# A Fast Post-Training Pruning Framework for Transformers

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Pruning is an effective way to reduce the huge inference cost of large Transformer models. However, prior work on model pruning requires retraining the model. This can add high cost and complexity to model deployment, making it difficult to use in many practical situations. To address this, we propose a fast post-training pruning framework for Transformers that does not require any retraining. Given a resource constraint and a sample dataset, our framework automatically prunes the Transformer model using structured sparsity methods. To retain high accuracy without retraining, we introduce three novel techniques: (i) a lightweight mask search algorithm that finds which heads and filters to prune based on the Fisher information; (ii) mask rearrangement that complements the search algorithm; and (iii) mask tuning that reconstructs the output activations for each layer. We apply our method to $\text{BERT}_{\text{BASE}}$ and DistilBERT, and we evaluate its effectiveness on GLUE and SQuAD benchmarks. Our framework achieves up to 2.0× reduction in FLOPs and 1.56× speedup in inference latency, while maintaining $< 1\%$ loss in accuracy. Importantly, our framework prunes Transformers in less than 3 minutes on a single GPU, which is over two orders of magnitude faster than existing pruning approaches that retrain. Our code will be publicly available at GitHub.

## 1 Introduction

In recent years, Transformer [74] has become a *de facto* standard model architecture in Natural Language Processing [11, 44, 4], and it is becoming common in Computer Vision [13, 73, 46] and Speech Recognition [2, 24, 7] as well. However, efficient deployment of Transformer architectures has been challenging due to their large model size and high latency. To address this, structured pruning has become a promising technique for efficient Transformer deployment.

While prior work on pruning Transformers substantially reduces inference time, it is often difficult to use in practice for several reasons. First, previous approaches require retraining the pruned model and/or jointly learning the pruning configurations during training. For instance, Block Movement Pruning [36] increases the training time by 10× for pruning. Such additional training adds significant computational overhead, given the large training cost of Transformers. Second, previous methods add many moving parts to the model deployment process. Pruning pipelines are often complex and involve additional hyperparameter tuning. For instance, ROSITA [45] uses a three-stage knowledge distillation [22] with sophisticated pruning schedules. Such techniques demand significant engineering efforts for reproducing and debugging, which impedes their adoption in production pipelines. Third, these previous methods do not directly adapt to the users' constraints. They either rely on vague regularization hyperparameters or fixed architectures selected independently of the user settings. This can result in sub-optimal models not tailored for the given constraint and hardware.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

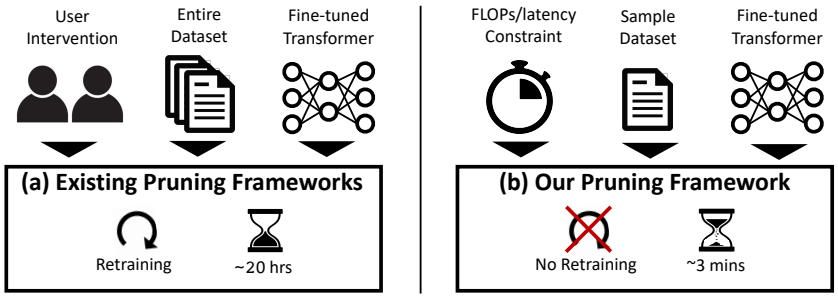


Figure 1: (a) Prior pruning frameworks require additional training on the entire training set and involve user intervention for hyperparameter tuning. This complicates the pruning process and requires a large amount of time (e.g., $\sim$20 hours). (b) Our pruning framework does not require retraining. It outputs pruned Transformer models satisfying the FLOPs/latency constraints within much less time (e.g., $\sim$3 minutes), without user intervention.

To address these limitations, we propose a fast *post-training pruning* framework for Transformers that does not require any retraining of the models. As illustrated in Figure 1, our framework takes as input a Transformer model, a sample dataset, and a FLOPs/latency constraint. It then outputs a pruned Transformer model that can be deployed immediately. By avoiding expensive retraining, the end-to-end compression pipeline can be extremely fast and simplified, typically in a few minutes, without any user interventions that complicate the whole process.

Indeed, post-training compression has been widely studied for quantization, and it gained considerable attention in both academia and industry [3, 92, 26]. Although quantization-aware training methods achieve higher compression rates in general, post-training quantization (PTQ) has often been more preferred in practice due to its retraining-free advantage. PTQ allows quantization to happen seamlessly when deploying models through the various frameworks such as TensorRT [53], TFLite [17], and OpenVINO [28]. Similar to these PTQ frameworks, our framework provides an out-of-the-box tool that enables pruning of Transformers without engineering efforts.

Our contributions can be summarized as follows:

- We propose a novel post-training pruning framework for Transformers that does not require model retraining. To retain accuracy without retraining, our framework consists of three stages of: (i) the *mask search* process guided by the Fisher information matrix to find which heads/filters to prune (Section 4.1); (ii) the *mask rearrangement* process that rearranges the pruned heads/filters by capturing intra-layer interactions (Section 4.2); and (iii) the *mask tuning* process that adjusts the mask variables to ensure that the output signal is recovered for each layer (Section 4.3).
- We extensively test our framework by applying it to $\text{BERT}_{\text{BASE}}$ and DistilBERT on GLUE and SQuAD tasks (Section 5.2). With 1% of accuracy drop, our framework reduces 30–50% of the original FLOPs (Figure 4), resulting in up to 1.56$\times$ speedup on an NVIDIA V100 GPU (Table 3).
- We show that our method achieves comparable or even better FLOPs-accuracy trade-off than prior structured pruning methods *without* retraining (Section 5.3, Figure 5). Our end-to-end pruning finishes in only 39 and 135 seconds on average for GLUE and SQuAD (Section 5.4, Table 5), which is over 100$\times$ faster than the retraining methods.

## 2 Related Works

**Efficient Transformers.** In order to improve the inference speed and reduce the memory footprint of Transformers, multiple different approaches have been proposed. These can be broadly categorized as follows: (i) efficient architecture design [37, 34, 69, 79, 27, 85]; (ii) hardware-software co-design [18, 78, 19, 70]; (iii) knowledge distillation [60, 30, 80, 68]; (iv) quantization [91, 63, 90, 32]; (v) neural architecture search [77, 87, 65, 64, 6, 89]; and (vi) pruning. In this paper, we only focus on pruning and briefly discuss the related works.

**Transformers Pruning.** Pruning has been a popular choice for reducing unimportant weights in Transformers. Pruning can be largely categorized into unstructured and structured pruning. For unstructured pruning, magnitude-based pruning [16], the lottery-ticket hypothesis [15, 55, 8, 9],

movement pruning [61] have been explored for Transformers. While these methods compress the model size, commodity hardware cannot take advantage of the unstructured sparsity for speedup.

For this reason, a number of structured pruning methods have been introduced to remove structured sets of parameters. For example, [49, 75] drop attention heads in multi-head attention layers. Another thread of work prunes entire Transformer blocks based on a simple heuristic [59] or an adaptive training using a layer-wise dropout method [14]. Relatedly, [81] structurally prunes weight matrices via low-rank factorization and $l_0$ regularization; [31, 42] attempt to jointly prune attention heads and filters of weight matrices; and [47, 23] take a step further by dynamically determining the pruning configurations at run time. Recent block pruning schemes chunk weight matrices into blocks and prune them based on group Lasso [41], adaptive regularization [88], and movement pruning [36].

While the structured pruning techniques have achieved high compression rates and speedups, the improvement largely attributes to retraining of the models during or after pruning, often combined with knowledge distillation. However, model retraining introduces several practical problems. Most notably, retraining in general increases training time significantly. For instance, Block Movement Pruning (BMP) [36], a state-of-the-art structured Transformer pruning method, requires $10\times$ longer training time than normal fine-tuning. In addition, the extra hyperparameters introduced by the pruning methods add additional moving parts and lead to increased training cost.

**Post-training Model Compression.** Post-training compression methods have been widely studied in quantization. These methods, categorized as post-training quantization (PTQ), perform quantization without any retraining, thereby minimizing the training cost and user intervention. Multiple approaches have been proposed for PTQ in order to mitigate the accuracy degradation without retraining, including analytic computation of optimal clipping ranges [3], outlier channel splitting [92], and adaptive rounding methods [51, 26].

Although not as much as for quantization, post-training schemes have also been explored for unstructured [25, 38] and structured pruning. For structured pruning, Srinivas and Babu [67] propose to detect and merge similar convolution filters iteratively, and Neuron Merging [33] compensates for the information loss due to pruning by merging similar neurons based on cosine distance. While these methods are effective for CNNs, their applicability is limited to models with simple architectures, and they rely on the characteristics of ReLU nonlinearity [25, 33]. Hence, the prior methods cannot be extended to the general Transformer architectures.

# 3 Overview

## 3.1 Background

**Transformer Architecture.** In this paper, we focus on the pruning of encoder-based Transformer [74] models, especially the BERT [11] architecture family. BERT is a stack of homogeneous Transformer encoder blocks, each of which consists of a multi-head attention (MHA) layer followed by a point-wise Feed-Forward Network (FFN) layer. Specifically, an MHA layer consists of $H$ independently parameterized attention heads:

$$\mathrm{MHA}(\mathrm{x}) = \sum_{i=1}^{H} \mathrm{Att}_i(\mathrm{x}), \quad \mathrm{x}_{\mathrm{MHA}} = \mathrm{LN}\big(\mathrm{x} + \mathrm{MHA}(\mathrm{x})\big),$$

where Att is a dot product attention head, LN is layer normalization, and x is the input sequence. The output of the MHA layer is then fed into the FFN layer, which consists of $N$ filters:

$$\mathrm{FFN}(\mathrm{x}) = \big(\sum_{i=1}^{N} \mathrm{W}^{(2)}_{:,i}\sigma(\mathrm{W}^{(1)}_{i,:}\mathrm{x} + b^{(1)}_i)\big) + b^{(2)}, \quad \mathrm{x}_{\mathrm{out}} = \mathrm{LN}\big(\mathrm{x}_{\mathrm{MHA}} + \mathrm{FFN}(\mathrm{x}_{\mathrm{MHA}})\big),$$

where $\mathrm{W}^{(1)}, \mathrm{W}^{(2)}, b^{(1)}$ and $b^{(2)}$ are the FFN parameters, and $\sigma$ is the activation function, typically GELU [21]. Note that ($H$, $N$) is (12, 3072) for $\mathrm{BERT}_{\mathrm{BASE}}$, and (16, 4096) for $\mathrm{BERT}_{\mathrm{LARGE}}$. We also denote $L$ as the number of Transformer layers.

**Granularity of Pruning.** Our framework considers the structured pruning of both heads in MHA and filters in FFN layers. We do not prune the embedding and the final classifier, as computation of those layers takes a negligible portion of the total inference latency. Since our pruning framework always produces a smaller dense architecture, the model can be readily accelerated without the need for specialized hardware logic, which is often required to gain latency speedup for unstructured sparsity.

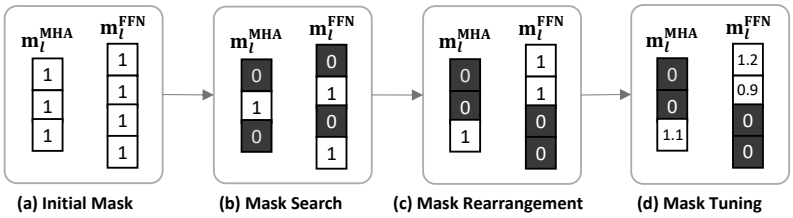


Figure 2: Overview of our pruning framework. (a) The mask variables are initialized as 1. Then they undergo the three-stage pipeline of (b) mask search (Section 4.1), (c) rearrangement (Section 4.2), and (d) tuning (Section 4.3).

**Notations.** For mathematical simplicity, we introduce mask variables associated with the outputs of heads and filters:

$$\text{MHA}(\text{x}; \text{m}_l^{\text{MHA}}) = \sum_{i=1}^{H} m_{l,i}^{\text{MHA}} \circ \text{Att}_i(\text{x}), \quad \text{FFN}(\text{x}; \text{m}_l^{\text{FFN}}) = (\sum_{i=1}^{N} m_{l,i}^{\text{FFN}} \circ \text{W}_{:,i}^{(2)} \sigma(\text{W}_{i,:}^{(1)} \text{x} + b_i^{(1)})) + b^{(2)},$$

where $\text{m}_l^{\text{MHA}} \in \mathbb{R}^H$ and $\text{m}_l^{\text{FFN}} \in \mathbb{R}^N$ are the mask variables for MHA and FFN in the $l$-th layer, respectively, and $m_{l,i}^{\text{MHA}}$ and $m_{l,i}^{\text{FFN}}$ are their $i$-th elements. Furthermore, $\circ$ denotes Hadamard product. Originally, the mask variables are initialized to 1. Zeroing out a mask variable is equivalent to pruning a head/filter associated with it. That is, setting $m_{l,i}^{\text{MHA}}$ and $m_{l,i}^{\text{MHA}}$ as zero is equivalent to pruning the $i$-th head and filter, respectively.

Overall, there are $HL$ head mask variables and $NL$ filter mask variables, summing up to $(H+N)L$ number of total mask variables in a Transformer model. To simplify notations, we additionally define $\text{m}^{\text{MHA}} \in \mathbb{R}^{HL}$, $\text{m}^{\text{FFN}} \in \mathbb{R}^{NL}$, and $\text{m} \in \mathbb{R}^{(H+N)L}$ as the flattened vectors of the head, filter, and total mask variables, respectively, across all layers. In what follows, we discuss how to find the optimal sparse masks under a given cost constraint and how to adjust their values to recover accuracy.

### 3.2 Framework Overview

Figure 1(b) and Figure 2 illustrate the overview of our framework.

**Inputs.** Our framework has 3 inputs: a Transformer model; a sample dataset; and a resource constraint. The input Transformer model should contain weights fine-tuned for a downstream task. The sample dataset is a small partition of the training dataset (typically 1–2K examples) of the downstream task. The resource constraint can be given either as the number of floating point operations (FLOPs) or as an actual latency on target hardware. In the later case, we further assume that a latency lookup table for the target hardware is provided.

**Compression Pipeline.** As illustrated in Figure 2, our framework consists of 3 stages: Fisher-based mask search; Fisher-based mask rearrangement; and mask tuning. During the *Fisher-based mask search* stage (Section 4.1), we search for a binary mask applied to the heads and filters by incorporating the Fisher information of the mask variables. Intuitively, the mask variables with relatively higher Fisher information are considered more important, and they should be less likely to be pruned [39, 50, 43]. As finding the optimal mask that minimizes the Fisher information loss is intractable using the full Fisher matrix, we propose a lightweight search algorithm that finds the optimal mask under reasonable approximations. Second, in the *Fisher-based mask rearrangement* stage (Section 4.2), the framework modifies the searched mask patterns in a layer-wise manner to better take into account the intra-layer interactions of the mask variables. Lastly, in the *mask tuning* stage (Section 4.3), the framework tunes the nonzero mask variables to recover the accuracy drop by reconstructing the layer-wise output signal.

## 4 Methodology

We pose the pruning problem as finding a binary mask to zero out a particular head or filter. After finding the initial set of mask variables, we then allow adaptation/tuning of the remaining mask variables to avoid changes in the output norm of a layer. This process is done without any retraining.

Note that the number of the mask variables is much less than the number of the parameters in a Transformer (e.g., 37K vs. 110M in case of $\text{BERT}_{\text{BASE}}$). This allows the framework to use only a small

number of examples without overfitting to the sample dataset, and thus to be extremely faster than the retraining-based pruning methods which typically use the entire dataset. As the framework keeps the model "as is" and only decides the mask variables, we henceforth regard the model parameters as constants and consider the mask variables as the only parameters for our pruning problem.

**Problem Formulation.** We formulate Transformer pruning as a constrained optimization problem on the mask m:

$$\arg\min_{\mathrm{m}} \mathcal{L}(\mathrm{m}) \quad \text{s.t.} \quad \mathrm{Cost}(\mathrm{m}) \leq C \tag{1}$$

where $\mathcal{L}$ denotes the loss function, Cost is the FLOPs/latency of the architecture pruned by the mask, and $C$ is the given FLOPs/latency constraint. Unfortunately, such a problem is generally intractable as Cost is usually a function of $l_0$-norm of the mask m, which is non-differentiable. Thus, in what follows, we introduce several assumptions and approximations to simplify the problem.

We start by approximating the loss function using the second-order Taylor expansion around the initial mask $\mathbb{1}$:

$$\mathcal{L}(\mathrm{m}) \approx \mathcal{L}(\mathbb{1}) - \mathrm{g}^\intercal(\mathbb{1} - \mathrm{m}) + \frac{1}{2}(\mathbb{1} - \mathrm{m})^\intercal \mathrm{H}(\mathbb{1} - \mathrm{m}) \tag{2}$$

$$\approx \mathcal{L}(\mathbb{1}) + \frac{1}{2}(\mathbb{1} - \mathrm{m})^\intercal \mathrm{H}(\mathbb{1} - \mathrm{m}), \tag{3}$$

where $\mathrm{g} = \mathbb{E}[\frac{\partial}{\partial \mathrm{m}}\mathcal{L}(\mathbb{1})]$ and $\mathrm{H} = \mathbb{E}[\frac{\partial^2}{\partial \mathrm{m}^2}\mathcal{L}(\mathbb{1})]$. Eq. 3 is deduced from an assumption that the model has converged to a local minima, where the gradient term is close to 0 [39]. As $\mathcal{L}(\mathbb{1})$ is a constant, we can rewrite the optimization objective as follows:

$$\arg\min_{\mathrm{m}} \mathcal{L}(\mathrm{m}) \approx \arg\min_{\mathrm{m}} (\mathbb{1} - \mathrm{m})^\intercal \mathrm{H}(\mathbb{1} - \mathrm{m}). \tag{4}$$

Eq. 4 shows that the optimal mask is determined by the Hessian of the loss with respect to the mask variables. Since forming the exact Hessian matrix explicitly is infeasible, we approximate the Hessian H with the empirical Fisher information matrix $\mathcal{I}$ of the mask variables:

$$\mathcal{I} := \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \left(\frac{\partial}{\partial \mathrm{m}}\mathcal{L}(x, y; \mathbb{1})\right)\left(\frac{\partial}{\partial \mathrm{m}}\mathcal{L}(x, y; \mathbb{1})\right)^\intercal, \tag{5}$$

where $\mathcal{D}$ is the sample dataset and $(x, y)$ is a tuple of an input example and its label.

### 4.1 Fisher-Based Mask Search

**Diagonal Approximation of Fisher Information Matrix.** It is intractable to solve the optimization objective in Eq. 4 using the full empirical Fisher information matrix $\mathcal{I}$. Thus, we first make a simple assumption that $\mathcal{I}$ is *diagonal*. This further simplifies Eq. 4 as follows:

$$\arg\min_{\mathrm{m}} \mathcal{L}(\mathrm{m}) \approx \arg\min_{\mathrm{m}} \sum_i (1 - m_i)^2 \mathcal{I}_{ii}, \tag{6}$$

Since we restrict the possible mask values to either 0 or 1, the following can be derived from Eq. 6:

$$\arg\min_{\mathrm{m}} \mathcal{L}(\mathrm{m}) \approx \arg\min_{\mathrm{m}} \sum_{i\in Z(\mathrm{m})} \mathcal{I}_{ii} \quad \text{where} \quad Z(\mathrm{m}) := \{i \mid m_i = 0\}. \tag{7}$$

We can interpret the diagonal element of $\mathcal{I}$ as the *importance score* of the head/filter associated with each mask variable, and Eq. 7 as a process of minimizing the total importance scores of the pruned heads and filters. Such an importance score has also been introduced in [72, 50] to guide pruning.

**Solving FLOPs-constrained Problem.** We need to solve Eq. 7 given a cost constraint. For a given target FLOP cost, denoted by C, we can formulate the binary mask search problem as follows:

$$\arg\min_{\mathrm{m}} \sum_{i\in Z(\mathrm{m})} \mathcal{I}_{ii} \quad \text{s.t.} \quad \mathrm{F}_{\text{head}}||\mathrm{m}^{\text{MHA}}||_0 + \mathrm{F}_{\text{filter}}||\mathrm{m}^{\text{FFN}}||_0 \leq C, \tag{8}$$

where $\mathrm{F}_{\text{head}} \in \mathbb{R}$ and $\mathrm{F}_{\text{filter}} \in \mathbb{R}$ are the FLOPs for computing a head and a filter, respectively. Note that the number of FLOPs of a head/filter is constant across all layers. While such an optimization

**Algorithm 1** Mask Search with a FLOPs Constraint

**Input:** FLOPs constraint $C$, diagonal Fisher information matrix $\mathcal{I}$

1: **for** $n = 0$ **to** $HL$ **do** ▷ # remaining heads
2: $k_1 = HL - n$ ▷ # heads to prune
3: HI = indicies of $k_1$ least important heads
4: $f = \lfloor (C - n\mathrm{F}_{\text{head}})/\mathrm{F}_{\text{filter}} \rfloor$ ▷ # remaining filters
5: $k_2 = NL - f$ ▷ # filters to prune
6: FI = indicies of $k_2$ least important filters
7: $S[n] = \sum_{i \in \text{HI} \cup \text{FI}} \mathcal{I}_{ii}$, $R[n] = (\text{HI}, \text{FI})$
8: **end for**
9: $n^* = \arg\min_n S[n]$ ▷ optimal # remaining heads
10: $\text{HI}^*, \text{FI}^* = R[n^*]$ ▷ indicies of heads/filters to prune
11: Initialize $\mathrm{m}^{\text{MHA}}$ and $\mathrm{m}^{\text{FFN}}$ as $\mathbb{1}$
12: $\mathrm{m}^{\text{MHA}}[\text{HI}^*] = 0$, $\mathrm{m}^{\text{FFN}}[\text{FI}^*] = 0$ ▷ prune the selected heads and filters

**Output:** $\mathrm{m}^* = (\mathrm{m}^{\text{MHA}}, \mathrm{m}^{\text{FFN}})$

problem can be generally solved by a knapsack algorithm [1, 62], the following observations allow a simpler and faster solution: (1) having more heads and filters unpruned always optimizes Eq. 8 since the diagonal elements of $\mathcal{I}$ are non-negative; and (2) if a certain number of heads needs to be pruned, they should be the ones with the lowest importance scores because each head accounts for the same amount of FLOPs. The same statement also holds for pruning filters. These lead to our mask search algorithm described in Algorithm 1.

Algorithm 1 partitions the solution space by the number of remaining heads in the pruned architecture ($n$ in line 1). For each $n$, by observation (1), the number of remaining neurons should be as line 4. Then by observation (2) the heads/filters with the lowest important scores are selected to be pruned. $S[n]$ is the evaluation of Eq. 8 with this mask. When the loop terminates, the output is the mask with the smallest $S[n]$. In Section A.1, we prove that the output mask $\mathrm{m}^*$ of Algorithm 1 is optimal. That is, any other mask m satisfying the given FLOPs constraint will have a higher loss:

$$\sum_{i \in Z(\mathrm{m}^*)} \mathcal{I}_{ii} \leq \sum_{i \in Z(\mathrm{m})} \mathcal{I}_{ii}. \tag{9}$$

**Solving Latency-constrained Problem.** If the cost constraint is given in terms of an actual latency on target hardware, we have a new cost constraint formula in the optimization target in Eq. 8:

$$\sum_{l=1}^{L} \mathrm{LAT}(\mathrm{m}_l^{\text{MHA}}) + \sum_{l=1}^{L} \mathrm{LAT}(\mathrm{m}_l^{\text{FFN}}) \leq C, \tag{10}$$

where the function LAT indicates the latency of a MHA/FFN layer after pruning. We assume that a latency lookup table on the target hardware is provided so that evaluating LAT takes negligible time.

Unfortunately, the latency constraint makes the problem more challenging such that it cannot be solved by directly applying Algorithm 1. This is because LAT is *not* linear to the number of remaining heads or filters after pruning [56], as shown in Figure 3 (Left). We can interpret this as follows: (1) with a sufficient number of heads/filters in a layer, the hardware resources such as parallel cores can be fully utilized, resulting in latency roughly proportional to the number of heads/filters; and (2) otherwise, the hardware is underutilized and a constant overhead dominates the latency [35, 48]. Thus, pruning more heads/filters below a certain threshold does not translate into actual speedup.

Based on the above analysis, we approximate LAT as a piece-wise linear function as in Figure 3 (Right) such that $\mathrm{LAT}(\mathrm{m}_l)$ is 0 if $||\mathrm{m}_l||_0 = 0$, $c$ if $0 < ||\mathrm{m}_l||_0 \leq T$, and $a(||\mathrm{m}_l||_0 - T) + c$ if $||\mathrm{m}_l||_0 > T$, where $c \in \mathbb{R}$ is the constant overhead, $T \in \mathbb{N}$ is the threshold number of heads/filters that the latency starts to be linear, and $a \in \mathbb{R}$ is the slope of the linear part. This can be easily obtained by fitting the actual latency in the lookup table with the minimum mean squared error.

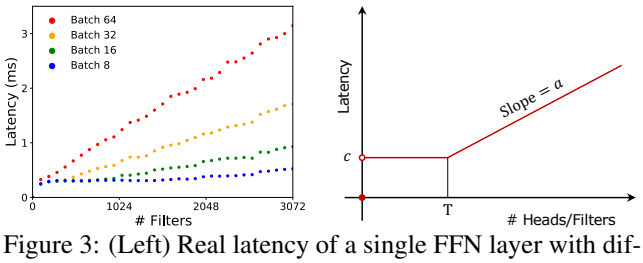


Figure 3: (Left) Real latency of a single FFN layer with different numbers of remaining filters. (Right) Schematic plot for the approximated latency as a piece-wise linear function.

This LAT approximation allows us to extend Algorithm 1 to solving the problems with latency constraints. The core idea is to consider separately the constant part of LAT and the linear part of LAT; after handling the constant part, we can apply the Algorithm 1 to the linear part. The detailed modification to Algorithm 1 is described in Section A.2.

### 4.2 Fisher-based Mask Rearrangement

**Block Diagonal Approximation of Fisher Information Matrix.** Although it simplifies the problem, the diagonal assumption in Section 4.1 alone might not find the best solution, as it does not take into account the interactions between different mask variables. We can better capture the interactions by using a block diagonal approximation to the Fisher operator, where a block corresponds to a MHA layer or a FFN layer. However, the block diagonal approximation results in an intractable optimization problem over the binary mask. To alleviate this, we use the results from the previous step to *warm start* the optimization problem with the block-diagonal approximation. That is, given the mask $\mathrm{m}^*$ obtained in Section 4.1, we constrain $||\mathrm{m}_l||_0$ to be equal to $||\mathrm{m}_l^*||_0$ for all layers $l$.

Given the two assumptions, that (i) there is no interaction between the mask variables in different layers (i.e., the block diagonal approximation), and (ii) the number of pruned heads/filters are predetermined for each layer (i.e., warm-start), Eq. 4 breaks down to a set of *layer-wise* optimization problems, as follows based on the derivation in Section A.3:

$$\hat{\mathrm{m}}_l = \arg\min_{\mathrm{m}_l} (\mathbb{1} - \mathrm{m}_l)^\intercal \mathcal{I}_l (\mathbb{1} - \mathrm{m}_l), \tag{11}$$

where $\mathcal{I}_l$ is the $l$-th diagonal block of $\mathcal{I}$. This can be approximately solved by greedily exploring the possible binary masks for $\mathrm{m}_l$ since $||\mathrm{m}_l||_0$ is fixed. Because this process does not change the number of heads/filters in each layer, the searched mask leads to the same FLOPs/latency as the one obtained in Section 4.1. In effect, this process *rearranges* the binary mask variables of each layer to find a better arrangement for pruning locations.

### 4.3 Mask Tuning

In the previous two stages, the possible mask values are restricted to either 0 or 1 in order to simplify the search process. In this stage, we further relax this restriction. The *nonzero* variables in the mask $\hat{\mathrm{m}}$ from Section 4.2 are tuned to any real values such that the pruned model recovers its accuracy.

**Layer-wise Reconstruction via Linear Least Squares.** We tune the mask variables toward minimizing the *layer-wise reconstruction error*, similarly to [20]. For each layer, we reconstruct the output activation of the original model with the remaining heads/filters in the pruned model. This can be formally written as follows:

$$\arg\min_{\mathrm{m}_l} ||\mathrm{x} + \mathrm{layer}(\mathrm{x}; \mathrm{m}_l) - (\mathrm{x}' + \mathrm{layer}(\mathrm{x}'; \mathbb{1}))||_2^2, \tag{12}$$

where layer is the either MHA or FFN, and x and $\mathrm{x}'$ are the inputs to the layer of the pruned model and the original model, respectively. Note that this stage does not incur any change in model FLOPs/latency, as we only tune the non-zero valued mask variables. We show in Section A.4 that Eq. 12 can be reduced to a linear least squares problem of $\arg\min_{\mathrm{m}_l} ||\mathrm{A}\mathrm{m}_l - \mathrm{b}||_2^2$, where the matrix A denotes a collection of the output activations of the model pruned by the binary mask and the vector b is the output activation of the original model.

Because the size of the matrix A can be large, our framework uses the LSMR solver in CuPy [52] to solve the linear least squares problem. For the regularization hyperparameter (i.e., `damp`) of LSMR, we fix its value to 1. Then, to increase stability, we restrict the acceptable range of the tuned mask variables to [-10, 10]. When the solver finds a layer mask that exceeds this range, we discard the mask for that layer and stop mask tuning. While the use of LSMR solver involves two hyperparameters (i.e., `damp` and the acceptable range), we empirically find that these need not be tuned for different tasks and models. In all of our experiments, we used the fixed hyperparameter values.

## 5 Evaluation

### 5.1 Experimental Setup

Our framework is implemented on top of PyTorch [54] and HuggingFace Transformers [84] library. We evaluate the effectiveness of our approach using $\mathrm{BERT}_{\mathrm{BASE}}$ [11] and DistilBERT [60] on GLUE [76] and SQuAD [58, 57] benchmarks. We use 2K examples from the training sets for pruning, and we evaluate the resulting models on the development sets. All of the results are averaged over the runs with 10 different seeds. More details on the experimental setup can be found in Section A.5.

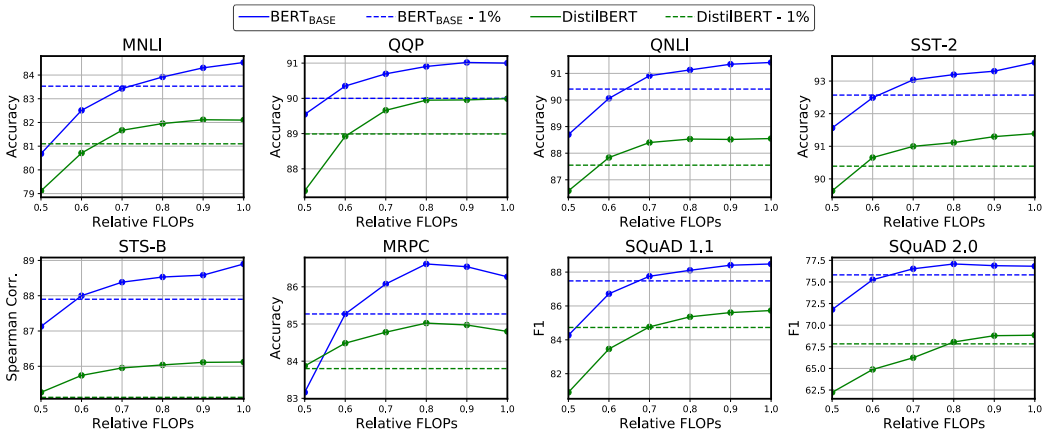


Figure 4: Accuracy of our pruning method applied to $\text{BERT}_{\text{BASE}}$ and DistilBERT with different FLOPs constraints. The dashed horizontal lines indicate 1% accuracy drop from the baseline models. Note that these results can be achieved in only 39 and 135 seconds for GLUE and SQuAD benchmarks, respectively, on a single GPU system, as described in Table 5 (in Section A.11).

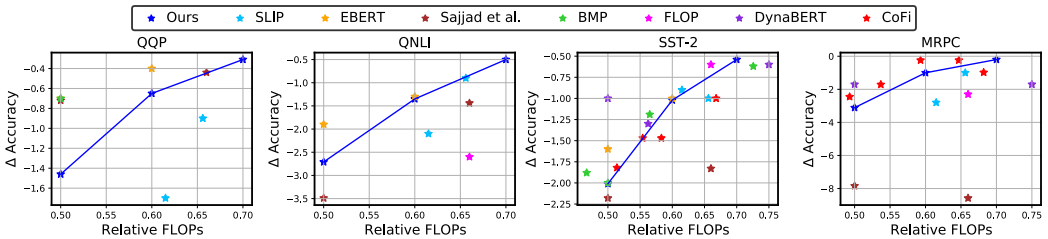


Figure 5: Amount of accuracy degradation from the baseline when pruning $\text{BERT}_{\text{BASE}}$ using our method and the prior structured pruning methods with different relative FLOPs. Note that our method does not require retraining, whereas all the other methods involve significant retraining overheads as described in Table 1.

### 5.2 Performance Evaluation

**FLOPs.** Figure 4 shows the accuracy of $\text{BERT}_{\text{BASE}}$ and DistilBERT with different FLOPs constraints on GLUE and SQuAD datasets. As can be seen in the plots, with only 1% of accuracy drop, $\text{BERT}_{\text{BASE}}$ achieves 60–70% of the original FLOPs for all tasks. DistilBERT also shows a similar pattern and shows 50% FLOPs reduction (in STS-B and MRPC) even though it is already a compressed architecture. Results with larger sample datasets are provided in Section A.6.

**Latency.** We further measure the latency on real hardware by pruning $\text{BERT}_{\text{BASE}}$ with latency constraints and deploying the resulting models on an NVIDIA V100 GPU. Table 3 (in Section A.7) lists the latency speedup with maximum accuracy drop of 1% for GLUE and SQuAD datasets. With batch size of 256, we achieve speedup of 1.47× on average and up to 1.56×.

### 5.3 Comparison with the Prior Methods

**FLOPs and Accuracy Comparison.** Here, we compare our method with the prior structured pruning methods for Transformers including Flop [81], SLIP [71], Sajjad et al. [59], DynaBERT [23], EBERT [47], Block Movement Pruning (BMP) [36], and CoFi [86] by the FLOPs-accuracy trade-off of $\text{BERT}_{\text{BASE}}$ on GLUE tasks. We use the results *without* knowledge distillation and data augmentation reported in each paper. Since the baseline accuracy differs slightly from paper to paper, we compare the amount of the accuracy drop from the baseline instead of the absolute accuracy. The results are plotted as Figure 5. We include the comparison details and full table in Section A.8.

Interestingly, our method exhibits comparable or sometimes better results than the prior methods *without* any model retraining and with substantially lower pruning costs. This empirically demonstrates that retraining and a complex pruning pipeline are not necessary for moderate level of pruning of Transformers. For high sparsity, we find that our framework *with* retraining works comparably to or better than the prior methods at the same pruning cost (See Section A.9).

Table 1: Pruning cost comparison between the prior structured pruning methods and ours. We compare the number of training epochs and the end-to-end (E2E) time required for pruning.

| | # Epochs | E2E time (hr) |
|---|---|---|
| DynaBERT [23] | 4 | 12 |
| EBERT [47] | 6 | 5 |
| BMP [36] | 20 | 17 |
| CoFi [86] | 40 | 33 |
| Ours | **0** | **0.01** |

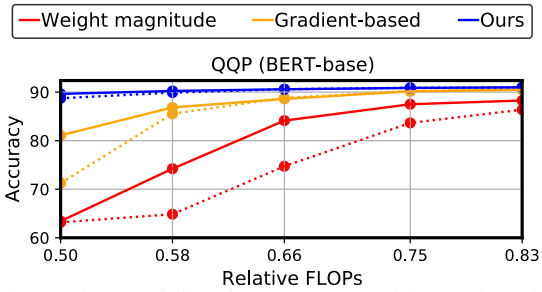

Figure 6: Retraining-free accuracy without (dotted) and with (solid) mask tuning.

Table 2: Ablation of our mask search, rearrangement, and tuning methods, described in Section 4. We use $\text{BERT}_{\text{BASE}}$ as a baseline model, and we prune with a 60% FLOPs constraint.

| | MNLI | QQP | QNLI | SST-2 | STS-B | MRPC | $\text{SQuAD}_{1.1}$ | $\text{SQuAD}_{2.0}$ | Avg. Diff |
|---|---|---|---|---|---|---|---|---|---|
| Baseline | 84.53 | 91.00 | 91.41 | 93.57 | 88.90 | 86.27 | 88.48 | 76.82 | |
| Mask Search | 81.21 | 89.99 | 88.38 | 92.13 | 87.10 | 83.14 | 82.66 | 71.12 | |
| + Mask Rearrangement | 81.81 | 90.08 | 88.77 | 92.09 | 87.68 | 83.23 | 84.47 | 72.38 | + 0.60 |
| + Mask Tuning | **82.51** | **90.35** | **90.06** | **92.49** | **88.00** | **85.27** | **86.72** | **75.26** | + 1.27 |

**Retraining Cost.** We select DynaBERT [23], EBERT [47], BMP [36], and CoFi [86] that achieve comparably good accuracy in Figure 5, and we systematically analyze their end-to-end retraining costs on the MNLI dataset. As shown in Table 1, these methods require 5−33 hours of retraining. On the other hand, our method finishes in less than a minute, which is 2−3 orders of magnitude faster. We also highlight that this training latency only accounts for a *single* hyperparameter, and the entire cost should be multiplied by the size of the hyperparameter space. While the prior methods rely on a number of hyperparameters, ours introduce only two hyperparameters (in Section 4.3) which we fix for all experiments. See Section A.10 for more details.

Unlike other methods (including ours) that take as input the fine-tuned models, Sajjad et al. [59] starts from the pre-trained model, heuristically drops the top layers, and then fine-tunes on downstream tasks. Therefore, one can regard it as a zero-cost pruning method. However, as can be seen in Figure 5, the resulting models suffer from large accuracy degradation in most cases.

### 5.4 Discussion

**Ablation Studies.** Table 2 lists an ablation of mask rearrangement (Section 4.2) and tuning (Section 4.3) stages for pruned $\text{BERT}_{\text{BASE}}$ with 60% of FLOPs. We find that both stages help recover the baseline accuracy, and that mask tuning is in particular critical, recovering up to 2.88% accuracy. To further investigate the importance of mask search and rearrangement, we compare the *retraining-free* performance of the binary masks obtained by our method and other pruning criteria: weight magnitude and gradient-based method used in DynaBERT. We uniformly pruned the layers using the two criteria with different width multipliers. Figure 6 shows that the two methods significantly degrade the accuracy under non-trivial sparsity. Even with mask tuning, the accuracy is not fully recovered. The results demonstrate that our mask search and re-arrangement are necessary to get optimal binary masks, and that mask tuning is only effective when the mask mostly preserves the accuracy.

**Time Breakdown.** We break down our pruning pipeline into 4 parts—gradient computation, mask search, rearrangement, and tuning—and we measure the latency for each stage as Table 5 (Section A.11). For GLUE and SQuAD tasks, our framework finishes in 39 and 135 seconds on average.

## 6 Conclusion

In this work, we have proposed a novel post-training pruning framework for Transformers that does not require model retraining. The core of our framework is to the three-stage decomposition of the pruning process. It uses a fast Fisher-based mask search algorithm to decide which heads/filters to prune, rearranges the pruned heads/filters, and tunes the mask variables to recover the output signal for each layer. We empirically evaluate our framework using $\text{BERT}_{\text{BASE}}$ and DistilBERT, where our pruning method achieves up to 50% FLOPs reduction with only 1% accuracy degradation on GLUE and SQuAD datasets. This results in up to 1.56× latency speedup on an NVIDIA V100 GPU. End-to-end pruning only needs 39 and 135 seconds for GLUE and SQuAD, which is 2−3 orders of magnitude faster than the prior methods. Overall, our method shows comparable or even better compression performance, as compared to the prior retraning-based methods.

# References


[1] Yonathan Aflalo, Asaf Noy, Ming Lin, Itamar Friedman, and Lihi Zelnik. Knapsack pruning with inner distillation. *arXiv preprint arXiv:2002.08258*, 2020.

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

[3] Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. Post-training 4-bit quantization of convolution networks for rapid-deployment. *arXiv preprint arXiv:1810.05723*, 2018.

[4] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[5] Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*, 2017.

[6] Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. Adabert: Task-adaptive bert compression with differentiable neural architecture search. *arXiv preprint arXiv:2001.04246*, 2020.

[7] Sanyuan Chen, Chengyi Wang, Zhengyang Chen, Yu Wu, Shujie Liu, Zhuo Chen, Jinyu Li, Naoyuki Kanda, Takuya Yoshioka, Xiong Xiao, et al. Wavlm: Large-scale self-supervised pre-training for full stack speech processing. *arXiv preprint arXiv:2110.13900*, 2021.

[8] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained BERT networks. *arXiv preprint arXiv:2007.12223*, 2020.

[9] Xiaohan Chen, Yu Cheng, Shuohang Wang, Zhe Gan, Zhangyang Wang, and Jingjing Liu. Earlybert: Efficient bert training via early-bird lottery tickets. *arXiv preprint arXiv:2101.00063*, 2020.

[10] Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, pages 177–190. Springer, 2005.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[12] William B Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

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

[14] Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*, 2019.

[15] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

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

[17] Google. Tensorflow Lite: https://www.tensorflow.org/lite, 2017.

[18] Tae Jun Ham, Sung Jun Jung, Seonghak Kim, Young H Oh, Yeonhong Park, Yoonho Song, Jung-Hun Park, Sanghee Lee, Kyoung Park, Jae W Lee, et al. Aˆ 3: Accelerating attention mechanisms in neural networks with approximation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 328–341. IEEE, 2020.

[19] Tae Jun Ham, Yejin Lee, Seong Hoon Seo, Soosung Kim, Hyunji Choi, Sung Jun Jung, and Jae W Lee. Elsa: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 692–705. IEEE, 2021.

[20] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017.

[21] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[22] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[23] Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. Dynabert: Dynamic bert with adaptive width and depth. *arXiv preprint arXiv:2004.04037*, 2020.

[24] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. Hubert: Self-supervised speech representation learning by masked prediction of hidden units. *arXiv preprint arXiv:2106.07447*, 2021.

[25] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[26] Itay Hubara, Yury Nahshan, Yair Hanani, Ron Banner, and Daniel Soudry. Improving post training neural quantization: Layer-wise calibration and integer programming. *arXiv preprint arXiv:2006.10518*, 2020.

[27] Forrest N Iandola, Albert E Shaw, Ravi Krishna, and Kurt W Keutzer. Squeezebert: What can computer vision teach nlp about efficient neural networks? *arXiv preprint arXiv:2006.11316*, 2020.

[28] Intel. OpenVINO: https://docs.openvino.ai/latest/index.html, 2021.

[29] Shankar Iyer, Nikhil Dandekar, and Kornl Csernai. First quora dataset release: Question pairs.(2017). *URL https://data. quora. com/First-Quora-Dataset-Release-Question-Pairs*, 2017.

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

[31] Ashish Khetan and Zohar Karnin. schubert: Optimizing elements of bert. *arXiv preprint arXiv:2005.06628*, 2020.

[32] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization. *arXiv preprint arXiv:2101.01321*, 2021.

[33] Woojeong Kim, Suhyun Kim, Mincheol Park, and Geonseok Jeon. Neuron merging: Compensating for pruned neurons. *arXiv preprint arXiv:2010.13160*, 2020.

[34] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2019.

[35] Woosuk Kwon, Gyeong-In Yu, Eunji Jeong, and Byung-Gon Chun. Nimble: Lightweight and parallel gpu task scheduling for deep learning. *arXiv preprint arXiv:2012.02732*, 2020.

[36] François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M Rush. Block pruning for faster transformers. *arXiv preprint arXiv:2109.04838*, 2021.

[37] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[38] Ivan Lazarevich, Alexander Kozlov, and Nikita Malinin. Post-training deep neural network pruning via layer-wise calibration. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 798–805, 2021.

[39] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[40] Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*. Citeseer, 2012.

[41] Bingbing Li, Zhenglun Kong, Tianyun Zhang, Ji Li, Zhengang Li, Hang Liu, and Caiwen Ding. Efficient transformer-based large scale language representations using hardware-friendly block structured pruning. *arXiv preprint arXiv:2009.08065*, 2020.

[42] Zi Lin, Jeremiah Zhe Liu, Zi Yang, Nan Hua, and Dan Roth. Pruning redundant mappings in transformer models via spectral-normalized identity prior. *arXiv preprint arXiv:2010.01791*, 2020.

[43] Liyang Liu, Shilong Zhang, Zhanghui Kuang, Aojun Zhou, Jing-Hao Xue, Xinjiang Wang, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, pages 7021–7032. PMLR, 2021.

[44] 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. *arXiv preprint arXiv:1907.11692*, 2019.

[45] Yuanxin Liu, Zheng Lin, and Fengcheng Yuan. Rosita: Refined bert compression with integrated techniques. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 8715–8722, 2021.

[46] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021.

[47] Zejian Liu, Fanrong Li, Gang Li, and Jian Cheng. Ebert: Efficient bert inference with dynamic structured pruning. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4814–4823, 2021.

[48] Lingxiao Ma, Zhiqiang Xie, Zhi Yang, Jilong Xue, Youshan Miao, Wei Cui, Wenxiang Hu, Fan Yang, Lintao Zhang, and Lidong Zhou. Rammer: Enabling holistic deep learning compiler optimizations with rtasks. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, pages 881–897, 2020.

[49] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *arXiv preprint arXiv:1905.10650*, 2019.

[50] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

[51] Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning*, pages 7197–7206. PMLR, 2020.

[52] ROYUD Nishino and Shohei Hido Crissman Loomis. Cupy: A numpy-compatible library for nvidia gpu calculations. *31st confernce on neural information processing systems*, 151, 2017.

[53] NVIDIA. TensorRT: https://developer.nvidia.com/tensorrt, 2018.

[54] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[55] Sai Prasanna, Anna Rogers, and Anna Rumshisky. When BERT plays the lottery, all tickets are winning. *arXiv preprint arXiv:2005.00561*, 2020.

[56] Valentin Radu, Kuba Kaszyk, Yuan Wen, Jack Turner, José Cano, Elliot J Crowley, Björn Franke, Amos Storkey, and Michael O'Boyle. Performance aware convolutional neural network channel pruning for embedded gpus. In *2019 IEEE International Symposium on Workload Characterization (IISWC)*, pages 24–34. IEEE, 2019.

[57] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

[58] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[59] Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. On the effect of dropping layers of pre-trained transformer models. *arXiv preprint arXiv:2004.03844*, 2020.

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

[61] Victor Sanh, Thomas Wolf, and Alexander M Rush. Movement pruning: Adaptive sparsity by fine-tuning. *arXiv preprint arXiv:2005.07683*, 2020.

[62] Maying Shen, Hongxu Yin, Pavlo Molchanov, Lei Mao, Jianna Liu, and Jose M Alvarez. Halp: Hardware-aware latency pruning. *arXiv preprint arXiv:2110.10811*, 2021.

[63] 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 *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821, 2020.

[64] David So, Quoc Le, and Chen Liang. The evolved transformer. In *International Conference on Machine Learning*, pages 5877–5886. PMLR, 2019.

[65] David R So, Wojciech Mańke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V Le. Primer: Searching for efficient transformers for language modeling. *arXiv preprint arXiv:2109.08668*, 2021.

[66] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[67] Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.

[68] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355*, 2019.

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

[70] Thierry Tambe, Coleman Hooper, Lillian Pentecost, Tianyu Jia, En-Yu Yang, Marco Donato, Victor Sanh, Paul Whatmough, Alexander M Rush, David Brooks, et al. Edgebert: Sentence-level energy optimizations for latency-aware multi-task nlp inference. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 830–844, 2021.

[71] Hidenori Tanaka, Daniel Kunin, Daniel LK Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *arXiv preprint arXiv:2006.05467*, 2020.

[72] Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787*, 2018.

[73] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

[74] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[75] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*, 2019.

[76] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[77] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. Hat: Hardware-aware transformers for efficient natural language processing. *arXiv preprint arXiv:2005.14187*, 2020.

[78] Hanrui Wang, Zhekai Zhang, and Song Han. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 97–110. IEEE, 2021.

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

[80] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *arXiv preprint arXiv:2002.10957*, 2020.

[81] Ziheng Wang, Jeremy Wohlwend, and Tao Lei. Structured pruning of large language models. *arXiv preprint arXiv:1910.04732*, 2019.

[82] Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019.

[83] Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

[84] Thomas Wolf, Julien Chaumond, Lysandre Debut, Victor Sanh, Clement Delangue, Anthony Moi, Pierric Cistac, Morgan Funtowicz, Joe Davison, Sam Shleifer, et al. 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, 2020.

[85] Zhanghao Wu, Zhijian Liu, Ji Lin, Yujun Lin, and Song Han. Lite transformer with long-short range attention. *arXiv preprint arXiv:2004.11886*, 2020.

[86] Mengzhou Xia, Zexuan Zhong, and Danqi Chen. Structured pruning learns compact and accurate models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1513–1528, 2022.

[87] Jin Xu, Xu Tan, Renqian Luo, Kaitao Song, Jian Li, Tao Qin, and Tie-Yan Liu. Nas-bert: Task-agnostic and adaptive-size bert compression with neural architecture search. *arXiv preprint arXiv:2105.14444*, 2021.

[88] Zhewei Yao, Linjian Ma, Sheng Shen, Kurt Keutzer, and Michael W Mahoney. Mlpruning: A multilevel structured pruning framework for transformer-based models. *arXiv preprint arXiv:2105.14636*, 2021.

[89] Yichun Yin, Cheng Chen, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. Autotinybert: Automatic hyper-parameter optimization for efficient pre-trained language models. *arXiv preprint arXiv:2107.13686*, 2021.

[90] Ali Hadi Zadeh, Isak Edo, Omar Mohamed Awad, and Andreas Moshovos. Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 811–824. IEEE, 2020.

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

[92] Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Christopher De Sa, and Zhiru Zhang. Improving neural network quantization without retraining using outlier channel splitting. *Proceedings of Machine Learning Research*, 2019.

# Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] The notations are defined in Section 3.1 and our problem is formally described in Section 4.
   (b) Did you include complete proofs of all theoretical results? [Yes] Section A.1 contains the proof of the optimiality of Algorithm 1. Section A.3 and Section A.4 describe the derivation processes omitted in the paper.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Our code is included in the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Our method has only two hyperparameters which were fixed in all of our experiments (See Section 4.3). For details on the datasets, see Section 5.1.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Because our method is training-free, the results are very stable across seeds. For simplicity, we omitted the error bar.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] For all experiments, we used an AWS p3.2xlarge instance which has 1 NVIDIA V100 GPU. See Section 5.1 for details.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] We cited the HuggingFace transformers library, from which we got the pre-trained Transformer weights.

(b) Did you mention the license of the assets? [N/A]

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]