EfficientBERT: Progressively Searching Multilayer Perceptron via Warm-up Knowledge Distillation

Chenhe Dong[1], **Guangrun Wang**[2], **Hang Xu**[3], **Jiefeng Peng**[4],
Xiaozhe Ren[3], **Xiaodan Liang**[1]*
[1] Shenzhen Campus of Sun Yat-sen University [2] University of Oxford
[3] Huawei Noah's Ark Lab [4] DarkMatter AI Research
dongchh@mail2.sysu.edu.cn, {xu.hang,renxiaozhe}@huawei.com
{wanggrun,jiefengpeng,xdliang328}@gmail.com

Abstract

Pre-trained language models have shown remarkable results on various NLP tasks. Nevertheless, due to their bulky size and slow inference speed, it is hard to deploy them on edge devices. In this paper, we have a critical insight that improving the feed-forward network (FFN) in BERT has a higher gain than improving the multi-head attention (MHA) since the computational cost of FFN is 2∼3 times larger than MHA. Hence, to compact BERT, we are devoted to designing efficient FFN as opposed to previous works that pay attention to MHA. Since FFN comprises a multilayer perceptron (MLP) that is essential in BERT optimization, we further design a thorough search space towards an advanced MLP and perform a coarse-to-fine mechanism to search for an efficient BERT architecture. Moreover, to accelerate searching and enhance model transferability, we employ a novel warm-up knowledge distillation strategy at each search stage. Extensive experiments show our searched EfficientBERT is 6.9× smaller and 4.4× faster than $\text{BERT}_{\text{BASE}}$, and has competitive performances on GLUE and SQuAD Benchmarks. Concretely, EfficientBERT attains a 77.7 average score on GLUE *test*, 0.7 higher than $\text{MobileBERT}_{\text{TINY}}$, and achieves an 85.3/74.5 F1 score on SQuAD v1.1/v2.0 *dev*, 3.2/2.7 higher than TinyBERT_4 even without data augmentation. The code is released at https://github.com/cheneydon/efficient-bert.

1 Introduction

Diverse pre-trained language models (PLMs) (e.g., BERT (Devlin et al., 2019)) have been intensively investigated by designing new pretext tasks, architectures, or attention mechanisms (Yang et al., 2019; Jiang et al., 2020; Beltagy et al., 2020). The performances of these PLMs far exceed the traditional methods on a variety of natural language processing (NLP) tasks. Nevertheless, their shortcomings are still evident (including a considerable model size and low inference efficiency), limiting real-world application scenarios.

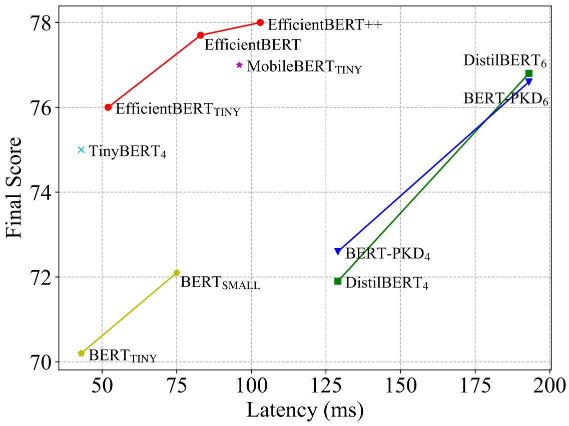


Figure 1: Final score vs. latency tradeoff curve. Final score refers to the average score on the GLUE test set.

To alleviate the aforementioned limitations, many model compression methods have been proposed, including quantization, weight pruning, and knowledge distillation (KD) (Shen et al., 2020; Michel et al., 2019; Jiao et al., 2020). Among them, KD (Hinton et al., 2015) that transfers the knowledge from larger teacher models to smaller student models with minimal performance sacrifice is most widely used due to its plug-and-play feasibility and its scalability in the rapid delivery of new models. Specifically, KD allows us to train our own BERT architecture significantly faster than training from scratch. Hence, we adopt KD in this paper. Besides, inspired by the impressive results by neural architecture search (NAS) in vision tasks (Howard et al., 2019; Li et al., 2020; Cai et al., 2020), adopting NAS to further boost the performance of PLMs or reduce the computational cost has attracted increasing attentions (So et al., 2019; Wang et al., 2020a; Chen et al., 2020).

Although considerable progress has been made

[*] Corresponding author.

Findings of the Association for Computational Linguistics: EMNLP 2021, pages 1424–1437
November 7–11, 2021. ©2021 Association for Computational Linguistics

in the field of KD for PLMs, the compression of the feed-forward network (FFN) has been rarely studied. This contradicts the fact that the computational cost of FFN is 2∼3 times larger than that of the multi-head attention (MHA). In addition, Dong et al. (2021) have proved that the multilayer perceptron (MLP) in FFN can prevent the undesirable rank collapse caused by self-attention and thus can improve BERT optimization. These motivate us to investigate the nonlinearity of FFN in BERT.

In this paper, we make the first attempt to compress and improve the barely-explored multilayer perceptron (MLP) in FFN and propose a novel coarse-to-fine NAS approach with warm-up KD to find the optimal MLP architectures, aiming to search for a universal small BERT model with competitive performance and strong transferability. Specifically, we design a rich and flexible search space to discover an excellent FFN with maximal nonlinearity and a minimal computational cost. Our search space contains various mathematical operations, stack numbers, and expansion ratios of intermediate hidden size. To efficiently search from our vast search space, we progressively shrink the search space in three stages.

- **Stage 1:** Perform a coarse search to explore the entire search space (i.e., jointly searching the mathematical operations, stack numbers, and expansion ratios)(Figure 2 (a)).
- **Stage 2:** Fix the stack numbers and expansion ratios, performing a fine-grained search for the optimal mathematical operations (Figure 2 (b)).
- **Stage 3:** Fix the mathematical operations, performing a fine-grained search for optimal stack numbers and expansion ratios (Figure 2 (c)).

Even with this elegant coarse-to-fine search strategy, pre-training each candidate model still needs a lot of time to converge during searching. To solve this problem, different from the conventional KD strategy (Jiao et al., 2020), we propose a warm-up KD strategy to fast transfer the knowledge, where a pre-trained supernet is additionally introduced to perform a joint warm-up for all candidate models. Note that the warm-up strategy in the third stage is slightly different from that of the first two stages. During the first two stages, each candidate model initially inherits its weights from a frozen warmed-up supernet to accelerate searching. But in the third stage, since there is no need to search mathematical operations, an unfrozen warmed-up supernet sharing weights across different candidate models is allowed, i.e., each model can inherit weights from this activated warmed-up supernet for a quick launch and is then trained with weight sharing in a multi-task manner to enhance transferability.

Extensive experimental results show that our searched architecture, named EfficientBERT, is 6.9× smaller and 4.4× faster than BERT$_{\mathrm{BASE}}$, and has competitive performance. On the test set of GLUE benchmark, EfficientBERT attains an average score of 77.7, which is 0.7 higher than MobileBERT$_{\mathrm{TINY}}$, and achieves an F1 score of 85.3/74.5 on the SQuAD v1.1/v2.0 dev dataset, which is 3.2/2.7 higher than TinyBERT$_4$ even without data augmentation.

2 Related Work

Compression for Pre-trained Language Models. For the past few years, pre-trained language models (PLMs) have demonstrated their strong powers on a variety of NLP tasks with the trend of larger and larger model size as well as better results. However, it is hard to deploy them on resource-limited edge devices for practical usage. To solve this problem, many efficient PLMs have been proposed (Turc et al., 2019; Lan et al., 2020). For example, Turc et al. (2019) directly pre-train and fine-tune smaller BERT models. In addition, many compression techniques for PLMs have been proposed recently to reduce the training cost, including quantization, weight pruning, and knowledge distillation (KD) (Shen et al., 2020; Sajjad et al., 2020; Jiao et al., 2020). Among them, KD (Hinton et al., 2015) is widely used due to its plug-and-play feasibility, which aims to transfer the knowledge from larger teacher models to smaller student models without sacrificing too much performance. For example, BERT-PKD (Sun et al., 2019) jointly distills the intermediate and last layers during fine-tuning. DistilBERT (Sanh et al., 2020) distills the last layers with a triple loss during pre-training. MobileBERT (Sun et al., 2020) designs an inverted-bottleneck model structure and progressively transfers the knowledge during pre-training. MiniLM (Wang et al., 2020b) performs a deep self-attention distillation during pre-training. TinyBERT (Jiao et al., 2020) introduces a comprehensive Transformer distillation method during pre-training and fine-tuning.

Nevertheless, the compression of the feed-forward network (FFN) has not been well studied, although its computational cost is 2∼3 larger than the multi-head attention (MHA) as pointed out by

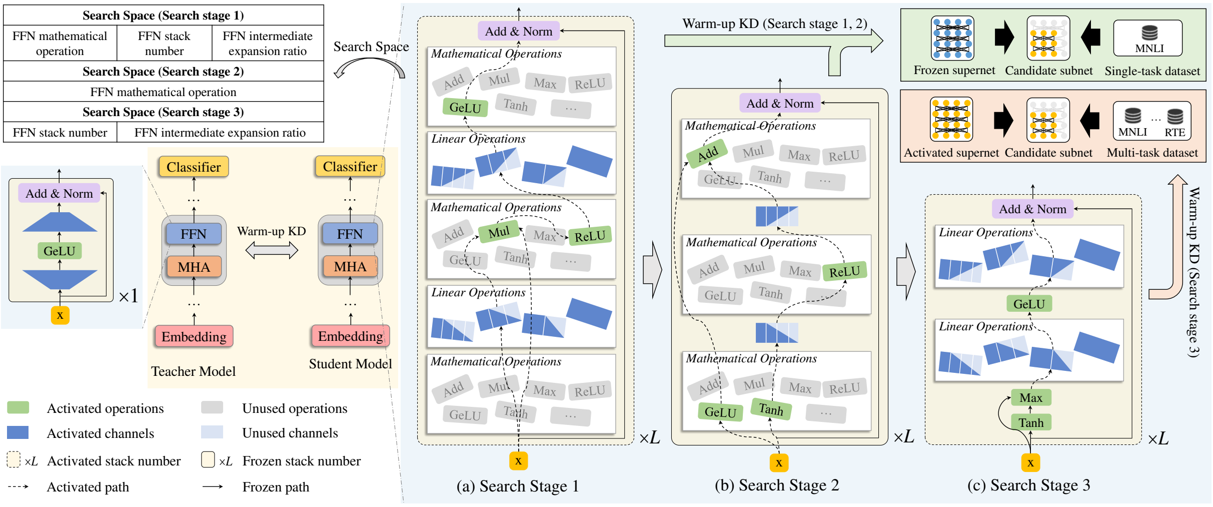


Figure 2: An overview of the search procedure of our EfficientBERT. The teacher model is $\text{BERT}_{\text{BASE}}$ (left), and the search space of our student model is designed towards achieving better nonlinearity of FFN, which contains mathematical operations, stack numbers, and intermediate expansion ratios (right). During searching, we progressively shrink the search space and divide the search process into three stages to conduct NAS ((a)-(c)). Both in the search and retraining stages, we use a novel warm-up knowledge distillation to transfer the teacher model's knowledge (middle). Specifically, each candidate or retrained subnet first inherits the weights from a frozen or activated warmed-up supernet, then conducts pre-training and fine-tuning with single-task or multi-task dataset.

Iandola et al. (2020). In contrast, compressing FFN is our main focus in this work.

Neural Architecture Search. Motivated by the success of neural architecture search (NAS) in computer vision (Howard et al., 2019; Li et al., 2020; Cai et al., 2020), increasing attention has been paid to applying NAS to NLP tasks (So et al., 2019; Wang et al., 2020a; Chen et al., 2020), aiming to automatically search for optimal architectures from a vast search space. Evolved Transformer (So et al., 2019) employs NAS to search for a better Transformer architecture with an evolutionary algorithm. HAT (Wang et al., 2020a) applies NAS to search for efficient hardware-aware Transformer models based on a Transformer supernet. AdaBERT (Chen et al., 2020) searches for task-adaptive small models with KD and differentiable NAS method. NAS-BERT (Xu et al., 2021) proposes a task-agnostic NAS method for adaptive-size model compression, where several acceleration techniques (including block-wise search, search space pruning, and performance approximation) are introduced to speed up the searching process.

Differently, in this paper, we design a comprehensive search space towards the nonlinearity of multilayer perceptron (MLP) in FFN, and propose a novel coarse-to-fine NAS approach with warm-up KD to find the optimal MLP architectures. Unlike AdaBERT, we apply NAS to search for a small universal BERT with competitive performance and strong transferability. And unlike NAS-BERT, we design a much more flexible search space and use warm-up KD with a coarse-to-fine searching paradigm to accelerate searching and enhance model transferability.

Table 1: Mathematical operations of FFN. Following the original papers of GeLU (Hendrycks and Gimpel, 2016) and Leaky ReLU (He et al., 2015), we let $c_1 = 0.5, c_2 = \sqrt{2/\pi}, c_3 = 0.044715, c_4 = 0.01$.

Operation	**Expression**	**Arity**
Add	x + y	2
Mul	x × y	2
Max	$\max(x, y)$	2
GeLU	$c_1 x(1 + \tanh(c_2(x + c_3 x^3)))$	1
Sigmoid	$1/(1 + e^{-x})$	1
Tanh	$(e^x - e^{-x})/(e^x + e^{-x})$	1
ReLU	$\max(x, 0)$	1
Leaky ReLU	x if $x \geq 0$ else $c_4 x$	1
ELU	x if $x \geq 0$ else $e^x - 1$	1
Swish	$x/(1 + e^{-x})$	1

3 Our EfficientBERT

We aim at discovering a lightweight MLP architecture with better nonlinearity in each FFN layer, ensuring the searched model can achieve compelling performance. We first present the search space towards better nonlinearity of MLP in FFN, as described in Section 3.1. Then we propose a novel coarse-to-fine NAS method with warm-up KD as discussed in Section 3.2.

3.1 Search Space Design

In a standard Transformer layer, there are two main components: a multi-head attention (MHA) and a feed-forward network (FFN). Theoretically, the computation (Mult-Adds) of MHA and FFN is $O(4Ld^2 + L^2d)$ and $O(2 \times 4Ld^2)$ respectively, where L is the sequence length and d is the channel number. As d gets larger, the computation of FFN gets larger than MHA. And as pointed out by previous works (Iandola et al., 2020), the latency of MHA and FFN in each layer of $\text{BERT}_{\text{BASE}}$ accounts for about 30% and 70% on a Google Pixel 3 smartphone, and the parameter numbers for MHA and FFN are about 2.4M and 4.7M, respectively. These demonstrate the potential of compressing FFN, i.e., compressing FFN may be more promising than squeezing MHA. In addition, as discussed by Dong et al. (2021), the MLP in FFN can prevent an optimization problem, i.e., rank collapse, caused by self-attention; thus, the nonlinearity ability of FFN deserves to be investigated. Hence, our main focus is compression and improvement of FFN.

We then design a search space towards the nonlinearity of MLP in FFN to search for a model with better nonlinearity of FFN and increase the performance. Many factors determine the FFN nonlinearity, such as the mathematical operations and the expansion ratios of intermediate hidden size. Inspired by MobileBERT (Sun et al., 2020), we find that by increasing the stack number of FFN, the model performance can also be remarkably improved. We integrate all of the above factors into our search space, including the mathematical operations, stack numbers, and intermediate expansion ratios of FFN. *(1) Mathematical operation:* We define some primitive operations (including several binary aggregation functions and unary activation functions) and search their different combinations, as shown in Table 1. *(2) Stack number:* The stack number of FFN is selected from $\{1, 2, 3, 4\}$. *(3) Intermediate expansion ratio:* The intermediate expansion ratio is selected from $\{1, 1/2, 1/3, 1/4\}$. Note that the stack number and the intermediate expansion ratio are jointly considered to balance the computation cost, e.g., network parameters. We use a directed acyclic graph (DAG) to represent each FFN architecture when searching the mathematical operations. The mathematical operations and linear operations are optionally placed in the intermediate nodes to process the hidden states. More details of our search space can be found in Figure 2.

3.2 Neural Architecture Search

Base Model Design. As discussed by previous BERT compression works (Jiao et al., 2020; Sun et al., 2020), there are several strategies to reduce the model size, including the embedding factorization and model width/depth reduction. However, most of the recent works only consider part of these strategies. In our work, we design a base model with all these strategies to make a comprehensive compression. Besides, we find that the expansion ratio of intermediate hidden size in FFN contributes a lot to the model size and inference latency. Thus the reduction of the intermediate expansion ratio is also considered. The detailed settings of our base model can be found in Section 4.2.

Coarse-to-Fine NAS with Warm-up KD. To speed up the search in the vast search space, we propose a coarse-to-fine NAS method by progressively shrinking the search space. The search process is divided into three stages where a coarse-grained search is conducted in the first stage to jointly search all of the factors in our search space, and fine-grained searches are conducted in the last two stages to search for partial factors.

In the first search stage, we jointly search all of the factors in our search space, including the mathematical operations, stack numbers, and intermediate expansion ratios. We use a DAG computation graph described in §3.1 to represent each MLP architecture. The initial search candidates are based on our base model, but different stack numbers and intermediate expansion ratios of FFN are allowed.

During searching, each candidate model is first sampled by a learnable sampling decision tree as proposed in LaNAS (Wang et al., 2019). Then warm-up KD is employed on each candidate model to accelerate the search process. Since we need to search for the mathematical operations, we cannot share the weights of different candidate models to avoid the potential interference problems. Instead, we first build a warmed-up supernet based on our base model with the maximum FFN stack number and intermediate expansion ratio in our search space. The supernet is pre-trained entirely (i.e., complete graph) with KD. The weights of the supernet are then frozen. When training each candidate model, we first inherit its weights from the supernet. Precisely, the weights of each stacked FFN are sliced from bottom to top layer; and the weights of each linear operation are sliced from left to right channel. After that, we only need to pre-train and fine-tune each model for a few steps via KD to adjust the inherited weights. This significantly reduces the search cost.

In the second search stage, to discover more diversified mathematical operations and evaluate their effects, we search them individually with the same method in the first search stage. The initial search candidates are built upon the searched model of the first search stage (i.e., we fix the stack numbers and expansion ratios). The sampling and KD strategies are the same as the first search stage.

In the third search stage, we jointly search the stack numbers and intermediate expansion ratios in the search space to explore their potentials further. The initial search candidates are based on the second search stage's searched model; the searched mathematical operations are fixed, but different stack numbers and intermediate expansion ratios of FFN are allowed. We also apply warm-up KD to accelerate the searching. Specifically, we first warm up the supernet entirely (i.e., complete graph) via KD again but do not freeze its weights. Then we share the weights of different candidate models (i.e., subgraphs of the supernet) during pre-training and fine-tuning to make acceleration. Each candidate model is sampled uniformly. Compared with the first two search stages, the search cost is dramatically reduced, enabling us to leverage more downstream datasets to enhance the model transferability. Inspired by MT-DNN (Liu et al., 2019), each candidate model is fine-tuned in a multi-task manner on different categories of downstream tasks. The weights of the embedding and Transformer layers for all tasks are shared, while those of the prediction layers are different.

Warm-up KD Formulations. In our warm-up KD, each candidate/retrained model initially inherits the weights from a warmed-up supernet. We use BERT$_{\text{BASE}}$ (Devlin et al., 2019) as the teacher model. Following TinyBERT (Jiao et al., 2020), we jointly distill the attention matrices, Transformer-layer outputs, embeddings, and predicted logits between the student and teacher models. In detail, the attention loss at the m-th student layer $\mathcal{L}^m_{attn}$ is calculated by the mean square error (MSE) loss as:

$$\mathcal{L}^m_{attn} = \frac{1}{h}\sum_{i=1}^{h} \text{MSE}(\mathbf{A}^S_{i,m}, \mathbf{A}^T_{i,n}), \qquad (1)$$

where $\mathbf{A}^S_{i,m}$ and $\mathbf{A}^T_{i,n}$ refer to the i-th head of attention matrices at m-th student layer and its matching n-th teacher layer, respectively, and h is the number of attention heads. The Transformer-layer output loss at the m-th student layer $\mathcal{L}^m_{hidn}$ and the embedding loss $\mathcal{L}_{embd}$ can be formulated as:

$$\begin{cases} \mathcal{L}^m_{hidn} = \text{MSE}(\mathbf{H}^S_m \mathbf{W}_h, \mathbf{H}^T_n) \\ \mathcal{L}_{embd} = \text{MSE}(\mathbf{E}^S \mathbf{W}_e, \mathbf{E}^T) \end{cases}, \qquad (2)$$

where $\mathbf{H}^S_m$ and $\mathbf{H}^T_n$ are the Transformer-layer outputs at m-th student layer and its matching n-th teacher layer, respectively. $\mathbf{E}$ is the embedding, and two learnable transformation matrices $\mathbf{W}_h$ and $\mathbf{W}_e$ are applied to align the mismatch dimensions between the student and teacher models. Moreover, the prediction loss $\mathcal{L}_{pred}$ calculated by the soft cross-entropy (CE) loss can be formulated as:

$$\mathcal{L}_{pred} = \text{CE}(\mathbf{z}^S/t, \mathbf{z}^T/t), \qquad (3)$$

where $\mathbf{z}$ is the predicted logits vector, and t is the temperature value. Finally, we combine all of the above losses and derive the overall KD loss as:

$$\mathcal{L} = \sum_{m=1}^{M} (\mathcal{L}^m_{attn} + \mathcal{L}^m_{hidn}) + \mathcal{L}_{embd} + \gamma \mathcal{L}_{pred}, \quad (4)$$

where M is the number of Transformer layers in the student model, γ controls the weight of the prediction loss $\mathcal{L}_{pred}$.

4 Experiment

This section demonstrates the superior performance and transferability of our EfficientBERT on a wide range of downstream tasks.

4.1 Datasets

We evaluate our model on two standard benchmarks for natural language understanding, i.e., the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018) and the Stanford Question Answering Dataset (SQuAD). The GLUE benchmark contains nine classification datasets, including MNLI (Williams et al., 2018), QQP (Chen et al., 2018), QNLI (Rajpurkar et al., 2016), SST-2 (Socher et al., 2013), CoLA

Table 2: Results on the test set of GLUE benchmark. The architectures of different models are as follows. $\text{BERT}_{\text{TINY}}$ & TinyBERT_4: (M=4, d=312, d_i=1200); $\text{BERT}_{\text{SMALL}}$: ($M$=4, d=512, d_i=2048); BERT-PKD_4 & DistilBERT_4: (M=4, d=768, d_i=3072); BERT-PKD_6 & DistilBERT_6: (M=6, d=768, d_i=3072). The latency is the average inference time over 100 runs on a single GPU with a batch size of 128.

Model	**#Params**	**Latency**	**MNLI-m/mm**	**QQP**	**QNLI**	**SST-2**	**CoLA**	**STS-B**	**MRPC**	**RTE**	**Avg**
$\text{BERT}_{\text{BASE}}$ (Google)	108.9M	362ms	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{BASE}}$ (Teacher)	108.9M	362ms	84.8/83.8	71.6	91.3	93.1	53.9	85.3	89.2	68.9	80.2
$\text{BERT}_{\text{TINY}}$ (Turc et al., 2019)	14.5M	43ms	75.4/74.9	66.5	84.8	87.6	19.5	77.1	83.2	62.6	70.2
$\text{BERT}_{\text{SMALL}}$ (Turc et al., 2019)	28.8M	75ms	77.6/77.0	68.1	86.4	89.7	27.8	77.0	83.4	61.8	72.1
BERT-PKD_4 (Sun et al., 2019)	52.8M	129ms	79.9/79.3	70.2	85.1	89.4	24.8	79.8	82.6	62.3	72.6
BERT-PKD_6 (Sun et al., 2019)	67.0M	193ms	81.5/81.0	70.7	89.0	92.0	43.5	81.6	85.0	65.5	76.6
DistilBERT_4 (Sanh et al., 2020)	52.8M	129ms	78.9/78.0	68.5	85.2	91.4	32.8	76.1	82.4	54.1	71.9
DistilBERT_6 (Sanh et al., 2020)	67.0M	193ms	82.6/81.3	70.1	88.9	**92.5**	**49.0**	81.3	86.9	58.4	76.8
TinyBERT_4 (Jiao et al., 2020)	14.5M	43ms	81.8/80.7	69.6	87.7	91.2	27.2	83.0	88.5	64.9	75.0
$\text{MobileBERT}_{\text{TINY}}$ (Sun et al., 2020)	15.1M	96ms	81.5/81.6	68.9	89.5	91.7	46.7	80.1	87.9	65.1	77.0
$\text{EfficientBERT}_{\text{TINY}}$	9.4M	52ms	82.4/81.0	70.3	88.5	91.2	37.5	80.9	87.8	64.6	76.0
EfficientBERT w/o Warm-up KD	15.7M	83ms	83.1/82.0	71.0	89.5	90.8	42.1	82.1	88.4	67.2	77.4
EfficientBERT	15.7M	83ms	**83.3**/82.3	71.0	90.2	92.1	43.8	82.9	88.2	65.7	77.7
EfficientBERT+	15.7M	83ms	83.0/82.3	**71.2**	89.3	92.4	38.1	**85.1**	**89.9**	**69.4**	77.9
EfficientBERT++	16.0M	103ms	83.0/**82.5**	**71.2**	**90.6**	92.3	42.5	83.6	88.9	67.8	**78.0**

Table 3: Results on the dev set of GLUE benchmark compared with other NAS methods. † indicates the results with data augmentation.

Model	**#Params**	**MNLI-m**	**QQP**	**QNLI**	**SST-2**	**CoLA**	**STS-B**	**MRPC**	**RTE**	**Avg**
AdaBERT (Chen et al., 2020) †	6.4∼9.5M	81.3	70.5	87.2	**91.9**	-	-	84.7	64.1	-
NAS-BERT_{10} (Xu et al., 2021)	10M	76.4	88.5	86.3	88.6	34.0	84.8	79.1	66.6	75.5
NAS-BERT_{30} (Xu et al., 2021)	30M	81.0	**90.2**	88.4	90.5	48.7	**87.6**	84.6	**71.8**	80.3
$\text{EfficientBERT}_{\text{TINY}}$	9.4M	81.7	86.7	89.3	90.1	39.1	79.9	90.1	63.2	77.5
EfficientBERT	15.7M	**83.1**	87.3	**90.4**	91.3	**50.2**	82.5	**91.5**	66.8	**80.4**

(Warstadt et al., 2019), STS-B (Cer et al., 2017), MRPC (Dolan and Brockett, 2005), RTE (Bentivogli et al., 2009), and WNLI (Levesque et al., 2011). The SQuAD task aims to predict the answer text span of the given question in a Wikipedia passage, which contains two datasets: SQuAD v1.1 (Rajpurkar et al., 2016) and SQuAD v2.0 (Rajpurkar et al., 2018). The metrics can be found in Wang et al. (2018) and Rajpurkar et al. (2016).

4.2 Model Settings

The embedding factorization strategy of our base model is the same as MobileBERT (Sun et al., 2020), the number of Transformer layers M is set to 6, the hidden size of the model d is set to 540, and the intermediate expansion ratio of FFN is set to 1 with intermediate hidden size d_i of 540. The remaining structures are the same as $\text{BERT}_{\text{BASE}}$.

We retrain our searched model of the third search stage by employing the warm-up KD method used in the first two search stages described in Section 3.2, referring to as EfficientBERT. EfficientBERT+ is obtained by inheriting the weights of EfficientBERT from the multi-task fine-tuned supernet and then directly fine-tune on each downstream task. Moreover, to verify the importance of model depth, we extend our EfficientBERT from 6 layers to 12 layers by affinely repeating each layer in EfficientBERT twice and shrink the hidden size from 540 to 360, forming EfficientBERT++. The weights are initially inherited from the warmed-up supernet of EfficientBERT in the same manner. In addition, to ensure a fair comparison with TinyBERT_4, we further shrink the hidden size of our EfficientBERT from 540 to 360, forming $\text{EfficientBERT}_{\text{TINY}}$, which has similar latency with TinyBERT_4.[1]

Table 4: Results on the SQuAD dev datasets. The architectures of MiniLM_4 and MiniLM_6 are (M=4, d=384, d_i=1536) and (M=6, d=384, d_i=1536), respectively. † indicates the results with data augmentation.

Model	**#Params**	**SQuAD v1.1** **EM/F1**	**SQuAD v2.0** **EM/F1**
$\text{BERT}_{\text{BASE}}$ (Google)	108.9M	80.8/88.5	-/-
$\text{BERT}_{\text{BASE}}$ (Teacher)	108.9M	80.5/88.2	74.8/77.7
BERT-PKD_4 (Sun et al., 2019)	52.8M	70.1/79.5	60.8/64.6
BERT-PKD_6 (Sun et al., 2019)	67.0M	77.1/85.3	66.3/69.8
DistilBERT_4 (Sanh et al., 2020)	52.8M	71.8/81.2	60.6/64.1
DistilBERT_6 (Sanh et al., 2020)	67.0M	78.1/86.2	66.0/69.5
TinyBERT_4 (Jiao et al., 2020) †	14.5M	72.7/82.1	68.2/71.8
MiniLM_4 (Wang et al., 2020b)	19.3M	-/-	-/69.7
MiniLM_6 (Wang et al., 2020b)	22.9M	-/-	-/72.7
$\text{EfficientBERT}_{\text{TINY}}$	9.4M	74.8/83.6	68.6/71.9
EfficientBERT	15.7M	77.0/85.3	71.4/74.5
EfficientBERT++	16.0M	**78.3/86.5**	**73.0/76.1**

[1] Our searched model, i.e., EfficientBERT, can be seen in Figure 6 of the Appendix A.

Table 5: Results of searched models at different search stages on the GLUE test set. Wiki and Books refer to the pre-training corpora of English Wikipedia and BooksCorpus, respectively.

Model (Pre-train Dataset)	#Params	MNLI-m/mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg
Base Model (Wiki)	15.3M	82.5/81.6	71.0	89.0	91.4	37.3	82.1	86.1	65.8	76.3
Search Stage 1 (Wiki)	15.4M	82.8/82.0	71.0	89.7	91.8	37.4	82.2	87.7	65.3	76.7
Search Stage 2 (Wiki)	15.4M	82.8/82.3	70.9	89.8	**92.2**	38.3	82.1	**88.5**	65.7	77.0
Search Stage 2 (Wiki+Books)	15.4M	82.8/82.0	**71.1**	89.7	92.1	42.5	82.2	88.2	**66.3**	77.4
EfficientBERT (Wiki+Books)	15.7M	**83.3/82.3**	71.0	**90.2**	92.1	**43.8**	**82.9**	88.2	65.7	**77.7**

Table 6: Effectiveness comparison between single-stage searching and our coarse-to-fine NAS method.

Method	Best Score	#Searched Arch	Search Cost
Single stage	76.7	2,700	84 GPU days
Search stage 1	76.7	1,900	54 GPU days
Search stage 1, 2	77.0	2,000	56 GPU days
Coarse-to-Fine NAS	77.7	5,000	58 GPU days

4.3 Implementation Details

In the first two search stages, the frozen supernet sliced by each candidate model is pre-trained for ten epochs, and we use 2% of the English Wikipedia corpus to pre-train each candidate model for one epoch. During fine-tuning, we use the first 10% training set of MNLI to train each model for three epochs and the last 1% training set for evaluation. In the third search stage, the activated supernet is pre-trained and fine-tuned for ten epochs, and each candidate model is optimized for one step. We use the entire corpora of English Wikipedia and BooksCorpus as the pre-training data, the combination of 90% training set of each downstream GLUE task as the fine-tuning data, and the rest 10% training set of MNLI as the evaluation data. The batch size at each search stage is set to 256. The learning rates for pre-training and fine-tuning at each stage are set to 1e-4 and 4e-4, respectively.

During retraining, each searched model is first pre-trained for ten epochs based on the inherited weights from the warmed-up supernet and is then fine-tuned on downstream tasks for ten epochs except for CoLA. Note that CoLA is fine-tuned for 50 epochs following the widely-used protocol. The batch sizes for pre-training and fine-tuning are set to 256 and 32, respectively. The learning rate for pre-training is set to 1e-4. The learning rates for fine-tuning on GLUE and SQuAD datasets are set to 5e-5 and 1e-4, respectively.

In all of our experiments, γ is set to 0 and 1 for pre-training and fine-tuning, respectively. t is set to 1. The maximum sequence length is set to 128. We use Adam with $\beta_1 = 0.9$, $\beta_2 = 0.999$, L2 weight decay of 0.01, warm-up proportion of 0.1, and linear decay of the learning rate.

4.4 Results on GLUE

We compare our searched models with BERT$_{\mathrm{TINY}}$, BERT$_{\mathrm{SMALL}}$ (Turc et al., 2019) and several state-of-the-art compressed BERT models, including BERT-PKD (Sun et al., 2019), DistilBERT (Sanh et al., 2020), TinyBERT$_4$ (Jiao et al., 2020), and MobileBERT$_{\mathrm{TINY}}$ (Sun et al., 2020). For a fair comparison, TinyBERT$_4$ is re-implemented by removing the data augmentation and fine-tuning from the official general distillation weights[2]. The experimental results on the test set of GLUE benchmark are listed in Table 2 and Figure 1.

From the results in Table 2, we can observe that: (1) Our EfficientBERT is 6.9× smaller and 4.4× faster than BERT$_{\mathrm{BASE}}$ and has achieved a competitive average GLUE score of 77.7, which is 0.7 higher than its counterpart MobileBERT$_{\mathrm{TINY}}$. (2) Our EfficientBERT+ has better transferability than EfficientBERT across different GLUE tasks with an improvement of 0.2 on the average score, demonstrating the effectiveness of our multi-task training strategy in the third search stage. (3) Our EfficientBERT++ has achieved state-of-the-art performance, which outperforms MobileBERT$_{\mathrm{TINY}}$ by 1.0 on the average score. (4) Our EfficientBERT$_{\mathrm{TINY}}$ outperforms TinyBERT$_4$ by a 1.0 average score with fewer parameters and similar latency. (5) Without our warm-up KD during retraining, i.e., pre-training the model from scratch rather than from the warmed-up supernet, the average score of EfficientBERT decreases by 0.3, demonstrating the advantage of retraining with our warm-up KD. And from the results in Figure 1, we can see that all of our searched models outperform other compared models with similar or lower latency.

Furthermore, to verify the effectiveness of our proposed NAS method, we compare with several related NAS methods on the GLUE dev set, includ-

[2] We use the 2nd version from https://github.com/huawei-noah/Pretrained-Language-Model/tree/master/TinyBERT

Table 7: Results of our EfficientBERT with different base models on the GLUE test set.

Model (Base Model)	#Params	MNLI-m/mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg
TinyBERT$_6$	67.0M	83.8/83.2	71.4	89.8	92.0	38.8	83.1	89.0	65.8	77.4
EfficientBERT (TinyBERT$_6$)	70.1M	**84.1/83.2**	**71.4**	**90.4**	**92.6**	**46.2**	**83.7**	**89.0**	**67.7**	**78.7**

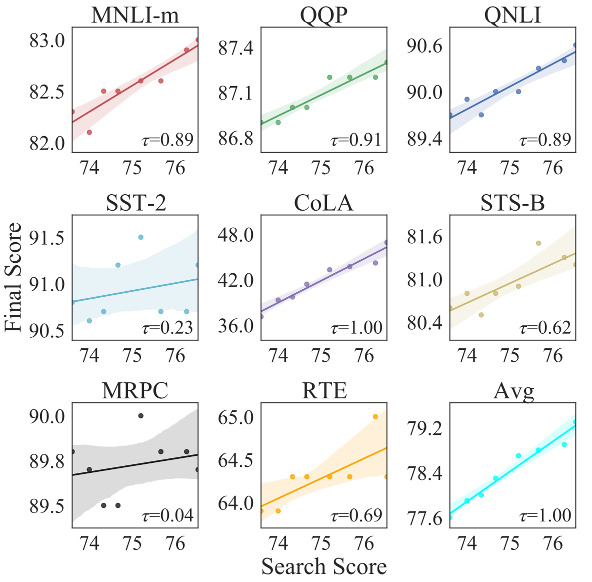


Figure 3: Results of model ranking correlation between the search and retraining phases on the GLUE dev set in the first search stage.

ing AdaBERT (Chen et al., 2020) and NAS-BERT (Xu et al., 2021). The results are shown in Table 3. As can be seen, with similar parameters, our EfficientBERT$_{\text{TINY}}$ has better performance than AdaBERT and NAS-BERT$_{10}$; and our EfficientBERT outperforms NAS-BERT$_{30}$ even with much fewer parameters. These results demonstrate the superiority of our NAS method.

4.5 Results on SQuAD

To measure the transferability of our searched models across different types of tasks, we further evaluate our models on SQuAD dev datasets, as shown in Table 4. We choose BERT-PKD, DistilBERT, TinyBERT$_4$, and MiniLM (Wang et al., 2020b) as the baseline models. From the results, we can see that our EfficientBERT still achieves competitive performances, which outperforms TinyBERT$_4$ by 3.2/2.7 F1 score on SQuAD v1.1/v2.0 dev dataset even without data augmentation, and surpasses MiniLM$_6$ by 1.8 F1 score on SQuAD v2.0 dev dataset. Besides, our EfficientBERT$_{\text{TINY}}$ can also outperform TinyBERT$_4$ on both SQuAD dev datasets. These results indicate the strong performance and transferability of our searched models.

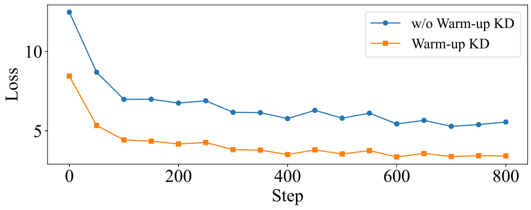


Figure 4: Efficiency comparison results between searching with and without our warm-up KD.

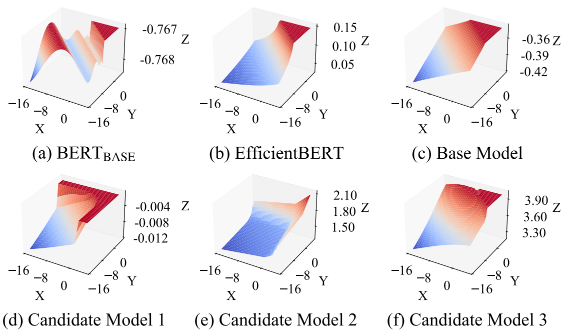
4.6 Discussion

Effectiveness of Coarse-to-Fine NAS Method. To measure the effectiveness of our coarse-to-fine NAS method, we first compare the performances of the searched models at different search stages on the GLUE test set in Table 5. It can be observed that the searched model in the first search stage has better performance than our base model, which proves the effectiveness of the coarse-grained NAS process. And from the first to the third search stages, the performances of the searched models are gradually enhanced, which shows the effectiveness of the fine-grained strategies and the necessity of each factor in our search space.

Then we compare the effectiveness between single-stage searching and our coarse-to-fine NAS method in Table 6. As shown, our coarse-to-fine NAS method has higher efficiency than single-stage searching, saving 26 GPU days. It can also search for 2,300 more architectures and observe better architecture with a higher GLUE test score.

Effectiveness of Warm-up KD. To evaluate the model ranking effectiveness of our warm-up KD method between the search and retraining phases, we first randomly sample eight candidate models in the first search stage, whose search scores range from 77.6 to 79.3. Then we retrain each model and obtain its final score on the GLUE dev set, as shown in Figure 3. The Kendall Tau τ (Kendall, 1938) for each downstream task is also calculated. From the results, we can see that the search and retraining phases have strong positive correlations on most downstream tasks, demonstrating the strong ranking capability of the warm-up KD strategy.

Next, to test the efficiency, we compare the fine-tuning losses of our base model in the first



(a) $\mathrm{BERT}_{\mathrm{BASE}}$ (b) EfficientBERT (c) Base Model

(d) Candidate Model 1 (e) Candidate Model 2 (f) Candidate Model 3

Figure 5: Visualization towards the FFN nonlinearity of (a) $\mathrm{BERT}_{\mathrm{BASE}}$, (b) our EfficientBERT, (c) our base model, and (d)-(f) randomly selected candidate models with worse performances in the first search stage.

search stage between searching with and without our warm-up KD strategy, as shown in Figure 4. From the results, we can observe that the loss with our warm-up KD can reach a lower value with much fewer steps.

Transferability across Different Base Models. To test the transferability of our EfficientBERT across different base models, we replace the architecture of $\mathrm{TinyBERT}_6$ with that of the EfficientBERT and evaluate it on the GLUE test set. For both models, we use English Wikipedia to pre-train for three epochs from scratch to be consistent with Jiao et al. (2020). Note that the intermediate expansion ratio in our search space is applied to the original intermediate hidden size of $\mathrm{TinyBERT}_6$ (i.e., 3072). The results are shown in Table 7. From the results, we can observe that our EfficientBERT with the base model of $\mathrm{TinyBERT}_6$ outperforms the original $\mathrm{TinyBERT}_6$ on most of the downstream tasks, and has gained an improvement of 1.3 on the average GLUE score, showing the strong transferability of our EfficientBERT.

Visualization of FFN Nonlinearity. In Figure 5, we typically visualize the FFN nonlinearity of $\mathrm{BERT}_{\mathrm{BASE}}$, our EfficientBERT, our base model, and three randomly selected candidate models with worse performances in the first search stage. The input embedding of each model has two dimensions serving as axes X and Y, whose values are uniformly selected from -15∼5 to approximate the distribution of the embedding in $\mathrm{BERT}_{\mathrm{BASE}}$. The average output of the last Transformer layer is regarded as the value of axis Z. Besides, we remove the MHA, replace the layer normalization with the simple average operation, and set the weights and bias in each linear operation to 1 and 0, respectively, in order to alleviate their impacts. From the results, we can observe that the curves of (a)-(c) are more fluent and have less sudden increase regions than (d)-(f); and from (a) to (c), the curve complexity gradually decreases. It reflects that $\mathrm{BERT}_{\mathrm{BASE}}$ (our teacher model) has the best FFN nonlinearity, and our EfficientBERT has better nonlinearity than the base model and the randomly selected candidate models. This verifies the superiority of our method in gaining better nonlinear mapping ability. More visualization of the nonlinearity can be seen in Figure 7-8 of the Appendix B.

5 Conclusion

In this paper, we focus on the compression and improvement of FFN and design a profound search space over the nonlinearity of MLP in FFN, aiming at searching for better MLP architectures to improve the model performance. Due to the enormous search space, we conduct NAS in a progressive manner and employ a novel warm-up KD strategy at each search stage to accelerate searching and enhance model transferability. Extensive experiments show that our searched architecture EfficientBERT is 6.9× smaller and 4.4× faster than $\mathrm{BERT}_{\mathrm{BASE}}$, and has competitive performance and strong generalization ability. In the future, we will leverage NAS to discover more dynamic PLMs *w.r.t* different hardwares and downstream tasks.

Acknowledgements

This work was supported in part by National Key R&D Program of China under Grant No. 2020AAA0109700, National Natural Science Foundation of China (NSFC) under Grant No.U19A2073 and No.61976233, Guangdong Province Basic and Applied Basic Research (Regional Joint Fund-Key) Grant No.2019B1515120039, Guangdong Outstanding Youth Fund (Grant No. 2021B1515020061), Shenzhen Fundamental Research Program (Project No. RCYX20200714114642083, No. JCYJ20190807154211365).

References

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo

Giampiccolo, and Bernardo Magnini. 2009. The fifth pascal recognizing textual entailment challenge. In *Text Analysis Conference*.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation*, pages 1–14.

Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. 2020. Adabert: Task-adaptive bert compression with differentiable neural architecture search. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 2463–2469.

Z. Chen, H. Zhang, X. Zhang, and L. Zhao. 2018. Quora question pairs. *Quora*.

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*, pages 4171–4186.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing*.

Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. 2021. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. *arXiv preprint arXiv:2103.03404*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *International Conference on Computer Vision*, pages 1026–1034.

Dan Hendrycks and Kevin Gimpel. 2016. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *arXiv preprint arXiv: 1606.08415*.

Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. 2019. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*.

Forrest Iandola, Albert Shaw, Ravi Krishna, and Kurt Keutzer. 2020. SqueezeBERT: What can computer vision teach NLP about efficient neural networks? In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pages 124–135.

Zihang Jiang, Weihao Yu, Daquan Zhou, Yunpeng Chen, Jiashi Feng, and Shuicheng Yan. 2020. Convbert: Improving bert with span-based dynamic convolution. *arXiv preprint arXiv:2008.02496*.

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.

Maurice G Kendall. 1938. A new measure of rank correlation. *Biometrika*, pages 81–93.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.

Hector J Levesque, Ernest Davis, and Leora Morgenstern. 2011. The winograd schema challenge. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*.

Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. 2020. Block-wisely supervised neural architecture search with knowledge distillation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1986–1995.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Multi-task deep neural networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4487–4496.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*, pages 14014–14024.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 784–789.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2020. Poor man's bert: Smaller and faster transformer models. *arXiv preprint arXiv:2004.03844*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

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*, pages 8815–8821.

David R. So, Quoc V. Le, and Chen Liang. 2019. The evolved transformer. In *Proceedings of the 36th International Conference on Machine Learning*, pages 5877–5886.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. 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.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for BERT model compression. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pages 4323–4332.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. 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: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355.

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. 2020a. HAT: Hardware-aware transformers for efficient natural language processing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7675–7688.

Linnan Wang, Saining Xie, Teng Li, Rodrigo Fonseca, and Yuandong Tian. 2019. Sample-efficient neural architecture search by learning action space. *arXiv preprint arXiv:1906.06832*.

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

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

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*, pages 1112–1122.

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

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems*, pages 5753–5763.

Appendix

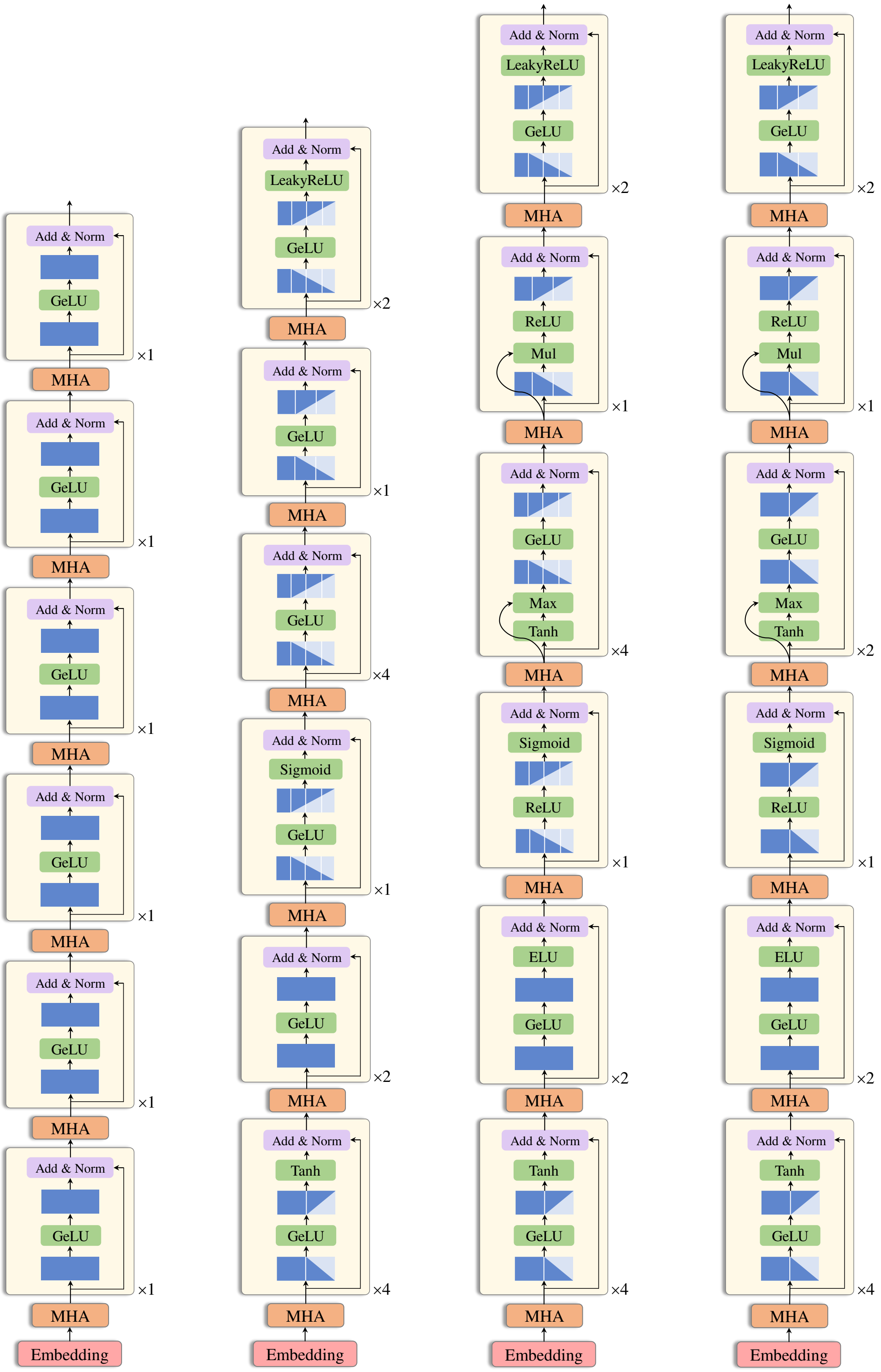
A Visualization of Searched Models

We visualize the architectures of our base model, the searched models of the first two search stages, and our EfficientBERT in Figure 6 from (a) to (d). From the architecture, we can observe that our EfficientBERT is more efficient since most of the searched intermediate expansion ratios are 1/2 while most of the searched stack numbers are less than 2. Besides, in our EfficientBERT, lower layers tend to have more FFN stack number or intermediate expansion ratio (e.g., layer 1, 2) so as to enrich the semantic representation to the maximum extent for processing by higher layers. In comparison, higher layers tend to learn more complex mathematical formulas (e.g., layers 4, 5) to enhance the nonlinearity of lower enriched representations. This could bring many inspirations for efficient and effective backbone design.

B Visualization of Model Nonlinearity

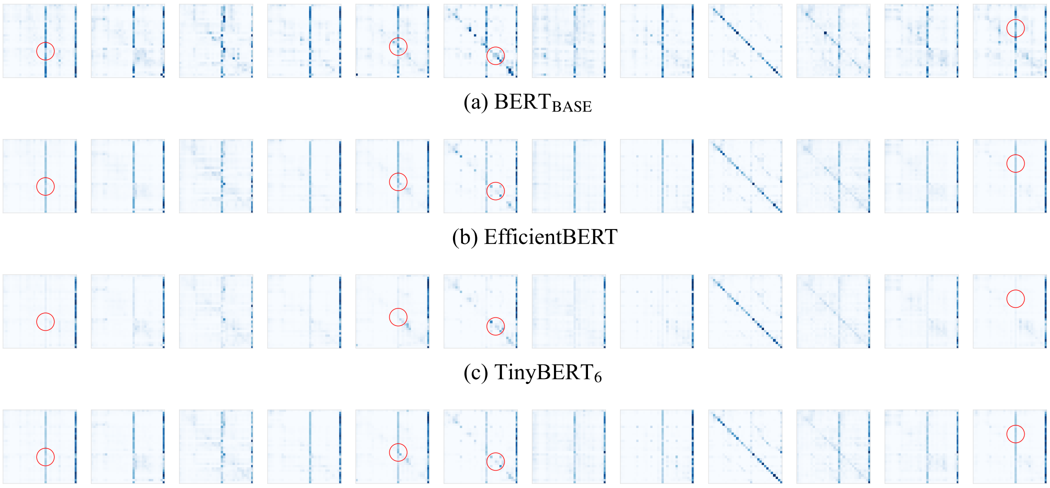
To further show the superior nonlinearity of our searched models, we visualize the attention maps in twelve attention heads for $\mathrm{BERT}_{\mathrm{BASE}}$, our EfficientBERT, $\mathrm{TinyBERT}_6$, and our EfficientBERT ($\mathrm{TinyBERT}_6$) in Figure 7, respectively. As can be seen, the feature maps of our EfficientBERT are close to those of $\mathrm{BERT}_{\mathrm{BASE}}$. This verifies the nonlinear mapping ability of our EfficientBERT in fitting the teacher model. Moreover, the attention distributions of our EfficientBERT ($\mathrm{TinyBERT}_6$) are closer to $\mathrm{BERT}_{\mathrm{BASE}}$ than $\mathrm{TinyBERT}_6$ in most of the attention heads. This proves the excellent nonlinear representation ability of our EfficientBERT ($\mathrm{TinyBERT}_6$) again.

Then, we visualize the feature maps of FFN outputs for the above four models, as shown in Figure 8. The observations in Figure 8 are similar to that of Figure 7, once again demonstrating the superior nonlinear representation ability of our EfficientBERT and EfficientBERT ($\mathrm{TinyBERT}_6$).



(a) Base Model (b) Search Stage 1 (c) Search Stage 2 (d) EfficientBERT

Figure 6: Architectures of (a) our base model, (b)-(c) the searched models of the first two search stages, and (d) our EfficientBERT.



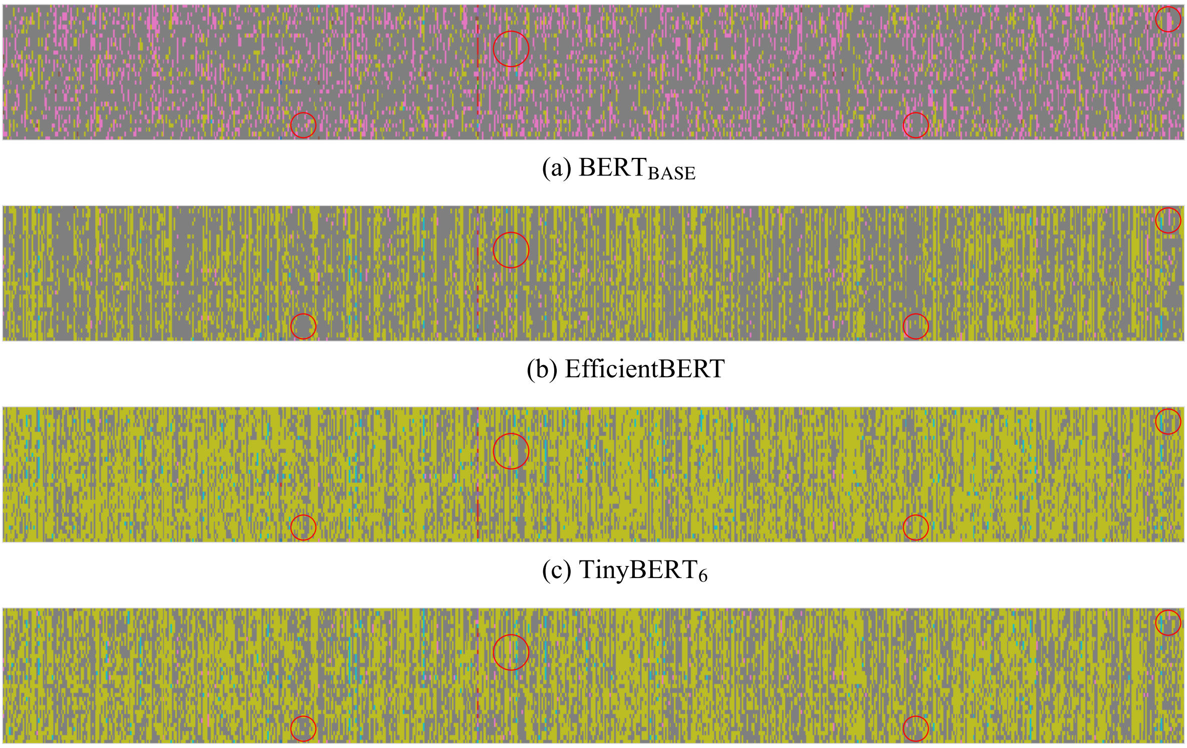
(a) $\mathrm{BERT}_{\mathrm{BASE}}$

(b) EfficientBERT

(c) $\mathrm{TinyBERT}_6$

(d) EfficientBERT ($\mathrm{TinyBERT}_6$)

Figure 7: Visualization for the attention distributions of (a) $\mathrm{BERT}_{\mathrm{BASE}}$, (b) our EfficientBERT, (c) $\mathrm{TinyBERT}_6$, and (d) our EfficientBERT ($\mathrm{TinyBERT}_6$) in the last Transformer layer.



(a) $\mathrm{BERT}_{\mathrm{BASE}}$

(b) EfficientBERT

(c) $\mathrm{TinyBERT}_6$

(d) EfficientBERT ($\mathrm{TinyBERT}_6$)

Figure 8: Visualization for the FFN output distributions of (a) $\mathrm{BERT}_{\mathrm{BASE}}$, (b) our EfficientBERT, (c) $\mathrm{TinyBERT}_6$, and (d) our EfficientBERT ($\mathrm{TinyBERT}_6$) in the last Transformer layer.