inference speedup at $< 1\%$ accuracy drop, 10x speedup at $< 2\%$ drop, 15x speedup at $< 3\%$ drop, and 29x speedup at $< 7.5\%$ accuracy drop. This shows how compound compression can optimize LLMs to various latencies. See Appendix Table 17 for full results.

4.4 Pruning for GPU speedups (N:M sparsity)

Even though our previous results targeted CPUs for deployment, we now show that our pruning approach can also be relevant to GPUs. We apply the semi-structured variant of oBERT to impose the 2-out-of-4 sparsity pattern, which is supported on NVIDIA Ampere GPUs (Mishra et al., 2021). More specifically, we prune in *one-shot*, and compare against the magnitude pruning baseline in Table 4. All other methods require full fine-tuning, and thus don't support the one-shot setup. oBERT significantly outperforms magnitude pruning, and with only 1-epoch of fine-tuning it is able to fully recover dense accuracy with (F1, EM) = (88.58, 81.16). With this sparsity pattern, the pruned model achieves 1.85x speedup on Ampere devices.

Table 4: One-shot 2:4 pruning of the fine-tuned $\text{BERT}_{\text{BASE}}$ model.

Task	$\text{BERT}_{\text{BASE}}$	Magnitude	oBERT (ours)
SQuAD F1 / EM	88.54 / 81.41	49.97 / 35.24	**83.17 / 74.18**

5 Discussion

Comparison with concurrent work. Concurrent work introduced PLATON (Zhang et al., 2022), which addresses unstructured pruning of BERT models via estimates of confidence bounds. It does not make use of KD, so for a fair comparison we rerun our experiments without KD as well. Contrary to PLATON, which reports best results after

an extensive hyper-parameter search for each task independently, we apply our sparse-transfer setup with the upstream pruned model and only sweep for the number of epochs $\in [1, 8]$. We employ early stopping to prevent overfitting on smaller GLUE tasks. As can be seen from Table 5, oBERT outperforms PLATON across all tasks.

Table 5: Compressed $\text{BERT}_{\text{BASE}}$ models to 90% sparsity on GLUE tasks without knowledge distillation.

Task	$\text{BERT}_{\text{BASE}}$	PLATON	oBERT (ours)
MNLI m / mm	84.6 / 83.4	82.0 / 82.2	**82.2 / 82.5**
QQP Acc / F1	91.5 / 88.5	90.2 / 86.8	**90.4 / 87.1**
QNLI Acc	91.3	88.9	**89.3**
MRPC Acc / F1	86.4 / 90.3	84.3 / 88.8	**85.6 / 89.3**
SST-2 Acc	92.7	90.5	**92.0**
CoLA Mcc	58.3	44.3	**48.47**
STS-B Pear / Spear	90.2 / 89.7	87.4 / 87.1	**88.0 / 87.6**

Broader comparison. We now contrast our compound-compressed $\text{BERT}_{\text{BASE}}$ models relative to alternative compression techniques. We compare against DistilBERT (Sanh et al., 2019), TinyBERT (Jiao et al., 2020), and Block Pruning For Faster Transformers (Hybrid Filled MvP) (Lagunas et al., 2021). DistilBERT leverages KD during pre-training and fine-tuning to obtain a 6-layer model fine-tuned for a specific downstream task. TinyBERT makes use of a specialized Transformer-KD scheme to distill knowledge and intermediate representations at both stages, pre-training and fine-tuning on a specific task. In contrast, we use a simpler approach and employ KD from teacher's outputs only. Hybrid Filled MvP (Lagunas et al., 2021) employs semi-structured pruning and weight reintroduction. The comparison is given in Table 6, where we report the number of unpruned encoder weights as size, compression ratio and inference speedup relative to the dense $\text{BERT}_{\text{BASE}}$ in the same inference environment, and F1 score on the dev-set of the SQuAD v1.1 dataset. The results suggest that our compressed models improve upon the current state-of-the-art techniques, setting new very competitive baselines with respect to all metrics: accuracy, model size, and inference speed.

Table 6: Compressed $\text{BERT}_{\text{BASE}}$ models on the SQuADv1.1 task. ($\text{oBERT}_{6,80}$ stands for the 6-layer model pruned to 80% sparsity.)

Model	Size	Compr.	Speedup	F1	Dev.
$\text{BERT}_{\text{BASE}}$	85.0M	1.00x	1.00x	88.54	
< 6-layers					
TinyBERT_4	4.5M	18.88x	9.40x	82.10	GPU
$\text{oBERT}_{3,90}$	**2.1M**	**40.00x**	**14.80x**	**82.50**	CPU
6-layers					
DistilBERT	42.5M	2.00x	2.00x	86.90	GPU
TinyBERT_6	42.5M	2.00x	2.00x	87.50	GPU
$\text{oBERT}_{6,80}$	**8.5M**	**10.00x**	**6.38x**	**88.20**	CPU
12-layers					
Hybrid F. MvP	30.7M	2.76x	1.84x	88.70	GPU
$\text{oBERT}_{12,80}$	**17.0M**	**5.00x**	**3.38x**	**89.04**	CPU

$\text{BERT}_{\text{LARGE}}$ results. Most of our results presented in Section 4 targeted the widely-adopted $\text{BERT}_{\text{BASE}}$ model. This gave us an opportunity for a fair comparison against many different methods. To verify that our approach does not pertain only to the $\text{BERT}_{\text{BASE}}$ model, in Table 7 we present downstream pruning results on the three times larger $\text{BERT}_{\text{LARGE}}$ model and the SQuADv1.1 task. As can be seen from the Table, even the model pruned with oBERT at double the sparsity (95%) outperforms Prune OFA (90%).

Table 7: Compressed $\text{BERT}_{\text{LARGE}}$ models on the SQuADv1.1 task.

$\text{BERT}_{\text{LARGE}}$ F1 / EM	Sparsity	Prune OFA	oBERT (ours)
91.22 / 84.45	90%	90.20 / 83.35	**91.07 / 84.61**
91.22 / 84.45	95%	NA	90.29 / 83.58

MLPerf Inference Benchmark. Motivated by our state-of-the-art results across-the-board, we apply our full compound compression pipeline to compress $\text{BERT}_{\text{LARGE}}$ and MobileBERT (Sun et al., 2020b) models in the context of the industrial MLPerf Inference Benchmark[1]. In brief, we were able to achieve order-of-magnitude improvements in terms of model size and inference speedups, while maintaining >99% of the dense $\text{BERT}_{\text{LARGE}}$ accuracy. For details please see Appendix A.1, as well as our open-source submission.

[1] https://mlcommons.org/en/

6 Broader Impact

Our work is part of the general trend of producing inference efficient models which approximate performance of their larger bases. By and large, this work should help increase model efficiency, thereby reducing computational and ultimately monetary cost of executing such models. Moreover, it could allow models to be used by those who do not have access to expensive specialized computing clusters: for instance, our main speedup results are aimed at widely-available CPUs.

7 Limitations

As any academic study, our work is not without its limitations. We split their discussion into limitations that are *inherent to our method*, and limitations *of our present study*; the latter can be overcome by extensions of our work. In the first category, we begin by highlighting the fact that our second-order method relies on approximations, which are inherent in order to scale such methods to BERT scale. Prior studies, e.g. (Singh and Alistarh, 2020) have performed careful examinations of the validity of these approximations in the context of CNN models. The strength of our empirical results can be seen as indirect evidence that these approximations apply to BERT models as well. A second, technical, limitation is the fact that our method requires non-trivial additional storage cost; while we have shown that our experiments can be executed on a single commodity GPU (NVIDIA RTX 3090), this limits the range of devices on which the technique may be applied. However, we provide an efficient and easy way to scale our approach with more GPUs, which is automatically utilized in a multi-GPU environment.

Another limitation which we aim to remove in future work is the focus on relatively fine-grained sparsity types, such as unstructured and semi-structured pruning.

References

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. 2020. The lottery ticket hypothesis for pre-trained bert networks. *Advances in neural information processing systems*, 33:15834–15846.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Angela Fan, Edouard Grave, and Armand Joulin. 2019. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*.

Wikimedia Foundation. Wikimedia downloads.

Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*.

Elias Frantar, Eldar Kurtic, and Dan Alistarh. 2021. M-fac: Efficient matrix-free approximations of second-order information. *Advances in Neural Information Processing Systems*, 34.

Trevor Gale, Erich Elsen, and Sara Hooker. 2019. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*.

Song Han, Huizi Mao, and William J Dally. 2015. A deep neural network compression pipeline: Pruning, quantization, huffman encoding. *arXiv preprint arXiv:1510.00149*, 10.

Babak Hassibi and David Stork. 1992. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5.

Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7).

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. Tinybert: Distilling bert for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174.

Sehoon Kim, Sheng Shen, David Thorsley, Amir Gholami, Woosuk Kwon, Joseph Hassoun, and Kurt Keutzer. 2022. Learned token pruning for transformers. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, page 784–794. Association for Computing Machinery.

Olga Kovaleva, Saurabh Kulshreshtha, Anna Rogers, and Anna Rumshisky. 2021. BERT busters: Outlier layernorm dimensions that disrupt BERT. *CoRR*, abs/2105.06990.

Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William Leiserson, Sage Moore, Bill Nell, Nir Shavit, and Dan Alistarh. 2020. Inducing and exploiting activation sparsity for fast inference on deep neural networks. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5533–5543, Virtual. PMLR.

François Lagunas, Ella Charlaix, Victor Sanh, and Alexander Rush. 2021. Block pruning for faster transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10619–10629, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. *Advances in neural information processing systems*, 2.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. 2021. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184. Association for Computational Linguistics.

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

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. 2021. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*.

NeuralMagic. 2021. Deep sparse: A fast cpu inference engine.

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

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *EMNLP*.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2020. Poor man's BERT: smaller and faster transformer models. *CoRR*, abs/2004.03844.

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

Victor Sanh, Thomas Wolf, and Alexander Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in Neural Information Processing Systems*, 33:20378–20389.

Iyer Shankar, Dandekar Nikhil, and Csernai Kornel. 2017. First quora dataset release: Question pairs.

S. Shankar. 2017. Identifying quora question pairs having the same intent.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821.

Sidak Pal Singh and Dan Alistarh. 2020. Woodfisher: Efficient second-order approximation for neural network compression. *Advances in Neural Information Processing Systems*, 33.

Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. 2022. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*.

Sharath Nittur Sridhar and Anthony Sarah. 2020. Undivided attention: Are intermediate layers necessary for bert? *arXiv preprint arXiv:2012.11881*.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020a. Mobilebert: a compact task-agnostic bert for resource-limited devices. In *ACL*.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020b. Mobilebert: a compact task-agnostic bert for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2158–2170.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: The impact of student initialization on knowledge distillation. *ArXiv*, abs/1908.08962.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Elena Voita, David Talbot, F. Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *ACL*.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*, 33:5776–5788.

Liu Weijie, Zhou Peng, Zhao Zhe, Wang Zhiruo, Deng Haotang, and Ju Qi. 2020. Fastbert: a self-distilling bert with adaptive inference time. In *Proceedings of ACL 2020*.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

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. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy J. Lin. 2020. Deebert: Dynamic early exiting for accelerating bert inference. In *ACL*.

Dongkuan Xu, Ian En-Hsu Yen, Jinxi Zhao, and Zhibin Xiao. 2021. Rethinking network pruning – under the pre-train and fine-tune paradigm. In *NAACL*.

Shixing Yu, Zhewei Yao, Amir Gholami, Zhen Dong, Sehoon Kim, Michael W Mahoney, and Kurt Keutzer. 2022. Hessian-aware pruning and optimal neural implant. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3880–3891.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, pages 36–39.

Ofir Zafrir, Ariel Larey, Guy Boudoukh, Haihao Shen, and Moshe Wasserblat. 2021. Prune once for all: Sparse pre-trained language models. *arXiv preprint arXiv:2111.05754*.

Qingru Zhang, Simiao Zuo, Chen Liang, Alexander Bukharin, Pengcheng He, Weizhu Chen, and Tuo Zhao. 2022. Platon: Pruning large transformer models with upper confidence bound of weight importance. In *International Conference on Machine Learning*, pages 26809–26823. PMLR.

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

M. Zhu and Suyog Gupta. 2018. To prune, or not to prune: exploring the efficacy of pruning for model compression. *ArXiv*, abs/1710.01878.

Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 19–27.

A Appendix

A.1 MLPerf Inference benchmark

Following the MLPerf benchmark guidelines on producing compressed and fast models while maintaining >99% of the $\text{BERT}_{\text{LARGE}}$ F1 score on the SQuADv1.1 task, we explore two directions. In the first one, dubbed oBERT-Large, we compound compress the $\text{BERT}_{\text{LARGE}}$ model without any changes to its architecture. Therefore, we apply

4-block downstream pruning to 95% sparsity followed by the quantization aware training (QAT). In the second direction we focus on recovering the $\text{BERT}_{\text{LARGE}}$ accuracy by compressing an already compact MobileBERT model, dubbed oBERT-MobileBERT. More specifically, we apply direct layer dropping, leaving only 14 transformer layers out of the original 24, followed by the 4-block pruning to 50% sparsity and quantization aware training. We present results in Table 8, where models were evaluated with the DeepSparse inference engine, using a server with two Intel(R) Xeon(R) Platinum 8380 (IceLake) CPUs with 40 cores each, batch-size 128 and sequence length 384. For more details please see our official submission at https://github.com/neuralmagic/mlperf_inference_results_v2.1/tree/master/open/NeuralMagic.

A.2 Additional comparisons

Here we reflect upon some other methods focused on efficient inference for LLMs, which are orthogonal to weight pruning. For example, Learned Token Pruning (Kim et al., 2022) tries to adaptively remove unimportant tokens in input sequences and provides 2x higher throughput at < 1% accuracy drop; at the same accuracy drop, our compressed model is able to achieve 8.4x higher throughput. DeeBERT (Xin et al., 2020) and FastBERT (Weijie et al., 2020) apply an *early-exit* technique for inference speedup. The latter achieves 2-3x faster inference without performance degradation. However, the method only applies to batch size one. Nevertheless, in terms of direct comparison, our compressed models are able to achieve 4x faster inference on CPUs without accuracy degradation. Overall, we emphasize the fact that these methods are complementary to our compression techniques, so it would be interesting to investigate computational gains by combining such methods.

A.3 Computational costs

In practice, for the 12-layer $\text{BERT}_{\text{BASE}}$ model with $d = 85M$ encoder weights and block size $B = 50$, the $\mathcal{O}(Bd)$ memory requirement translates to approximately 17GB, which can be easily kept on the 24GB RTX 3090 card. While this amount of memory is available on high-performance GPUs, it is also straightforward to split the $N_B \times B \times B$ tensor along the batch-dimension N_B and utilize additional GPUs or even memory swapping with CPU. Our implementation updates the inverse Hessian approximation in negligible time, and can run asynchronously while the next gradient is being fetched. Computing saliency scores and optimal weight updates takes only a few seconds.

A.4 Optimal BERT Surgeon (oBERT) hyper-parameters

Hyper-parameters. The oBERT pruning method has three tunable hyper-parameters: number of gradients (m), block size (B), and dampening (λ). These are supposed to be tuned with respect to the model and available computational resources. In all of our runs, across all models and datasets, we use the same set of hyper-parameters which we found to work best for the $\text{BERT}_{\text{BASE}}$ model on the SQuAD v1.1 dataset. We conjecture that further tuning for smaller models (3 and 6-layer models) could improve their results, but for simplicity and fairness to other methods, we apply the same ones found for the $\text{BERT}_{\text{BASE}}$.

Ablation studies. The procedure to find the optimal set of hyper-parameters for a model consists of a grid search over the possible hyper-parameter combinations and one-shot pruning runs to various high sparsity targets to evaluate the quality of the pruning approximation for each combination. We found that $m = 1024$, $B = 50$, and $\lambda = 10^{-7}$ produce state-of-the-art results for a negligible computational overhead with the $\text{BERT}_{\text{BASE}}$ model. Frantar et al. (2021) shows that larger block sizes require more gradients for better approximation. Given the massive size of the $\text{BERT}_{\text{BASE}}$ model, we picked this setup as it was the best performing one that could still fit on a single 24GB RTX 3090 GPU card. In Figures 3, 4, and 5 we visualize a fraction of the one-shot pruning ablations with respect to all three hyper-parameters that motivated us to pick these specific values.

A.5 Downstream pruning

Teacher preparation. For all downstream pruning runs we make use of the KD from the fine-tuned $\text{BERT}_{\text{BASE}}$ teacher outputs. The teacher is fine-tuned on the corresponding downstream task following the default hyper-parameters for SQuAD[2]

[2]https://github.com/huggingface/transformers/tree/main/examples/pytorch/question-answering

Table 8: MLPerf inference results for oBERT compressed $\text{BERT}_{\text{LARGE}}$ and MobileBERT models.

Model	Precision	F1 Score (R=X% recovery)	File Size	Compression Ratio	Throughput (samples/sec)	Speedup
BERT-Large dense baseline	FP32	90.87 (R=100%)	1.30 GB	1x	15.49	1x
oBERT-Large	INT8	90.21 (R=99.27%)	38.20 MB	34x	230.74	15x
oBERT-MobileBERT	INT8	90.32 (R=99.39%)	9.56 MB	136x	928.58	60x

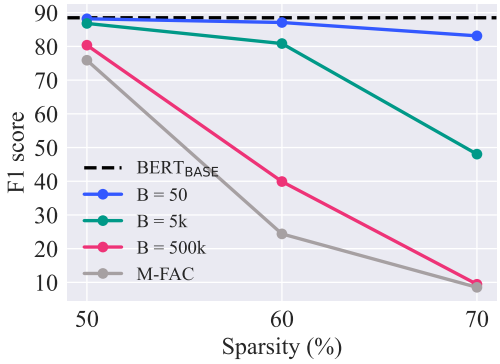

Figure 3: One-shot pruning ablation study with respect to the block size (B), with $m = 1024$ and $\lambda = 10^{-7}$, on the $\text{BERT}_{\text{BASE}}$ model and the question-answering SQuAD v1.1 dataset. M-FAC stands for the full inverse Hessian approximation (Frantar et al., 2021).

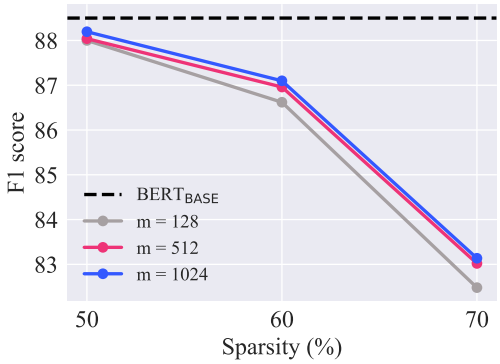

Figure 4: One-shot pruning ablation study with respect to the number of gradients (m), with $B = 50$ and $\lambda = 10^{-7}$, on the $\text{BERT}_{\text{BASE}}$ model and the question-answering SQuAD v1.1 dataset.

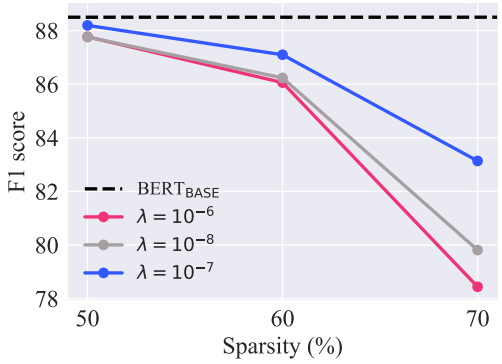

Figure 5: One-shot pruning ablation study with respect to the dampening (λ), with $m = 1024$ and $B = 50$, on the $\text{BERT}_{\text{BASE}}$ model and the question-answering SQuAD v1.1 dataset.

and GLUE (QQP and MNLI)[3].

Pruning setup. In Table 9 we describe in detail all hyper-parameters for downstream pruning results presented in Tables 1 and 3. For easier comprehension, we also visualize learning rate schedules in Figures 6 and 8, and sparsity schedules in Figures 7 and 9.

3-, 6-layer models. We prepare our 3 and 6 layer models for downstream runs in two stages: layer dropping and retraining phase. We drop layers from our upstream teacher model (more details on it in Appendix A.6). After dropping, we retrain the remaining layers, following insights from (Turc et al., 2019), in the same setup used to prepare the upstream teacher with addition of the KD from it.

A.6 Upstream pruning

Teacher preparation. We prepare a teacher for upstream pruning by following some insights from (Liu et al., 2019). More concretely we start with the *bert-base-uncased*[4] model, adopt pre-training on

[3]https://github.com/huggingface/transformers/tree/main/examples/pytorch/text-classification

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

	10 Epochs	30 Epochs
Batch size	16 for SQuAD, 32 for GLUE	
Learning rate (initial, final)	(8e-5, 3e-5) for SQuAD, (8e-5, 2e-5) for GLUE	(8e-5, 8e-6) for SQuAD, (5e-5, 5e-6) for GLUE
Learning rate schedule	linear decay with rewinds	
Learning rate rewinds	one at epoch=8	periodic every 4 epochs, start at epoch=2
Knowledge Distillation (hardness, temp.)	(1.0, 2.0)	
Student model	12-layer: bert-base-uncased 6-layer: layer drop + pre-train with KD 3-layer: layer drop + pre-train with KD	
Teacher model	$\text{BERT}_{\text{BASE}}$	
Prune start	epoch=2	
Prune end	epoch=8	epoch=26
Pruning frequency	2x per epoch	once every 4 epochs
Initial sparsity step	12-layer: 70% 6-layer: 30% 3-layer: 30%	
Sparsity distribution	global over all layers	
oBERT parameters	Number of gradients $m = 1024$ Block size $B = 50$ Dampening $\lambda = 10^{-7}$	

Table 9: Downstream pruning hyper-parameters used to obtain results presented in Tables 1 and 3.

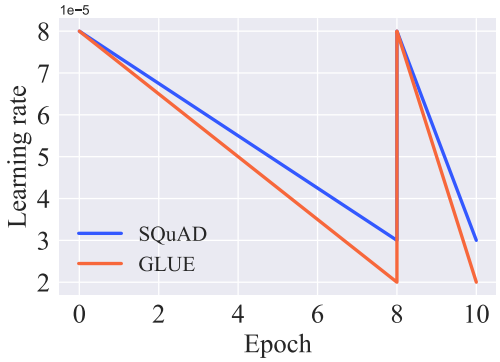

Figure 6: Visualized learning rate schedule for 10-epoch downstream runs.

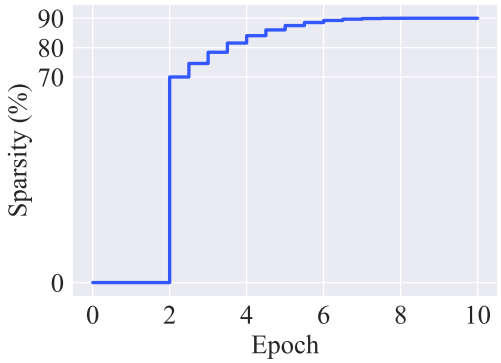

Figure 7: Visualized sparsity schedule for 10-epoch downstream runs with initial sparsity of 70% and target sparsity of 90%, following the cubic interpolation (Zhu and Gupta, 2018).

two datasets (BookCorpus[5] & English Wikipedia[6]) with focus on the masked language modeling task (MLM) for 10-epochs with batch size 256 and learning rate linearly decaying to zero from the initial value of 1e-4.

Pruning setup. In Table 10 we describe in detail our upstream pruning recipe. As can be noticed, our upstream pruning recipe is just a downscaled version of our 30-epoch downstream-pruning recipe to 3-epochs.

[5]https://huggingface.co/datasets/bookcorpus
[6]https://huggingface.co/datasets/wikipedia

A.7 Downstream quantization

We perform QAT on top of dense and 4-block pruned models on SQuAD v1.1 as shown in Table 3. We quantize to 8 bits the embedding matrices,

	3 Epochs
Datasets	BookCorpus & English Wikipedia
Batch size	256
Initial learning rate Learning rate schedule Learning rate rewinds	5e-4 linear decay with rewinds periodic every 0.5 epochs
Max sequence length Weight decay	512 0.01
Knowledge Distillation (hardness, temperature)	(1.0, 5.5)
Student model Teacher model	prepared upstream teacher prepared upstream teacher
Pruning frequency	4x per epoch

Table 10: Upstream pruning hyper-parameters.

	8 Epochs
Initial learning rate Learning rate schedule	1.5e-4 linear decay to 1.5e-6
Batch size	16 for SQuAD, 32 for GLUE
Knowledge Distillation (hardness, temperature)	(1.0, 5.5)
Teacher model	$\text{BERT}_{\text{BASE}}$

Table 11: Sparse-transfer learning hyper-parameters used to fine-tune upstream-pruned models at downstream tasks. These hyper-parameters are used to obtain results presented in Table 2.

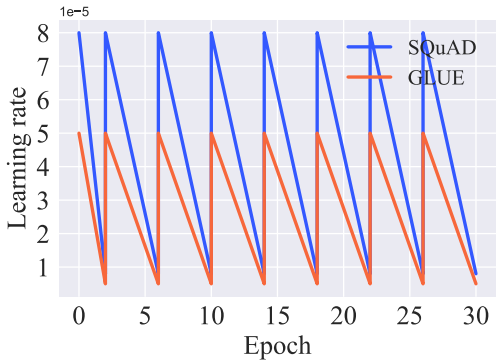

Figure 8: Visualized learning rate schedule for 30-epoch downstream runs.

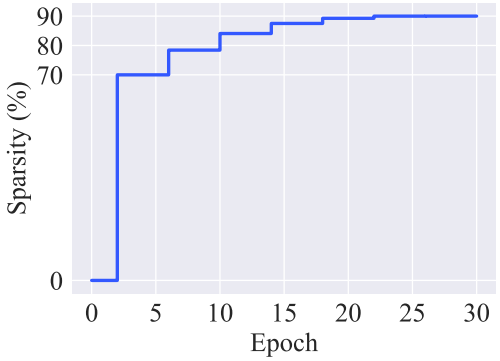

Figure 9: Visualized sparsity schedule for 30-epoch downstream runs with initial sparsity of 70% and target sparsity of 90%, following the cubic interpolation (Zhu and Gupta, 2018).

linear modules of all encoder units which includes matrices in their attention and feed forward layers, and the linear module of the output layer. Weights that were pruned are kept constant (zero) during quantization (sparsity mask preserved). Non-linear operations within the Softmax, LayerNorm and GeLU are not quantized. For each dense and 4-block pruned model in Table 3, we perform a total of ten epochs training where the quantization observers are active for the first five and the remaining is fine-tuning. We do hyper-parameter search over the learning rates of 1e-4, 8e-5, 5e-5, 3e-5 and the distillation hardness of 0.9 and 1.0. We then pick the model with the best F1 score.

A.8 Additional performance metrics

Due to the space constraints, in the paper we report F1 score for SQuAD v1.1, matched accuracy for MNLI, and accuracy for QQP dataset. As all of our hyper-parameters for MNLI and QQP are exactly

the same, we refer to these two datasets as GLUE. In Table 12 we report the additional metrics too: exact match (EM) for SQuAD v1.1, mismatched accuracy for MNLI, and F1 score for QQP dataset. Tables 15 and 16 present standard deviations of the corresponding results in Tables 1, 2 and 12. Finally, Table 14 presents the exact-match metric for the corresponding results in Table 3.

Task	BERT BASE	Sparsity	Soft MvP	oBERT (ours)	oBERT (ours)
	Epochs		10 Epochs		30 Epochs
SQuAD EM	81.22	80%	-	-	**82.08**
		90%	76.60	**80.76**	**81.12**
		97%	72.70	**76.14**	**78.11**
MNLI mm-acc	85.06	80%	-	-	**84.91**
		90%	81.80	**83.58**	**84.35**
		97%	80.10	**80.67**	**82.01**
QQP F1	88.00	80%	-	-	**88.63**
		90%	86.80	**87.69**	**88.30**
		97%	85.50	**87.05**	**87.66**

Table 12: Additional evaluation metrics for results presented in Table 1.

Task	BERT BASE	Sparsity	Prune OFA	oBERT (ours)
SQuAD EM	81.42	90%	79.83	**81.43**
		97%	-	76.90
MNLI mm-acc	85.06	90%	82.43	**83.78**
		97%	-	81.13
QQP F1	88.00	90%	87.72	**87.81**
		97%	-	86.97

Table 13: Additional evaluation metrics for results presented in Table 2.

Layers	Sparsity	Unstructured	4-block	+QAT
12	0%	82.71	82.71	81.99
	80%	82.08	81.46	80.57
	90%	81.12	80.14	78.84
6	0%	81.17	81.17	80.85
	80%	81.15	79.55	78.27
	90%	79.16	77.65	76.56
3	0%	76.62	76.62	76.06
	80%	75.62	74.07	72.70
	90%	73.61	71.36	70.00

Table 14: Additional evaluation metric (exact-match) for results presented in Table 3.

Task	Sparsity	oBERT (ours)
	Epochs	30 Epochs
SQuAD F1, EM	80%	0.11, 0.03
	90%	0.13, 0.13
	97%	0.11, 0.17
MNLI m, mm	80%	0.14, 0.13
	90%	0.05, 0.04
	97%	0.35, 0.22
QQP acc, F1	80%	0.08, 0.08
	90%	0.04, 0.06
	97%	0.05, 0.08

Table 15: Standard deviations for results presented in Tables 1 and 12.

Task	Sparsity	oBERT (ours)
SQuAD F1, EM	90%	0.13, 0.13
	97%	0.03, 0.14
MNLI m, mm	90%	0.08, 0.24
	97%	0.17, 0.35
QQP acc, F1	90%	0.06, 0.07
	97%	0.09, 0.18

Table 16: Standard deviations for results presented in Table 2 and 13.

A.9 Inference speedups and compression ratios of compressed models

Details on the results shown in Figure 2 are drawn from Table 17. As shown in the results, not all compound compressed models yield improvements in inference or compression relative to retained model performance but those that do allow for massive improvements.

A.10 Responsible NLP Research - Reproducibility Checklist

In addition to many items from the "Reproducibility Checklist" which are already carefully addressed throughout the paper and Appendix sections, here we provide the remaining details to facilitate reproducibility of our results.

A.10.1 Scientific Artifacts

Datasets. Our experiments use existing and well established benchmarks for pre-training and fine-tuning of LLMs. Each dataset was used without any additional forms of modifications. Given that we did not modify any of the datasets, we did not inspect for personal, sensitive, or offensive con-

Layers	Sparsity (%)	Compression Method	F1 score	F1 recall (%)	Throughput (items per sec.)	Speedup DeepSparse	Model size (gzip MB)	Compression Ratio (w.r.t. gzip)
12	0	none	88.54	100.00	65.81	1.00	384.7	1.00
12	80	unstructured	89.04	100.56	222.66	3.38	173.1	2.22
12	90	unstructured	88.31	99.74	292.40	4.44	140.1	2.75
12	80	4-block+QAT	87.89	99.26	552.22	8.39	37.8	10.18
6	80	unstructured	88.20	99.62	419.68	6.38	128.3	3.00
6	90	unstructured	86.78	98.01	663.02	10.07	111.8	3.44
6	80	4-block+QAT	86.10	97.24	989.54	15.04	26.2	14.70
3	80	unstructured	84.08	94.96	737.62	11.21	105.9	3.63
3	90	unstructured	82.50	93.18	974.00	14.80	97.7	3.94
3	80	4-block+QAT	82.04	92.66	1892.27	28.75	20.3	18.92

Table 17: Compression effects on model size and inference speed, evaluated at batch size 32 with sequence length 128 on SQuAD v1.1 dataset. Evaluated at the c5.12xlarge AWS instance.

tent, nor did we perform any kind of anonymization. For pre-training, we make use of the Toronto Book Corpus (TBC) (Zhu et al., 2015) [7] and the wikipedia.20200501.en (Foundation) [8]. For fine-tuning we make use of SQuAD v1.1 (Rajpurkar et al., 2016) [9], Quora Duplicate Question Dataset (QQP) (Shankar, 2017) [10], and Multi-Genre Natural Language Inference (MNLI) (Williams et al., 2018) [11] datasets. All these datasets are publicly available via HuggingFace datasets repository (Lhoest et al., 2021). The terms of usage and further details on each dataset can be found in their respective repositories.

Models. The model used as a starting point for all of our experiments is $\text{BERT}_{\text{BASE}}$, publicly available via HuggingFace Hub [12]. All other models presented in this paper will be released in openly-available repositories along with their compression recipes, training metrics and hyper-parameters.

A.10.2 Dataset Statistics

Dataset statistics are detailed in Table 18.

A.10.3 Computational Experiments

Upstream. All upstream runs are in general computationally expensive due to the large batch sizes and huge datasets. In our experiments we make use of 4x A100 40GB NVIDIA GPUs. In this configuration, a single training epoch takes approximately 6 hours. Since the cost of such a large compute instance is high, these experiments were only run with a single seed and without major hyper-parameter exploration.

Dataset	Train	Eval
SQuAD (examples)	87599	10570
MNLI (examples)	392702	19628
QQP (examples)	363,846	40,430
Wikipedia (words)	6078422	-
TBC (words)	74004228	-

Table 18: Statistics for training and evaluation datasets

Downstream. Our downstream experiments make use of various different GPU cards that were at out disposal: 16GB V100, 11GB RTX 2080 Ti, and 24GB RTX 3090. Each training epoch takes approximately 30 minutes, and as a result the 30 epoch runs take approximately 15 hours. For these experiments, we report mean results of three runs with different random seeds.

DeepSparse inference. We pair our compressed models with DeepSparse (NeuralMagic, 2021) a publicly-available sparsity-aware CPU inference engine. This CPU runtime can leverage both structured and unstructured sparsity, and quantization to deliver high performance on commodity CPUs. We ran DeepSparse on a 24-core Intel AWS c5.12xlarge server with 24 cores, 96 vCPUs, 192 GB of RAM and an AVX-512 compatible instruction set. All models are exported using the standard ONNX[13] format.

[7] https://huggingface.co/datasets/bookcorpus
[8] https://huggingface.co/datasets/wikipedia
[9] https://huggingface.co/datasets/squad
[10] https://huggingface.co/datasets/glue
[11] https://huggingface.co/datasets/glue
[12] https://huggingface.co/bert-base-uncased
[13] https://onnx.ai/

A.10.4 Computational Packages

Our experiments build on publicly available libraries to ensure ease of reproduction and extensibility. All of our implementations, training and evaluation code are built on top of HuggingFace's Transformers [14] and Datasets [15] libraries, NeuralMagic's SparseML [16] library for model compression, and their DeepSparse [17] engine for efficient inference on commodity CPUs.

[14] https://github.com/huggingface/transformers
[15] https://github.com/huggingface/datasets
[16] https://github.com/neuralmagic/sparseml
[17] https://github.com/neuralmagic/deepsparse