# Knowledge Distillation with Reptile Meta-Learning for Pretrained Language Model Compression

**Xinge Ma**[†], **Jin Wang**[†][*], **Liang-Chih Yu**[‡][*] and **Xuejie Zhang**[†]
[†]School of Information Science and Engineering, Yunnan University, Yunnan, P.R. China
[‡]Department of Information Management, Yuan Ze University, Taiwan
Contact:wangjin@ynu.edu.cn, lcyu@saturn.yzu.edu.tw

## Abstract

The billions, and sometimes even trillions, of parameters involved in pre-trained language models significantly hamper their deployment in resource-constrained devices and real-time applications. Knowledge distillation (KD) can transfer knowledge from the original model (i.e., teacher) into a compact model (i.e., student) to achieve model compression. However, previous KD methods have usually frozen the teacher and applied its immutable output feature maps as soft labels to guide the student's training. Moreover, the goal of the teacher is to achieve the best performance on downstream tasks rather than knowledge transfer. Such a fixed architecture may limit the teacher's teaching and student's learning abilities. Herein, a knowledge distillation method with reptile meta-learning is proposed to facilitate the transfer of knowledge from the teacher to the student. The teacher can continuously meta-learn the student's learning objective to adjust its parameters for maximizing the student's performance throughout the distillation process. In this way, the teacher *learns to teach*, produces more suitable soft labels, and transfers more appropriate knowledge to the student, resulting in improved performance. Unlike previous KD using meta-learning, the proposed method only needs to calculate the first-order derivatives to update the teacher, leading to lower computational cost but better convergence. Extensive experiments on the GLUE benchmark show the competitive performance achieved by the proposed method. For reproducibility, the code for this paper is available at: `https://github.com/maxinge8698/ReptileDistil`.

## 1 Introduction

In recent years, pre-trained language models (PLMs) have brought natural language processing to a new era and achieved state-of-the-art performance in a variety of tasks. Based on a multilayer Transformer architecture (Vaswani et al., 2017), PLMs, such as BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), and XLNet (Yang et al., 2019), typically contain hundreds of millions of parameters, making them computationally expensive and inefficient. Consequently, such large-scale models are difficult to be deployed on resource-constrained devices and real-time applications due to high computational complexity, huge storage requirements, and slow inference speed.

Knowledge distillation (KD) (Hinton et al., 2015) has been recommended to compress PLMs. Based on the teacher-student architecture applied, it teaches a smaller student model to reproduce the behavior of a larger teacher model. In practice, the teacher produces output feature maps as soft labels, providing the student with more information than the ground truth of one-hot labels to learn. By minimizing the Kullback-Leibler (KL) divergence between the softened probability distributions of the teacher and student, the student is then trained as an equally-effective model without significant sacrifice in performance.

However, previous studies (Sanh et al., 2019; Sun et al., 2019; Turc et al., 2019; Jiao et al., 2020; Wang et al., 2021) have usually frozen the teacher during the KD process and used its immutable knowledge to teach the student. Inevitably, this architecture has two major limitations: 1) *The parameters of the teacher were fixed for KD*. Concretely, the teacher is frozen throughout the distillation process, and the student can only passively receive the fixed knowledge conveyed by the teacher, which ignores the capacity gap and incompatibility between the teacher and student; 2) *The teacher is unaware of the existence of the student*. Specifically, the teacher model is usually fine-tuned to optimize its performance in downstream applications rather than to optimize the ability to distill the knowledge to the student.

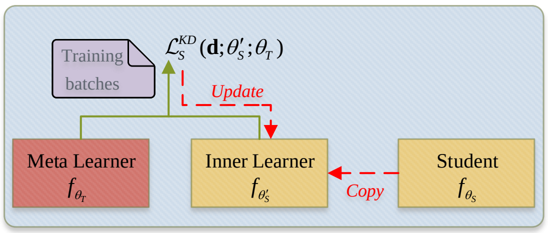
Recent studies (Park et al., 2021; Pham et al., 2021; Zhou et al., 2022) have suggested that up-

[*]Corresponding authors.

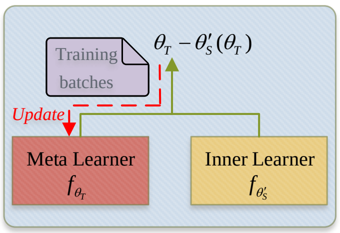
dating the teacher together with the student during the KD process, rather than freezing it, can make the student learn better. To be specific, Park et al. (2021) proposed a student-friendly teacher network for training the teacher and student branches to make the teacher aware of the student before distillation and obtain student-friendly knowledge, which is then transferred to the student via vanilla KD. Pham et al. (2021) presented meta pseudo labels where the student is trained based on the pseudo labels generated by the teacher and the teacher is trained based on the performance of the student on labeled data. Zhou et al. (2022) employed a meta-learning strategy to explicitly optimize the teacher with the optimization objective of the student's performance on a hold-out training subset during the KD process, allowing the teacher to evolve continuously to adapt to the current state of the student and better transfer knowledge to it.

The application of model-agnostic meta-learning (MAML) (Finn et al., 2017) in MetaDistil (Zhou et al., 2022) significantly improved the performance of the distilled student by calculating the second-order derivatives of the student's loss on the query set to obtain the update gradients of the teacher. However, limited by the MAML algorithm, there are some concerns in MetaDistil: 1) A query set needs to be designed. MetaDistil split the query set from the training set at a ratio of 9:1, as a result, it missed one-tenth of the training data since this separate query set is not directly used to train the student model. Nevertheless, this one-tenth of the data could lead to significant performance gain, especially when the training data is scarce; 2) The training process may be slow due to the nature of meta-learning. Furthermore, MAML involves the calculation of the second-order derivatives, so that more training time and computing resource are required.

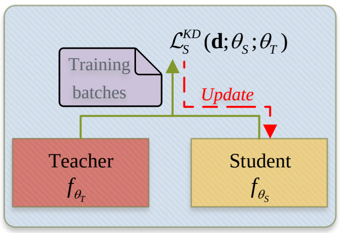
To address the above issues, a knowledge distillation method with reptile meta-learning (Nichol et al., 2018) is proposed to facilitate the transfer of knowledge from the teacher to the student, termed ReptileDistil, where the update gradients of the teacher are approximated by calculating the difference between the parameters of the meta-learner (i.e., teacher) and inner-learner (i.e., student), thereby reducing the computational burden by avoiding calculating the second-order derivatives, and mitigating the tedious operation of partitioning the query set and the performance loss that


Training batches
$\mathcal{L}_S^{KD}(\mathbf{d};\theta_S';\theta_T)$
Update
Meta Learner $f_{\theta_T}$
Inner Learner $f_{\theta_S'}$
Copy
Student $f_{\theta_S}$

(a) Inner update


Training batches
$\theta_T-\theta_S'(\theta_T)$
Update
Meta Learner $f_{\theta_T}$
Inner Learner $f_{\theta_S'}$

(b) Meta update



(c) Optimized knowledge distillation

Figure 1: An overview of the proposed ReptileDistil framework.

may be caused. To illustrate the effectiveness and practicality of the proposed ReptileDistil, extensive experiments are conducted on the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018). Empirical results demonstrate that the proposed method can yield meaningful improvements compared with other KD methods in distilling a 12-layer $\mathrm{BERT}_{\mathrm{BASE}}$ model with 110M parameters into a 6-layer $\mathrm{BERT}_6$ model with 66M parameters and 1.94 times speedup.

In summary, the contributions of this paper are as follows:

- A knowledge distillation method with reptile meta-learning is proposed for compressing PLMs, for which the reptile meta-learning is introduced to calculate the update gradients of the teacher model for optimizing it.

- Four different layer-wise parameter update strategies are proposed for updating the parameters of the teacher model, which avoid calculating the second-order derivatives and lead to better and faster convergence.

- Extensive experiments are conducted on the GLUE benchmark, and the results show that the proposed method performs better in distilling BERT than state-of-the-art KD methods.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 details the proposed ReptileDistil framework. Section 4 presents extensive experiments. Section 5 provides experimental results and analysis. Section 6 summarizes the paper.

## 2 Related Work

Model compression is a potential approach to reduce the model size and improve computational efficiency. Currently, the model compression techniques for PLMs can be divided into the following six categories: 1) model pruning (Michel et al., 2019; Fan et al., 2020; Gordon et al., 2020), which removes redundant or less important parameters; 2) weight quantization (Zafrir et al., 2019; Shen et al., 2020), which uses fewer bits to represent the parameters; 3) knowledge distillation (Sanh et al., 2019; Sun et al., 2019; Turc et al., 2019; Jiao et al., 2020; Wang et al., 2021; Zhou et al., 2022), which trains a smaller model that learns from the output feature maps of the original model; 4) module replacing (Xu et al., 2020), which replaces the modules of the original model with more compact substitutes; 5) matrix factorization (Lan et al., 2020), which reduces the parameters using methods such as cross-layer parameter sharing and factorized embedding parameterization; 6) early exit (Zhou et al., 2020; Liu et al., 2020; Xin et al., 2020), which allows the model to exit early at an off-ramp instead of passing through the entire model.

Here, we briefly review state-of-the-art work on investigating knowledge distillation to compress PLMs, especially the popular BERT model. DistilBERT (Sanh et al., 2019) is performed at the pre-training stage, which distills output logits from a pre-trained $\text{BERT}_{\text{BASE}}$ teacher into a 6-layer $\text{BERT}_6$ student initialized by taking one layer out of every two layers of a pre-trained $\text{BERT}_{\text{BASE}}$. BERT-PKD (Sun et al., 2019) is performed at the fine-tuning stage, which distills output logits and hidden states from a fine-tuned $\text{BERT}_{\text{BASE}}$ teacher into a 6-layer $\text{BERT}_6$ student initialized with the first 6 layers of a pre-trained $\text{BERT}_{\text{BASE}}$. PD (Turc et al., 2019) is performed at the pre-training stage, in which a randomly initialized 6-layer $\text{BERT}_6$ student is first trained with a masked language modeling objective and then is distilled with the output logits from a pre-trained $\text{BERT}_{\text{BASE}}$ teacher. TinyBERT (Jiao et al., 2020) first is performed at the pre-training stage, which distills embedding outputs, hidden states, and self-attention distributions from a pre-trained $\text{BERT}_{\text{BASE}}$ teacher into a 4-layer $\text{TinyBERT}_4$ student with a hidden size of 312 and an intermediate size of 1200, or a 6-layer $\text{TinyBERT}_6$ student with a hidden size of 768 and an intermediate size of 3072 as same as $\text{BERT}_6$. These two models are then treated as the initialization of student models for further distillation. After that, at the fine-tuning stage, the output logits, embedding outputs, hidden states, and self-attention distributions from a fine-tuned $\text{BERT}_{\text{BASE}}$ teacher are distilled into the aforementioned $\text{TinyBERT}_4$ student or $\text{TinyBERT}_6$ student on the augmented task-specific dataset. MiniLM v2 (Wang et al., 2021) is performed at the pre-training stage, which distills self-attention relation from the last Transformer layer of a pre-trained $\text{BERT}_{\text{BASE}}$ teacher into a randomly initialized 6-layer $\text{BERT}_6$ student. MetaDistil (Zhou et al., 2022) is performed at the fine-tuning stage, which distills output logits from a pre-trained $\text{BERT}_{\text{BASE}}$ teacher into a 6-layer $\text{BERT}_6$ student obtained from the aforementioned PD (Turc et al., 2019) via a meta-learning strategy.

## 3 Knowledge Distillation with Reptile Meta-Learning

Figure 1 shows an overview of the proposed ReptileDistil framework. Unlike prior work on improving KD, which usually designed new distillation loss, we aim to improve KD via reptile meta-learning to optimize the parameters of the teacher model adaptively with the distillation process of the student model. This allows the teacher model to adjust its pace to search for the optimal global solution, thus providing more suitable soft labels for the student by considering the current capacity of the latter. In addition, to update the parameters of the teacher, we propose four different layer-wise parameter update strategies to provide faster and better convergence.

### 3.1 Vanilla Knowledge Distillation

Knowledge distillation aims to transfer hidden knowledge from a larger teacher model $f_{\theta_T}$ to a shallow student model $f_{\theta_S}$ to improve the performance of the student model significantly, where $\theta_T$ and $\theta_S$ denote the trainable parameters of the teacher and student, respectively.

Formally, consider a labeled dataset $\mathcal{D} = \{\mathbf{x}^{(n)}, y^{(n)}\}_{n=1}^{N}$ containing $N$ training samples, where $\mathbf{x}^{(n)}$ and $y^{(n)}$ are the input feature and corresponding ground-truth label, respectively. The data is passed through the teacher model with $L$ layers, and the output hidden representation of each layer can be obtained by,

$$\mathbf{t}_1^{(n)}, ..., \mathbf{t}_l^{(n)}, ..., \mathbf{t}_L^{(n)} = f_{\theta_T}(\mathbf{x}^{(n)}; \theta_T), \quad (1)$$

where $\mathbf{t}_l^{(n)} \in \mathbb{R}^{d_\mathbf{h}}$ is the output representation of the $l$-th layer corresponding to the `[CLS]` token, and $d_\mathbf{h}$ is the dimensionality of the model. Particularly, the `[CLS]` output from the last layer is fed into a fully-connected layer with a *softmax* activation function, as follows,

$$\hat{\mathbf{y}}_T^{(n)} = \mathrm{softmax}(\frac{\tanh(W_T \mathbf{t}_L^{(n)} + b_T)}{\tau}), \quad (2)$$

where $W_T$ and $b_T$ are the weights and biases of the classifier, respectively, and $\hat{\mathbf{y}}_T^{(n)}$ denotes the predicted probability distribution of the $n$-th sample, i.e., the probabilities that the output belongs to the classes, which is usually softened by a temperature hyperparameter $\tau$ in the *softmax* activation function to control the degree of smoothness. With a lower temperature, the student focuses more on matching the maximal logits of the teacher outputs. On the contrary, a higher temperature encourages the student to focus on the logits other than the maximal ones. The training objective of the teacher model is a categorical cross-entropy loss between distributions of its predicted probability and the ground-truth label, which is defined as,

$$\mathcal{L}_T^{CE}(\mathcal{D}; \theta_T) = -\frac{1}{N}\sum_{n=1}^{N} \mathbb{I}(y^{(n)}) \circ \log \hat{\mathbf{y}}_T^{(n)}, \quad (3)$$

where $\mathbb{I}(y^{(n)})$ denotes the one-hot vector of the ground-truth label, and $\circ$ means the element-wise multiplication operation.

The student model is smaller with $K$ layers, where $K < L$. Similarly, the intermediate hidden representations of the student model are obtained by,

$$\mathbf{s}_1^{(n)}, ..., \mathbf{s}_k^{(n)}, ..., \mathbf{s}_K^{(n)} = f_{\theta_S}(\mathbf{x}^{(n)}; \theta_S), \quad (4)$$

where $\mathbf{s}_k^{(n)} \in \mathbb{R}^{d_\mathbf{h}}$ is the output representation of the $k$-th layer corresponding to the `[CLS]` token. Also, the predicted probability distribution of the student on the $n$-th sample is calculated by,

$$\hat{\mathbf{y}}_S^{(n)} = \mathrm{softmax}(\frac{\tanh(W_S \mathbf{s}_K^{(n)} + b_S)}{\tau}). \quad (5)$$

Same as the teacher, the student is trained on the task-specific objective to fit the training samples, as follows,

$$\mathcal{L}_S^{CE}(\mathcal{D}; \theta_S) = -\frac{1}{N}\sum_{n=1}^{N} \mathbb{I}(y^{(n)}) \circ \log \hat{\mathbf{y}}_S^{(n)}. \quad (6)$$

In addition, a knowledge distillation objective that aligns the behavior of the student and teacher is employed as additional supervision to train the student to learn the generalization ability of the teacher. Specifically, taking the soft output $\hat{\mathbf{y}}_T^{(n)}$ from Eq. (2) as the pseudo label, the student model can learn more from it than the ground-truth label by,

$$\mathcal{L}^{KL}(\mathcal{D}; \theta_S; \theta_T) = \tau^2 \frac{1}{N}\sum_{n=1}^{N} \mathrm{KL}(\hat{\mathbf{y}}_T^{(n)} || \hat{\mathbf{y}}_S^{(n)}), \quad (7)$$

where $\mathrm{KL}(\cdot||\cdot)$ means computing the Kullback-Leibler divergence between two distributions, which can force the student model to replicate the behavior of the teacher model by shrinking it. Furthermore, following Hinton et al. (2015), we scale the loss by multiplying $\tau^2$ to ensure that gradient magnitudes are approximately constant when the temperature changes. Therefore, the final training objective for KD is a weighted sum over the task-specific cross-entropy and KL-divergence, i.e.,

$$\mathcal{L}_S^{KD}(\mathcal{D}; \theta_S; \theta_T) = (1-\alpha)\mathcal{L}_S^{CE} + \alpha \mathcal{L}^{KL}, \quad (8)$$

where $\alpha \in [0, 1]$ is a weight hyperparameter used to balance the importance of these two objectives.

## 3.2 Reptile Meta-Learning for Better Distillation

In the original formulation of vanilla KD and previous KD methods, the teacher model was first fine-tuned and then frozen to transfer knowledge to the student by teaching fixed soft labels or intermediate features to the student for updating the student's parameters $\theta_S$. Instead, we introduce reptile meta-learning (Nichol et al., 2018) to optimize the teacher's parameters $\theta_T$ as the student's parameters $\theta_S$ are updated. Specifically, we designate the student and teacher as the inner-learner and meta-learner, respectively, to fit the bi-level optimization framework in reptile meta-learning. The

**Algorithm 1** Knowledge Distillation with Reptile Meta-Learning (ReptileDistil)

```
Require: $\theta_S$, $\theta_T$: parameters of the student and teacher models, respectively
Require: $\lambda$, $\mu$: learning rate of the studnet and teacher models, respectively
Require: $\mathcal{D}$: training set
1: while not done do
2:     for each batch of training set $\mathbf{d} \in \mathcal{D}$ do
3:         Copy an inner-learner $\theta'_S$ from the student model $\theta_S$: $\theta'_S \leftarrow \theta_S$
4:         Inner-update $\theta'_S$ with $\mathbf{d}$ and $\theta_T$: $\theta'_S \leftarrow \theta'_S - \lambda \frac{\partial \mathcal{L}_S^{KD}(\mathbf{d};\theta'_S;\theta_T)}{\partial \theta'_S}$
5:         Meta-update $\theta_T$ with the updated $\theta'_S$: $\theta_T \leftarrow \theta_T - \mu(\theta_T - \theta'_S(\theta_T))$
6:         Update $\theta_S$ with $\mathbf{d}$ and the updated $\theta_T$: $\theta_S \leftarrow \theta_S - \lambda \frac{\partial \mathcal{L}_S^{KD}(\mathbf{d};\theta_S;\theta_T)}{\partial \theta_S}$
7:     end for
8: end while
```

inner-learner is trained to accomplish a task or a distribution of tasks with the help of the meta-learner, and this procedure is called an inner-loop. In return, the meta-learner is optimized with a meta-objective that generally maximizes the expected performance of the inner-learner after the inner-loop, and this procedure is called a meta-loop. After the above two processes, the teacher can perceive the current state of the student and adjust its parameters in the direction of maximizing the student's performance. Then we use this optimized teacher to distill its more appropriate knowledge to the student, making the student learn better.

Formally, for each training step, we copy the student's parameters $\theta_S$ to an inner-learner as $\theta'_S$. Given a batch of training samples $\mathbf{d} = \{\mathbf{x}, y\} \in \mathcal{D}$, the inner-learner is inner-updated with one-step stochastic gradient descent, as follows,

$$\theta'_S \leftarrow \theta'_S - \lambda \frac{\partial \mathcal{L}_S^{KD}(\mathbf{d}; \theta'_S; \theta_T)}{\partial \theta'_S}, \tag{9}$$

where $\lambda$ is the learning rate of the student model. It is noteworthy that the updated inner-learner's parameters $\theta'_S$ are essentially a function of the teacher's parameters $\theta_T$ on account of learning $\theta'_S$ depends on $\theta_T$, i.e., $\theta'_S(\theta_T)$. The reptile meta-learning is then conducted by measuring the difference between the parameters of the teacher and the inner-learner as approximate gradients to meta-update the teacher's parameters $\theta_T$ via stochastic gradient descent, as follows,

$$\theta_T \leftarrow \theta_T - \mu(\theta_T - \theta'_S(\theta_T)), \tag{10}$$

where $\mu$ is the learning rate of the teacher model. Finally, we again apply vanilla KD between the updated teacher model and the original student model, as follows,

$$\theta_S \leftarrow \theta_S - \lambda \frac{\partial \mathcal{L}_S^{KD}(\mathbf{d}; \theta_S; \theta_T)}{\partial \theta_S}. \tag{11}$$

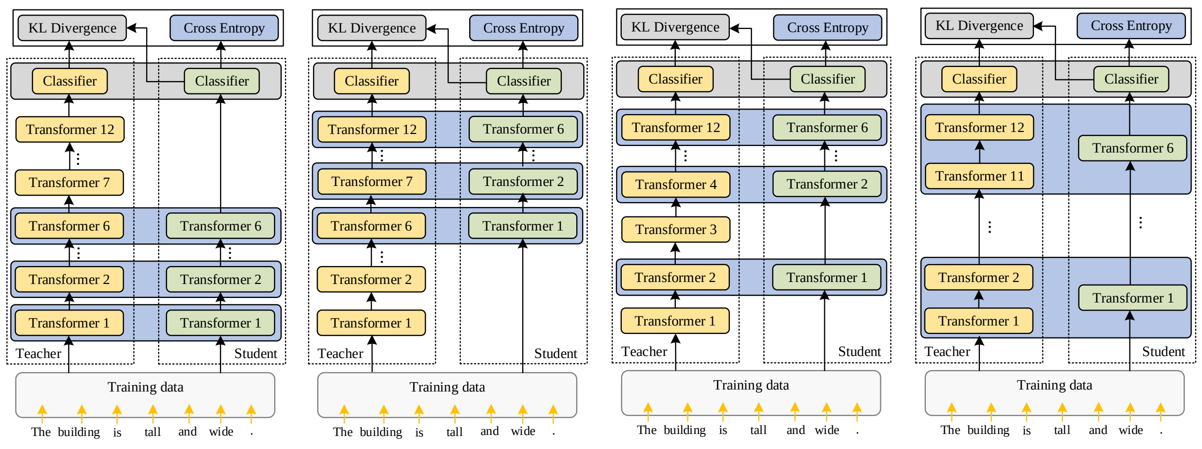
Such a meta-learning framework allows the teacher model to adjust its parameters according to the current learning state of the student model for better transferring knowledge to it. The detailed algorithm is illustrated in Algorithm 1.

### 3.3 Updating of Layer-wise Parameters

Notably, as shown in Eq. (10), the computation of gradients for updating the teacher's parameters requires the participation of the inner-learner's parameters and the teacher's parameters. However, the layers of these models are different, i.e., $K < L$, as mentioned. Taking knowledge distillation from a 12-layer teacher to a 6-layer student as an example, we also proposed four different strategies for updating the parameters of the teacher model. As listed below, the $L_T$ layers of the teacher model will be updated by the corresponding $k$ layers of the student model.

- **First-*k***: The first $k$ layers of the teacher will be updated, i.e., $L_T = \{1, 2, 3, 4, 5, 6\}$.
- **Last-*k***: The last $k$ layers of the teacher will be updated, i.e., $L_T = \{7, 8, 9, 10, 11, 12\}$.
- **Skip-*k***: The $2k$-th layers of the teacher will be updated, i.e., $L_T = \{2, 4, 6, 8, 10, 12\}$.
- **Both-*k***: The ($2k$-1)-th and $2k$-th layers of the teacher will be updated, i.e., $L_T = \{(1, 2), (3, 4), (5, 6), (7, 8), (9, 10), (11, 12)\}$.

Figure 2 shows the details of different strategies for updating the teacher's parameters, and the effect



(a) First-$k$ (b) Last-$k$ (c) Skip-$k$ (d) Both-$k$

Figure 2: Illustration of different strategies for updating the parameters of the teacher model.

of different strategies is discussed in the following section.

## 4 Experimental Setup

### 4.1 Datasets

We evaluate the proposed ReptileDistil on the commonly used GLUE benchmark (Wang et al., 2018), which is composed of nine natural language understanding tasks, including CoLA (Warstadt et al., 2019) for linguistic acceptability, SST-2 (Socher et al., 2013) for sentiment analysis, MRPC (Dolan and Brockett, 2005), STS-B (Cer et al., 2017), and QQP[1] for semantic similarity matching, MNLI (Williams et al., 2018), QNLI (Rajpurkar et al., 2016), and WNLI (Levesque et al., 2012) for natural language inference, and RTE (Dagan et al., 2005; Bar-Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009) for textual entailment. We excluded WNLI from GLUE following previous studies (Sanh et al., 2019; Sun et al., 2019; Turc et al., 2019; Jiao et al., 2020; Xu et al., 2020; Wang et al., 2021; Zhou et al., 2022).

### 4.2 Baselines

We apply the proposed ReptileDistil to a situation in which a 12-layer BERT$_{\text{BASE}}$ model with 110M parameters is distilled into a 6-layer BERT$_6$ model with 66M parameters and 1.94 times speedup, and compare it with several state-of-the-art BERT compression approaches, including DistilBERT (Sanh et al., 2019), BERT-PKD (Sun et al., 2019), PD (Turc et al., 2019), TinyBERT (Jiao et al., 2020), BERT-of-Theseus (Xu et al., 2020), MiniLM v2 (Wang et al., 2021), and MetaDistil (Zhou et al., 2022).

[1]https://quoradata.quora.com

### 4.3 Experimental Settings

In previous work, DistilBERT (Sanh et al., 2019), BERT-PKD (Sun et al., 2019), and BERT-of-Theseus (Xu et al., 2020) initialized their student model by truncating certain layers of a pre-trained BERT$_{\text{BASE}}$ model. More concretely, both BERT-PKD and BERT-of-Theseus initialized the student model with the first six layers of parameters from the pre-trained BERT$_{\text{BASE}}$ model. DistilBERT initialized the student model from the pre-trained BERT$_{\text{BASE}}$ model by taking one of every two layers. Different from the above approaches, TinyBERT (Jiao et al., 2020) initialized the student with the general TinyBERT[2], which can capture the general domain knowledge by learning from intermediate layers of the pre-trained BERT$_{\text{BASE}}$ model. Therefore, following TinyBERT, we initialize the student model with a 6-layer general TinyBERT$_6$ for better generalization and performance.

### 4.4 Implementation Details

The experiments include two stages, i.e., fine-tuning a teacher model and distilling the fine-tuned teacher model into a student model.

For experiments on fine-tuning the teacher, we use the same architecture in the original BERT (Devlin et al., 2019) and fine-tune each task separately. Specifically, for each task, we fine-tune the pre-trained BERT$_{\text{BASE}}$ model for 5 epochs using the

[2]https://github.com/huawei-noah/Pretrained-Language-Model/tree/master/TinyBERT

| Method | CoLA | SST-2 | MRPC | STS-B | QQP | MNLI | QNLI | RTE | Score |
|---|---|---|---|---|---|---|---|---|---|
| | (8.5k) | (67k) | (3.7k) | (5.7k) | (364k) | (393k) | (105k) | (2.5k) | |
| | Mcc | Acc | F1/Acc | Pear/Spea | F1/Acc | Acc m/mm | Acc | Acc | |
| **Development set** | | | | | | | | | |
| BERT$_{\text{BASE}}$ (Devlin et al., 2019) | 58.9 | 93.0 | 91.6/87.6 | 90.2/89.8 | 88.5/91.4 | 84.6/84.9 | 91.2 | 71.4 | - |
| DistilBERT (Sanh et al., 2019) | 51.3 | 91.3 | 87.5/- | -/86.9 | -/88.5 | 82.2/- | 89.2 | 59.9 | - |
| BERT-PKD (Sun et al., 2019) | 45.5 | 91.3 | 85.7/- | -/86.2 | -/88.4 | 81.3/- | 88.4 | 66.5 | - |
| PD (Turc et al., 2019) | - | 91.1 | 89.4/84.9 | - | 87.4/90.7 | 82.5/83.4 | 89.4 | 66.7 | - |
| TinyBERT (Jiao et al., 2020) | 54.0 | **93.0** | 90.6/86.3 | **90.1/89.6** | 88.0/91.1 | **84.5/84.5** | **91.1** | 73.4 | - |
| BERT-of-Theseus (Xu et al., 2020) | 51.1 | 91.5 | 89.0/- | -/88.7 | -/89.6 | 82.3/- | 89.5 | 68.2 | - |
| MiniLM v2 (Wang et al., 2021) | 52.5 | 92.4 | 88.9/- | - | -/91.1 | 84.2/- | 90.8 | 72.1 | - |
| MetaDistil (Zhou et al., 2022) | **58.6** | 92.3 | 91.1/86.8 | 89.4/89.1 | **88.1/91.0** | 83.5/83.8 | 90.4 | 69.4 | - |
| ReptileDistil | 54.8 | 92.2 | **91.6/87.7** | 89.5/89.3 | 87.6/90.1 | 83.7/83.7 | 90.5 | **75.3** | - |
| **Test set** | | | | | | | | | |
| BERT$_{\text{BASE}}$ (Devlin et al., 2019) | 52.1 | 93.5 | 88.9/84.8 | 87.7/85.8 | 71.2/89.2 | 84.6/83.4 | 90.5 | 66.4 | 78.3 |
| DistilBERT (Sanh et al., 2019) | 45.8 | 92.9 | 87.6/83.1 | 71.0/71.0 | 69.6/88.2 | 81.6/81.3 | 88.8 | 54.1 | 73.6 |
| BERT-PKD (Sun et al., 2019) | 43.5 | 92.0 | 85.0/79.9 | 83.4/81.6 | 70.7/88.9 | 81.5/81.0 | 89.0 | 65.5 | 75.6 |
| PD (Turc et al., 2019) | - | 91.8 | 86.8/81.7 | - | 70.4/88.9 | 82.8/82.2 | 88.9 | 65.3 | - |
| TinyBERT (Jiao et al., 2020) | **51.1** | 93.1 | 87.3/82.6 | 85.0/83.7 | 71.6/89.1 | **84.6/83.2** | 90.4 | 70.0 | 78.1 |
| BERT-of-Theseus (Xu et al., 2020) | 47.8 | 92.2 | 87.6/83.2 | 85.6/84.1 | **71.6/89.3** | 82.4/82.1 | 89.6 | 66.2 | 77.1 |
| MiniLM v2 (Wang et al., 2021) | - | 92.9 | 89.1/- | -/84.3 | 70.9/- | 83.8/83.3 | 90.2 | 69.2 | - |
| MetaDistil (Zhou et al., 2022) | 50.7 | **93.5** | 88.7/84.7 | 86.1/85.0 | 71.1/88.9 | 83.8/83.2 | 90.2 | 67.2 | 78.0 |
| ReptileDistil | 47.9 | 92.8 | **89.2/85.4** | **87.1/85.9** | 71.0/89.0 | 83.6/82.9 | **90.4** | **73.5** | **78.5** |

Table 1: Experiment results on the development and test sets of GLUE. The numbers and strings under each dataset indicate the number of training samples and the evaluation metrics, respectively. All student models listed above have the same architecture of 6 Transformer layers, 66M parameters, and 1.94 times speed-up, and are distilled from a BERT$_{\text{BASE}}$ model fine-tuned during the corresponding task. The results on the development set are reported from the corresponding original paper, and the results on the test set are reported from the official leaderboard of GLUE. Note that the column "**Score**" is reported from the official leaderboard and represents the average score for all tasks, including WNLI. **Acc** refers to accuracy, **Mcc** refers to Matthew's correlation, **Pear/Spea** refers to Pearson and Spearman correlation, respectively, and **Acc m/mm** refers to the accuracy of MNLI-m and MNLI-mm, respectively. The best results for each task are marked with boldface.

AdamW optimizer (Loshchilov and Hutter, 2019) and save the best checkpoint on the development set as the teacher, with a batch size of 32, a maximum sequence length of 128, a learning rate of {1e-5, 3e-5, 5e-5} with linear decay. Consequently, we can obtain a teacher model with comparable performance with BERT$_{\text{BASE}}$ reported on the GLUE official leaderboard[3].

For experiments on distilling the student, to reduce the hyperparameter search space, we set the maximum sequence length and batch size to 128 and 32, respectively, and fix the temperature $\tau$ and weight $\alpha$ as 5 and 0.5, respectively. To select the model with the best performance when applying the development set, a grid search strategy is then applied over the sets of the learning rate for the teacher model as {1e-5, 3e-5, 5e-5} and the learning rate for the student model as {1e-5, 3e-5, 5e-5} for 5 epochs and save the best checkpoint.

[3]https://gluebenchmark.com/leaderboard

## 5 Results and Analysis

### 5.1 Comparative Results

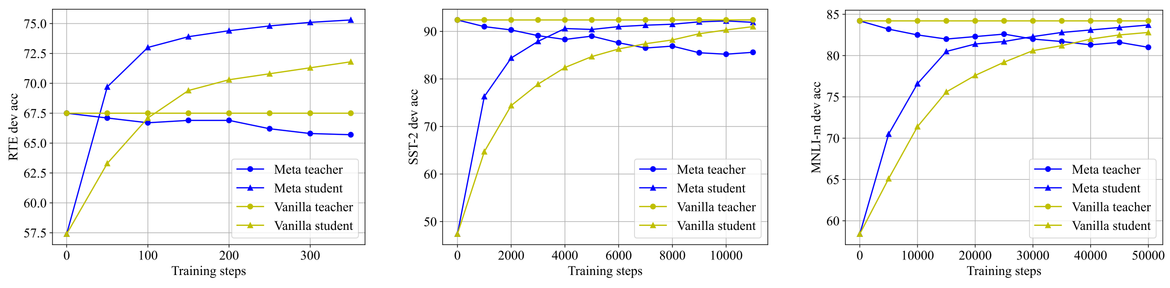
Table 1 summarizes the comparative results for both the development and test sets of the GLUE tasks. We also report the average score of all tasks, including WNLI, which can be obtained from the GLUE official leaderboard by submitting predictions to the GLUE test server. As the result shows, ReptileDistil achieves state-of-the-art performance on the overall average performance across all tasks of the GLUE benchmark. In detail, the proposed ReptileDistil outperforms baselines on 4 out of 8 tasks, and performs as well or better than BERT$_{\text{BASE}}$, particularly on small datasets, e.g., RTE, MRPC, and STS-B. Correspondingly, ReptileDistil is not significantly improved on large datasets, e.g., CoLA, SST-2, and MNLI. Nevertheless, it is still very close to MetaDistil and requires less training time and calculated amount. One potential reason is that ReptileDistil updates the teacher's parameters by calculating the differ-

| Method | CoLA | SST-2 | MRPC | STS-B | QQP | MNLI | QNLI | RTE |
|---|---|---|---|---|---|---|---|---|
| ReptileDistil | **54.8** | **92.2** | **91.6/87.7** | **89.5/89.3** | **87.6/90.1** | **83.7/83.7** | **90.5** | **75.3** |
| w/o Reptile | 51.9 | 91.0 | 90.0/86.5 | 88.9/88.6 | 86.5/89.4 | 82.8/83.1 | 90.0 | 72.8 |

Table 2: Ablation results in terms of removing reptile meta-learning (w/o Reptile).

| Method | CoLA | SST-2 | MRPC | STS-B | QQP | MNLI | QNLI | RTE |
|---|---|---|---|---|---|---|---|---|
| First-$k$ | 53.1 | 91.9 | 91.4/87.5 | 89.5/89.3 | 87.0/90.1 | 83.3/83.2 | **90.5** | 74.7 |
| Last-$k$ | 52.5 | 91.2 | 91.1/87.2 | 89.5/89.3 | 85.9/89.9 | 83.4/82.9 | 90.1 | 74.4 |
| Skip-$k$ | **54.8** | **92.2** | **91.6/87.7** | **89.5/89.3** | **87.6/90.1** | **83.7/83.7** | 90.3 | **75.3** |
| Both-$k$ | 51.3 | 90.5 | 90.8/87.0 | 89.5/89.3 | 86.4/89.5 | 81.9/81.8 | 90.0 | 73.6 |

Table 3: Performance comparison of four different layer-wise parameter update strategies.



(a) Results on RTE (b) Results on SST-2 (c) Results on MNLI-m

Figure 3: Validation accuracy curves of the teacher and student in ReptileDistil and vanilla KD, which are experimented on the development sets of RTE, SST-2 and MNLI-m datasets, respectively.

ence between the meta-learner (i.e., teacher) and inner-learner (i.e., student) as approximate gradients. Therefore, ReptileDistil can update the teacher model at a large pace on small datasets, leading to better and faster convergence of the teacher model. Conversely, MetaDistil uses the second-order derivatives as gradients to update the teacher's parameters. Although it is smoother than ReptileDistil, it requires a certain number of datasets to accomplish the model convergence. As a result, ReptileDistil is likely to train better in most circumstances, which is critical for improvement on small datasets but may not be so obvious for improvement on large datasets.

## 5.2 Ablation Analysis

To examine the effectiveness of the proposed ReptileDistil, we conducted an ablation test on the development sets of all tasks in terms of removing reptile meta-learning, allowing ReptileDistil to become vanilla KD where the teacher is fixed instead of dynamically evolving to adapt to the student throughout the distillation process. As the ablation results indicated in Table 2, remarkable performance degradation is observed after removing reptile meta-learning, indicating that updating the teacher model with the current state of the student model is crucial.

## 5.3 Effect of Updating Strategy

We further investigate the performance gain from four different layer-wise parameter update strategies of First-$k$, Last-$k$, Skip-$k$, and Both-$k$. Table 3 summarizes the comparative results on the development sets of all tasks. As shown, Skip-$k$ performs slightly better than the other three strategies in most cases since information across every $k$ layers involves low- to high-level semantic representations while the first $k$ layers or the last $k$ layers involve relatively homogeneous semantic representations. Particularly, the Both-$k$ strategy performs to be not competitive with the other three strategies since the Both-$k$ strategy can induce a mismatch between the parameters of the teacher and student, resulting in poor performance.

| Method | Training Time (Best) | Training Time (Match) | Memory Footprint | Best Acc/F1 |
|---|---|---|---|---|
| Vanilla KD (Hinton et al., 2015) | 10 min | 10 min | 3.7 GB | 90.0/86.5 |
| MetaDistil (Zhou et al., 2022) | 31 min | 15 min | 11.4 GB | 91.1/86.8 |
| ReptileDistil | 22 min | 9 min | 8.1 GB | 91.6/87.7 |

Table 4: Comparison of training time and memory footprint of ReptileDistil with vanillaKD and MetaDistil on MRPC. "**Training Time (Best)**" denotes the training time for each method to achieve its own best performance on the development set. "**Training Time (Match)**" denotes the training time for each method to match the best performance of vanilla KD on the development set. All experiments are conducted on a single Nvidia V100 GPU with a batch size of 4.

### 5.4 Improvement Analysis

Figure 3 shows the validation accuracy curves of the teacher and student in ReptileDistil and vanilla KD on the development sets of the small dataset RTE, the medium dataset SST-2, and the large dataset MNLI-m, respectively. As indicated, the teacher model is always fixed and constant in vanilla KD, whereas in ReptileDistil, it continuously adjusts itself adaptively. Although the accuracy of the teacher model continuously fluctuates and even declines, the student model still maintains a growing accuracy, indicating that the teacher model is not aimed at optimizing its performance but rather at optimizing its teaching ability to enable the student model to learn better and achieve higher performance. The results suggest that under the scenario of knowledge distillation, improving knowledge distillation by enhancing the teaching ability of the teacher model may be another potential direction.

### 5.5 Performance-Efficiency Tradeoff

MetaDistil (Zhou et al., 2022) inevitably needs to calculate the second-order derivatives for deriving the gradients to update the teacher model. Thus, more computational time and resources are required. However, in the proposed ReptileDistil, this limitation has been greatly improved. Specifically, the proposed ReptileDistil achieved a comparable or even better performance using the approximate gradient computation of reptile meta-learning and the proposed layer-wise parameter update strategies to replace the computation of the second-order derivatives in MAML. Meanwhile, the training time is reduced by approximately a third. To verify this, following MetaDistil, we performed an additional experiment on the MRPC task to compare the computational overhead of the proposed ReptileDistil with MetaDistil and vanilla KD. As shown in Table 4, the memory footprint of ReptileDistil is higher than vanilla KD but lower than MetaDistil. Moreover, both the training time and model convergence time of ReptileDistil are significantly lower than MetaDistil, proving that the proposed method can effectively alleviate the hardware limitation of MetaDistil and achieve better and faster convergence.

## 6 Conclusions

In this paper, a knowledge distillation method with reptile meta-learning was proposed to compress pre-trained language models, which also utilizes the current performance of the student model during each step of the distillation process as feedback to optimize the teacher model, allowing the teacher to learn to teach and thus produce more appropriate soft labels to teach the student. However, unlike previous knowledge distillation methods applying meta-learning, the proposed method avoids calculating the second-order derivatives and achieves better and faster convergence, owing to the integration of reptile meta-learning and the proposed layer-wise parameter update strategies. Extensive experiments demonstrated the competitive performance achieved by the proposed method. Future work attempts to continue to explore such a dynamic and interactive teacher-student distillation architecture to improve the performance of the distilled model further.

## Acknowledgements

This work was supported by the National Natural Science Foundation of China (NSFC) under Grant No.61966038, the Ministry of Science and Technology, Taiwan, ROC, under Grant No.MOST 111-2628-E-155-001-MY2 and the Postgraduate Research and Innovation Foundation of Yunnan University under Grant No.2021Y282. The authors would like to thank the anonymous reviewers for their constructive comments.

## References

Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second pascal recognising textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*.

Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernardo Magnini. 2009. The fifth pascal recognizing textual entailment challenge. In *Proceedings of the Third Text Analysis Conference*.

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 (SemEval 2017)*, pages 1–14.

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

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 (NAACL-HLT 2019)*, 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 (IWP 2005)*, pages 9–16.

Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. In *Proceedings of the 8th International Conference on Learning Representations (ICLR 2020)*.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, pages 1126–1135.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 1–9.

Mitchell A. Gordon, Kevin Duh, and Nicholas Andrews. 2020. Compressing bert: Studying the effects of weight pruning on transfer learning. In *Proceedings of the 5th Workshop on Representation Learning for NLP (RepL4NLP 2020)*, pages 143–155.

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

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.

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 *Proceedings of the 8th International Conference on Learning Representations (ICLR 2020)*.

Hector J. Levesque, Ernest Davis, and Leora Morgenstern. 2012. The winograd schema challenge. In *Proceedings of the Thirteenth International Conference on Knowledge Representation and Reasoning (KR 2012)*, pages 552–561.

Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. 2020. Fastbert: A self-distilling bert with adaptive inference time. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020)*, pages 6035–6044.

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

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *Proceedings of the 7th International Conference on Learning Representations (ICLR 2019)*.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems (NeurIPS 2019)*.

Alex Nichol, Joshua Achiam, and John Schulman. 2018. On first-order meta-Learning algorithms. *arXiv preprint arXiv:1803.02999*.

Dae Young Park, Moon-Hyun Cha, Changwook Jeong, Dae Sin Kim, and Bohyung Han. 2021. Learning student-friendly teacher networks for knowledge distillation. In *Advances in Neural Information Processing Systems (NeurIPS 2021)*, pages 13292–13303.

Hieu Pham, Zihang Dai, Qizhe Xie, and Quoc V. Le. 2021. Meta pseudo labels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2021)*, pages 11557–11568.

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 (EMNLP 2016)*, pages 2383–2392.

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

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 34th AAAI Conference on Artificial Intelligence (AAAI 2020)*, pages 8815–8821.

Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. 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 (EMNLP 2013)*, 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 (EMNLP-IJCNLP 2019)*, pages 4323–4332.

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*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, and Lukasz Kaiser. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS 2017)*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. 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.

Wenhui Wang, Hangbo Bao, Shaohan Huang, Li Dong, and Furu Wei. 2021. Minilmv2: Multi-head self-attention relation distillation for compressing pretrained transformers. In *Findings of the Association for Computational Linguistics (ACL-IJCNLP 2021)*, pages 2140–2151.

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

Adina Williams, Nikita Nangia, and Samuel R. 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 (NAACL-HLT 2018)*, pages 1112–1122.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. Deebert: Dynamic early exiting for accelerating bert inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020)*, pages 2246–2251.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. 2020. Bert-of-theseus: Compressing bert by progressive module replacing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020)*, pages 7859–7869.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems (NeurIPS 2019)*.

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

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. Bert loses patience: Fast and robust inference with early exit. In *Advances in Neural Information Processing Systems (NeurIPS 2020)*, pages 18330–18341.

Wangchunshu Zhou, Canwen Xu, and Julian McAuley. 2022. Bert learns to teach: Knowledge distillation with meta learning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL 2022)*, pages 7037–7049.