Learned Adapters Are Better Than Manually Designed Adapters

Yuming Zhang[1], **Peng Wang**[2*], **Ming Tan**[3*], **Wei Zhu**[4*†]
[1] College of Computer Science and Software Engineering, Shenzhen University
[2] Tomorrow Advancing Life
[3] Southern University of Science and Technology
[4] East China Normal University

Abstract

Recently, a series of works have looked into further improving the adapter-based tuning by manually designing better adapter architectures. Understandably, these manually designed solutions are sub-optimal. In this work, we propose the Learned Adapter framework to automatically learn the optimal adapter architectures for better task adaptation of pre-trained models (PTMs). First, we construct a unified search space for adapter architecture designs. In terms of the optimization method on the search space, we propose a simple-yet-effective method, GDNAS, for better architecture optimization. Extensive experiments show that our Learned Adapter framework can outperform the previous parameter-efficient tuning (PETuning) baselines while tuning comparable or fewer parameters. Moreover: (a) the learned adapter architectures are explainable and transferable across tasks. (b) We demonstrate that our architecture search space design is valid.

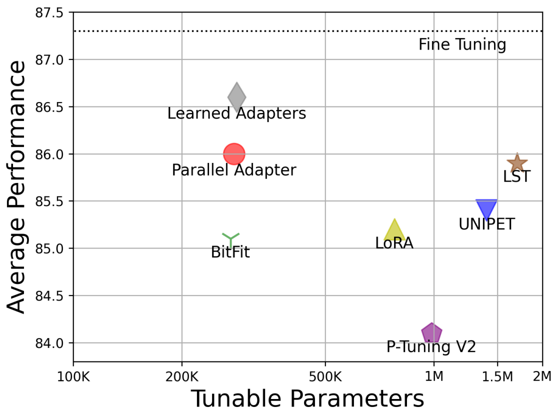


Figure 1: Overall comparison between our Learned Adapter framework and baselines. The x-axis is the number of tunable parameters, and the y-axis is the average performance on the GLUE benchmark with RoBERTa-large backbone. The details can be found in Section 5.

1 Introduction

Increasingly large pre-trained models (Han et al., 2021; Devlin et al., 2019; Peters et al., 2018; Liu et al., 2019b; Radford and Narasimhan, 2018; Raffel et al., 2019) built upon the Transformer architecture (Vaswani et al., 2017) have been emerging and achieving the state-of-the-art (SOTA) results on a variety of downstream tasks (Gao et al., 2023; Zhu et al., 2023; Li et al., 2019; Zhu, 2021b; Zuo et al., 2022; Zhang et al., 2022; Guo et al., 2021b; Zhu et al., 2021a; Sun et al., 2020; Zhu et al., 2019). Despite their effectiveness, these large-scale models also bring the curse of prohibitive computation (Zhu et al., 2021c; Zhu, 2021c; Sun et al., 2022) and storage costs during the adaptations to downstream tasks due to the gradient computation of the whole model and the giant size of the fine-tuned checkpoint.

Recently, parameter efficient tuning (PETuning) has raised much attention in the research field since it can only train a small portion of PTMs and keep the vast majority of parameters frozen, thus alleviating the computation costs during full fine-tuning. A series of studies (Houlsby et al., 2019; Pfeiffer et al., 2021; Mahabadi et al., 2021; Ben-Zaken et al., 2021; Hu et al., 2021; Guo et al., 2021a; Li and Liang, 2021; Lester et al., 2021) has verified that PETuning can achieve competitive performance compared to conventional fine-tuning with very few trainable parameters, resulting in a considerable reduction in model adaptation costs. Adapter-based methods (Houlsby et al., 2019; Pfeiffer et al., 2021; Mahabadi et al., 2021; He et al., 2021) inject newly-introduced layers after or around the attention or feed-forward modules of the Transformer block, and yield promising results by fine-tuning a small portion of the PTM's parameters.

Recently, a branch of recent research has ad-

*Equal contribution.
†Corresponding author: michaelwzhu91@gmail.com

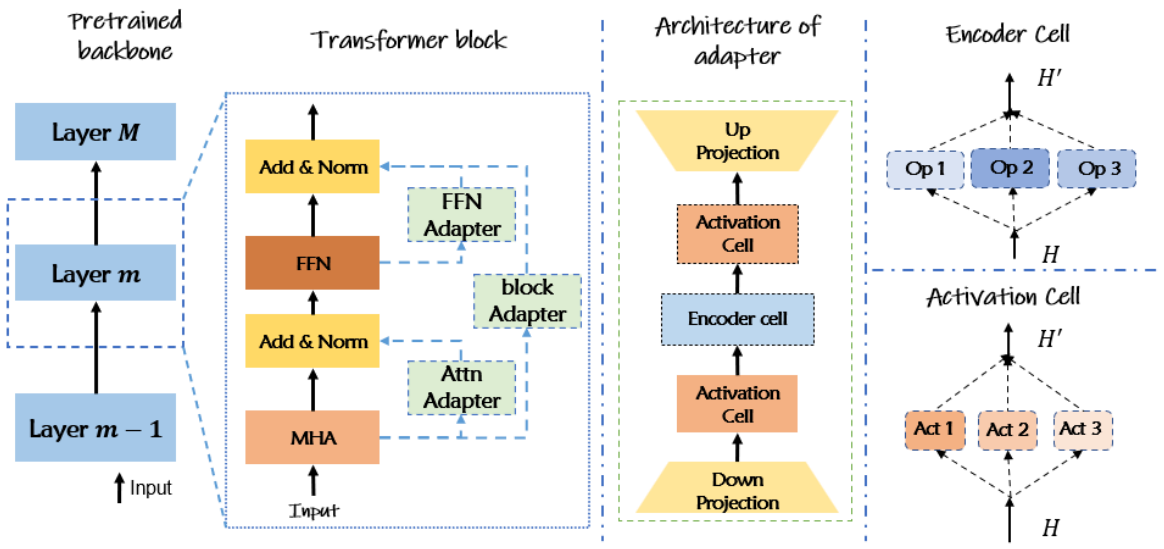


Figure 2: The overall framework of our Learned Adapter.

vanced the understanding of adapter-based tuning more deeply and improved the adapters' architectures to improve parameter efficiency further. Adaptable adapters (Moosavi et al., 2022) propose that adapters at different layers should have different activation functions. Thus they fit the rational activation functions to downstream tasks during parameter tuning. AdapterDrop (Rücklé et al., 2020) tries to reduce the number of adapters' parameter number by not inserting adapters on the lower layers. He et al. (2021) bridge connections among different PETuning approaches to form a unified framework and further propose to insert adapters in parallel to the modules of the Transformer block. Jie and Deng (2022) and Sung et al. (2022) propose to add encoding operations between the projection layers of an adapter and achieve better PETuning performances. The above empirical evidence implies that altering the adapters' architecture designs can help to improve the PETuning performances of adapters with even fewer tunable parameters. Predictably, such an optimal architecture is difficult to construct manually and may vary across different PTM backbones and tasks. Therefore, we propose to search for the optimal architecture of adapters automatically.

We present the **Learned Adapter** framework to search for the optimal architecture of adapters automatically. We first construct a unified search space (Figure 2) that considers various design choices of adapters, including the activation functions, encoding operations, and how the adapters are connected to the PTM backbone. In terms of the specific methodology for optimization on the search space, make a simple-yet-effective modification to the optimization method in DARTS (Liu et al., 2019a), which is better at identifying the proper components for adapters at different intermediate layers.

We conduct extensive experiments to study the effectiveness of our Learned Adapter framework. The experimental results show that with 0.068% parameters, we can recover 99.5% fine-tuning performances on the GLUE (Wang et al., 2018) benchmark. Moreover, the searched architecture outperforms the manually designed PETuning baselines while tuning fewer parameters. Figure 1 depicts the overall comparison between our Learned Adapter and the baselines. Furthermore, the learned architectures of adapters are transferable across tasks, which significantly strengthens the usefulness of the searched structures. Further experiments demonstrate that our newly proposed search space for adapters is valid.

2 Related work

Adapter-based tuning. One of the most important research lines of PETuning is adapter-based tuning. Adapter (Houlsby et al., 2019) inserts adapter modules with bottleneck architecture between every consecutive Transformer (Vaswani et al., 2017) sublayers. AdapterFusion (Pfeiffer et al., 2021) only inserts sequential adapters after the feed-forward module. Adapter-based tuning methods have comparable results with model tuning when only tuning a fraction of the backbone model's parameter number. Due to their amazing results on PETuning, a branch of literature has investigated the architecture of adapters in search of further improvements. He et al. (2021) analyze a wide range of PETuning methods and show that they are essentially equivalent. They also propose the general architecture of PETuning. AdapterDrop (Rücklé et al., 2020) investigates the efficiency of removing adapters from lower layers. Adaptive adapters (Moosavi et al., 2022) investigate the activation functions of adapters and propose to learn the activation functions of adapters via optimizing the parameters of rational functions as a part of the model parameters. Compacter (Mahabadi et al., 2021) uses low-rank parameterized hypercomplex multiplication (Le et al., 2021) to compress adapters' tunable parameters. There is also work (Sung et al., 2022; Jie and Deng, 2022) trying to add different encoding operations, like self-attention operations and convolutions between the bottleneck structure of adapters, and achieve better performances.

Our work complements this branch of literature by investigating: (a) whether and how the adapter architectures affect the PETuning performances, and whether different layers of PTMs need different adapter architectures; (b) whether we can obtain better adapter architectures via neural architecture search.

Other PETuning methods Another main research line of PETuning is the prompt-based tuning that inserts some additional soft prompts into the hidden states instead of injecting new neural modules to PTMs. Prompt tuning (Lester et al., 2021) and P-tuning (Liu et al., 2022) insert a soft prompt to word embeddings only, and can achieve competitive results when applied to supersized PTMs. Prefix-tuning (Li and Liang, 2021) and P-tuning v2 (Liu et al., 2021) insert prompts to every hidden layer of PTM. IDPG (Wu et al., 2022) uses the prompt generator with parameterized hypercomplex multiplication (Le et al., 2021) to generate a soft prompt for every instance. There are also some other popular PETuning methods, such as BitFit (Ben-Zaken et al., 2021) which only tunes the bias terms, LoRA (Hu et al., 2021) which optimizes low-rank decomposition matrices of the weights within self-attention layers.

Neural architecture search In the early attempts, neural architecture search (NAS) requires massive computations, like thousands of GPU days (Zoph and Le, 2017; Zoph et al., 2018; Liu et al., 2018). Recently, a particular group of one-shot NAS, led by the seminal work DARTS (Liu et al., 2019a) has attracted much attention. DARTS formulates the search space into a super-network that can adjust itself in a continuous space so that the network and architectural parameters can be optimized alternately (bi-level optimization) using gradient descent. A series of literature try to improve the performance and efficiency of DARTS, such as Xie et al. (2019), Chen et al. (2021), Chu et al. (2021), Nayman et al. (2019). SNAS (Xie et al., 2019) reformulate DARTS as a credit assignment task while maintaining the differentiability. Gao et al. (2020) penalize the entropy of the architecture parameters to encourage discretization on the hyper-network. P-DARTS (Chen et al., 2021) analyze the issues during the DARTS bi-level optimization, and propose a series of modifications. PC-DARTS (Xu et al., 2021) reduces the memory cost during search by sampling a portion of the channels in super-networks. FairDARTS (Chu et al., 2021) change the softmax operations in DARTS into sigmoid and introduce a zero-one loss to prune the architectural parameters. XNAS (Nayman et al., 2019) dynamically wipes out inferior architectures and enhances superior ones. NAS is widely applied in both computer vision and natural language processing, especially in knowledge distillation (Zhu, 2021a; Zhang et al., 2021).

Our work complements the literature by examining the optimization of DARTS on our search space and propose a new training procedure that does not require re-training after discretization.

3 Search space of Learned Adapter

3.1 Pilot experiments and motivations

In this subsection, we conduct a series of experiments on the RTE (Dagan et al., 2005) and MRPC (Dolan and Brockett, 2005) datasets to demonstrate the necessity of investigating the architecture of adapters. The baseline model $\mathcal{M}$ is RoBERTa-large model with an parallel adapter at the feed-forward module (FFN adapter) (He et al., 2021). The backbone model is frozen and we only tune the adapters on downstream tasks. The bottleneck dimension is 32 and the activation function is ReLU. The other experimental settings follows Appendix B. We now consider a series of simple modifications to the baseline model.

Modifying the activation function We replace the activation functions of the adapters from ReLU to GeLU, SWISH or Tanh, while keeping the other

Model	RTE	MRPC
$\mathcal{M}$	79.1 (0.5)	89.3 (0.4)
$\mathcal{M}_{gelu}$	79.3 (0.2)	89.5 (0.3)
$\mathcal{M}_{swish}$	79.6 (0.3)	89.2 (0.4)
$\mathcal{M}_{tanh}$	79.2 (0.5)	88.9 (0.6)
$\mathcal{M}_{sa}$	79.5 (0.3)	89.6 (0.2)
$\mathcal{M}_{conv}$	79.5 (0.6)	89.4 (0.5)
$\mathcal{M}_{attn}$	79.0 (0.4)	89.1 (0.3)
$\mathcal{M}_{block}$	79.6 (0.5)	89.4 (0.3)

Table 1: Results of the pilot experiments.

settings unchanged. The three modified models are denoted as $\mathcal{M}_{gelu}$, $\mathcal{M}_{swish}$ and $\mathcal{B}_{tanh}$, respectively.

Adding encoding operations We add a self-attention operation (Vaswani et al., 2017) or a convolutional operation of kernel size 3 after the down-projection and before the activation function. The two variants of model $\mathcal{M}$ are denoted as $\mathcal{M}_{sa}$ and $\mathcal{M}_{conv}$, respectively. Since extra operations introduce more parameters, we reduce the bottleneck dimension of $\mathcal{M}_{sa}$ and $\mathcal{M}_{conv}$ to 24 to ensure fair comparison.

Alternative adapter placements Instead of inserting the adapter around the FFN module of the transformer block, we now consider: (1) $\mathcal{M}_{attn}$ inserts the adapters at the attention modules (attn adapter); (2) $\mathcal{M}_{block}$ inserts the adapters around the entire transformer block (block adapter). Note that the setting of block adapters is theoretically supported by the general framework of PETuning in He et al. (2021) but not considered by the previous work. In this work, we will demonstrate the usefulness of block adapters via experiments.

Table 1 reports the experimental results of the above models. The evaluation metrics for the RTE and MRPC tasks are introduced by Appendix A.2. We can see that the four simple modifications to the baseline model, $\mathcal{M}_{gelu}$, $\mathcal{M}_{sa}$, $\mathcal{M}_{conv}$ and $\mathcal{M}_{block}$, can slightly outperform $\mathcal{M}$, demonstrating that the adapter architectures are essential for adapter tuning, and it is promising to design better adapter architectures for better adapter tuning performances.

The pilot experiments raise a vital research question: **What are the optimal architectures for adapters?** Obviously, such an optimal architecture will be different across tasks and PTM models and even across different intermediate layers of a PTM, making it impossible for manual designs. We are motivated to investigate the problem of optimizing the architectures of adapters via neural architecture search.

3.2 General architecture of adapters

As depicted in Figure 2, we now construct the search space of the Learned Adapter. The adapter is a bottleneck architecture with bottleneck dimension r, consisting of down-projection layer MLP$_d$, an activation function g_1, an encoder layer Enc, another activation function g_2 and finally a up-projection layer MLP$_u$. Formally, the hidden states h_x goes through the adapter and becomes

$$h_x^{(A)} = \text{MLP}_u(g_2(\text{Enc}(g_1(\text{MLP}_d(h_x))))). \quad (1)$$

Following He et al. (2021), the hidden representation h_x will also go through the backbone's certain encoding module BEnc, and the adapted hidden states will become $h_x^{'} = \text{BEnc}(h_x) + h_x^{(A)}$. Following (Wu et al., 2022; Mahabadi et al., 2021; Le et al., 2021), we employ the parameterized hypercomplex multiplication (PHM) layer (Le et al., 2021) with parameter n to reduce the parameters of MLP$_d$ and MLP$_u$. The PHM layer has a parameter complexity of $\mathcal{O}(rd/n)$, reducing the parameters of the projection layers by at most $\frac{1}{n}$.

3.3 Search space

We now formally introduce the search space of our Learned Adapter framework. The whole search space contains three types of search cells as shown in Figure 2:

Activation Search Cell The Activation Search Cell is designated to choose the proper activation functions g_1 and g_1 from several candidates. Similar to So et al. (2019), the collection of candidate activation functions considered is: (a) **ReLU** (Agarap, 2018); (b) **GeLU** (Hendrycks and Gimpel, 2016); (c) **SWISH** (Ramachandran et al., 2017); (d) **Tanh** (Krizhevsky et al., 2012); (e) **NullAct**, which means no activation function and not to make changes to the input.

Encoder Search Cell As is shown in Figure 2, different from (Wang et al., 2020; Zhu et al., 2021b), we construct our encoder cell as a simple DAG with a single edge. Our collection of encoder operations consists of the following four groups: (a) 1-d convolutional layers, with stride 1, same padding, output filters equal to the input's dimension, and kernel size equal to 1, 3, 5, or 7 (denoted as **conv**_k, $k = 1, 3, 5, 7$). (b) Multi-head self-attention (MHA) layers (Vaswani et al., 2017), with head size equal to 2 or 8 (denoted as **mha_k**,

$k = 2, 8$). (c) Skip connection (He et al., 2015), denoted as **skip-connect**. (d) The null encoding operation that multiplies zero tensors to the input (**null**).[1]

Adapter Placement Search Cell This search cell is designated to decide the placement of the adapter in an intermediate transformer block. We consider three candidate placements shown in Figure 2:[2] (a) **FFN adapter**, that is, to insert the adapter in parallel to the feed-forward module; (b) **Attn adapter**, parallel to the self-attention module; (c) **Block adapter** inserts the adapter in parallel to the whole transformer block. This placement option is supported by the theoretical analysis of He et al. (2021) but has not been considered by the literature. In the experiments, we will show that including the above three choices for adapter placements are necessary.

Note that the above three search cells are single-edge DAGs. Following Pham et al. (2018); Wang et al. (2020); Zhu et al. (2021b); Zhu (2021a); Zhu et al. (2021d), we consider the macro search space, that is, different adapter architectures are learned for different intermediate layers. Intuitively, the macro search space allows for idiosyncratic architectures for different intermediate layers, leading to easier model adaptation.

Despite the simple structures of search cell DAGs compared to the general NAS literature, our macro search space can result in 6.38e+90 combinations of different adapter architectures across different intermediate layers of the PTM backbones. Note that our search space contains the adapter architectures from Sung et al. (2022); Jie and Deng (2022) as special cases.

4 Search Method

4.1 Preliminaries on DARTS

Assume there is a pre-defined space of operations denoted by $\mathcal{O}$, where each element, $o(\cdot)$, denotes a neural network operation, like convolutional operation, self-attention, activation, etc. DARTS (Liu et al., 2019a) initialize a hyper-network in which each block is a search cell, that is, a fully connected directed acyclic graph (DAG) with N nodes. Let (i, j) denote a pair of nodes in the DAG. The core idea of DARTS is to use a weighted sum to include all $|\mathcal{O}|$ operations in $\mathcal{O}$, $f_{i,j}(z_i) = \sum_{o \in \mathcal{O}} a_{i,j}^{o} \cdot o(z_i)$, where $a_{i,j}^{o} = \frac{\exp \alpha_{i,j}^{o}}{\sum_{o' \in \mathcal{O}} \exp \alpha_{i,j}^{o'}}$, z_i denotes the output of the i-th node, and $\alpha_{i,j}^{o}$ is the architectural parameters that represents the weight (or the importance score) of $o(\cdot)$ in edge (i, j). The output of a node is the sum of all input flow, i.e., $z_j = \sum_{i<j} f_{i,j}(z_i)$. The output of the entire cell is formed by summing the last two nodes.

This design makes the entire framework differentiable to both layer weights and architectural parameters $\alpha_{i,j}^{o}$ so that it is possible to perform architecture search in an end-to-end fashion. After the search process is completed, the discretization procedure extracts the final sub-network by selecting the best operation on each edge and dropping the lower-score operations. And the final sub-network will train on the original train set with randomly initialized parameters.

4.2 Discussion on the search method

The standard optimization method for the above framework is the bi-level optimization proposed in DARTS (Liu et al., 2019a). However, there are recent works arguing that the single-level optimization method could also work for the DARTS framework. As pointed out by Bi et al. (2019) and Bi et al. (2020), bi-level optimization suffers considerable inaccuracy of gradient estimation and the potential instability can increase with the complexity of the search space. And Bi et al. (2020) conduct experiments to demonstrate that one-level optimization performs comparably with bi-level optimization but with better efficiency. Their experiments are conducted mainly on computer vision benchmarks like CIFAR-10 (Krizhevsky et al., 2012). In this work, we would like to investigate which optimization method is better under our framework.

Note that the original DARTS requires one to re-train the learned networks from scratch after the search procedure, which definitely introduce additional computation costs. In this work, we propose to gradually discretize the hyper-network and obtain a sub-network without re-training. We will refer to this method as gradually discretizing neural architecture search (GDNAS). We first train the complete hyper-network M for K_1 epochs. Then we select a edge e in the search space to discretize (for example, the edge in the encoder cell). Dis-

[1]Choosing this operation means the model decides to discard an operation in the encoder, thus making the encoder a lighter architecture.

[2]Through our initial experiments, we find no improvements to include other standard adapter placement options, like Houlsby et al. (2019); Pfeiffer et al. (2021)

cretization simply means selecting the operation o_*^e with highest architectural parameter, and drop the other operations. Now we have obtain a new reduced hyper-network M. The discretized edge may cause the performance of the hyper-network to drop significantly, so we further finetune the hyper-network M for K_2 epochs.

In addition to the advantage of not to retrain the learned network, GDNAS retains the knowledge in the hyper-network, and obtain the performance gains against the re-trained sub-network. This is analogous to the model pruning literature, where a network pruned from a larger one is usually better than the network trained from scratch (Liang et al., 2021).

Algorithm 1: GDNAS

```
Input: A hyper-network $M$, all edges $\mathcal{E}$ on
       hyper-network $M$;
Output: Set of selected operations $\{o_e^*\}_{e\in\mathcal{E}}$
Data: Training set $D_{train}$, a batch of
      validation data $B_{val}$
1 Train hyper-network $M$ on the training set
    $D_{train}$ for $K_1$ epochs;
2 for edge $e$ in $\mathcal{E}$ do
3     Select the best operation
        $o_e^* \leftarrow \arg\max_o \alpha_e^o$;
4     Discretize edge $e$ of hyper-network $M$
        by only keeping $o_e^*$;
5     Further train the hyper-network $M$ on
        $D_{train}$ for $K_2$ epochs;
```

5 Experiments

5.1 Evaluation datasets

We evaluate the performance of the methods on the GLUE (Wang et al., 2018) benchmarks. These benchmarks cover multiple tasks of paraphrase detection (MRPC, QQP), sentiment classification (SST-2), natural language inference (MNLI, RTE, QNLI), linguistic acceptability (CoLA).[3]

Since the original test sets of the GLUE benchmark are not publicly available, we follow Zhang et al. (2020) and Mahabadi et al. (2021) to construct the train/dev/test splits as follows to ensure a fiar comparison: (a) for datasets with fewer than 10k samples (RTE, MRPC, STS-B, CoLA), we divide the original validation set in half, using one half for validation and the other for testing. (b) for larger datasets, we split 1k samples from the training set as the development set, and use the original development set as the test set. The detailed statistics and evaluation metrics of the GLUE benchmark is presented in Table 7 of Appendix A.

5.2 Experiment Settings

We run all the experiments on NVIDIA V100 32GB GPUs. We mainly evaluate our method on the GLUE benchmarks with the RoBERTa-large (Liu et al., 2019c) backbone model. We also evaluate our framework with the DeBERTa-large (He et al., 2020) and GPT2-large (Radford et al., 2019) backbones. We use the HugginFace Transformers (Wolf et al., 2020) for implementing all the methods. Unless otherwise specified, GDNAS will adopt the bi-level optimization method of DARTS. For GDNAS' discretization procedure, we set $K_1 = 5$ and $K_2 = 0.5$ on large datasets (SST-2, QNLI, QQP and MNLI), and $K_1 = 20$ and $K_2 = 2$ on low-resource datasets. And the batch size is set to 128 for dataset with more than 10k training samples, and 32 otherwise. For Learned Adapter, we set the bottleneck dimension r to 32 and select at most one adapter at each transformer layer. For the PHM layers, we use the PyTorch implementation of Le et al. (2021) and set n to 8. We run each task under 5 different random seeds and report the average performance and standard deviation. More details of the experimental settings are put in Appendix B.

5.3 Baselines

We compare our Learned Adapter framework with the current SOTA baseline methods.

Fine-tune The traditional fine-tuning methods that train all parameters in the PTM backbone.

Adapter-based tuning For adapter-based tuning methods, we compare with: (1) Adapter (Houlsby et al., 2019); (2) Compacter (Mahabadi et al., 2021); (3) Parallel Adapter proposed by (He et al., 2021) added on the FFN module; (4) LST (Sung et al., 2022). We re-implement Parallel Adapter with PHM projection layers ($n = 8$).

Prompt-based tuning For prompt-based tuning methods, we compare with (1) Prompt Tuning (Lester et al., 2021), (2) P-tuning v2 (Liu et al., 2021). The number of prompt tokens in these methods is set to 20.

[3]Following Devlin et al. (2019) and (Raffel et al., 2019), as a common practice, we do not experiment with the WNLI task (Levesque et al., 2011) due to its adversarial nature with respect to the training set.

Method	Tunable Params	CoLA (mcc)	SST-2 (acc)	MRPC (acc-f1)	QQP (acc-f1)	STS-B (corr)	MNLI (acc)	QNLI (acc)	RTE (acc)	Avg
Baselines										
RoBERTa-large	355M	65.3	95.4	91.5	90.3	91.8	89.8	94.5	80.1	87.3
Prompt Tuning	21K	54.6 (2.5)	90.9 (0.5)	74.8 (1.3)	87.4 (0.5)	90.1 (0.3)	83.4 (0.2)	92.4 (0.3)	68.7 (1.9)	80.3
P-tuning v2	985K	57.3 (2.1)	92.3 (0.3)	86.9 (1.2)	88.1 (0.4)	90.4 (0.1)	87.3 (0.2)	92.9 (0.4)	77.2 (2.1)	84.1
BitFit	273K	59.2 (0.9)	94.1 (0.3)	88.6 (0.8)	88.5 (0.6)	91.1 (0.3)	87.3 (0.1)	93.2 (0.2)	78.6 (1.4)	85.1
LoRA	778K	59.7 (1.4)	93.6 (0.1)	88.9 (0.7)	88.3 (0.4)	90.9 (0.2)	87.9 (0.2)	93.2 (0.1)	78.8 (1.3)	85.2
UNIPELT	1.4M	61.5 (1.7)	93.6 (0.2)	89.2 (0.8)	88.7 (0.4)	91.0 (0.5)	87.5 (0.2)	92.9 (0.1)	79.1 (0.9)	85.4
Adapter	1.6M	63.1 (1.4)	93.8 (0.1)	87.5 (0.8)	88.7 (0.4)	90.9 (0.4)	88.5 (0.2)	93.2 (0.1)	78.9 (1.3)	85.6
LST	1.7M	63.6 (1.5)	93.9 (0.2)	89.2 (0.7)	88.7 (0.2)	91.2 (0.4)	88.3 (0.1)	93.0 (0.2)	79.2 (0.9)	85.9
Parallel Adapters	279K	63.8 (1.5)	94.2 (0.3)	89.3 (0.5)	88.9 (0.6)	91.1 (0.4)	88.3 (0.1)	93.2 (0.1)	79.1 (0.5)	86.0
Compacter	279K	63.7 (0.9)	94.2 (0.4)	89.1 (0.7)	88.6 (0.3)	90.8 (0.2)	88.2 (0.2)	92.8 (0.2)	78.7 (0.9)	85.8
Our proposed methods										
Learned Adapter	294K	64.3 (0.9)	94.9 (0.3)	89.8 (0.9)	89.2 (0.3)	91.3 (0.2)	88.6 (0.2)	93.5 (0.2)	80.4 (1.3)	86.5

Table 2: The Overall comparison on the GLUE benchmark with RoBERTa-large backbone. We report mean and standard deviation of performance over 5 random seeds. Bold and Underline indicate the best and the second best results. The metric for each task is explained in Appendix A.2.

Other PETuning methods We also compare: (1) BitFit (Ben-Zaken et al., 2021); (3) LoRA (Hu et al., 2021); (4) UNIPET (Mao et al., 2021) combines different types of PETuning methods in an automatical manner.

We implement Aadpter, BitFit, and LoRA using the OpenDelta[4] library. Other baselines are implemented using their open-sourced codes with their default settings. For a fair comparison, we do not use supplementary training like Wu et al. (2022) to enhance performance.

5.4 Results on the GLUE benchmark

Table 2 shows the results on GLUE with RoBERTa-large. Our Learned Adapter framework, outperforms the previous PETuning methods and notably preserves 99.4% performance of the full-model fine-tuning method while only tuning 240K to 300K parameters.

We can observe from Table 2 that: (a) Note that our Learned Adapter framework obtains further improvements by automatically designing adapter architectures for different intermediate layers of the PTM. (b) Note that although we add encoding operations in adapters, the total tunable parameters of the Learned Adapter in the macro setting are fewer than Compacter since our framework can automatically drop adapters on certain layers when necessary.

5.5 Further analysis

Explanations of the searched architectures To understand the searched adapter architectures under our Learned Adapter framework, we present the learned adapter architectures on the RTE and SST-2 tasks on Table 9 and 10 in Appendix D, respectively. From the learned adapter architectures, we can observe that: (a) The adapter architecture varies for different layers, showing that different layers require different adapter architectures. (b) On each task, Learned Adapter chooses the **null** encoding operation on 3-5 intermediate layers, meaning to drop the adapters on these layers. (c) Regarding the adapter placement choices, we find that on each task, all three placement candidates, FFN adapter, Attn adapter, and Block adapter, are selected. This observation demonstrates that introducing block adapters into our search space is necessary. (d) Most adapters select the convolutional operations, and multi-head self-attention operations tend to occur in adapters of deeper layers. (e) around half of the learned adapters choose the **NullAct** for the second activation function g_2. Furthermore, we observe that there are adapters on the deeper transformer layers that requires no activation function but an encoder operation, demonstrating novel design patterns for adapters.

Exploring the limit of parameter efficiency To explore the limit of parameter efficiency, we train the Learned Adapter, and Compacter (Mahabadi et al., 2021) with different rank parameters $n \in \{1, 2, 4, 8, 16, 32\}$. Note that in the main experiments, we set n equal to 4. With larger n, the parameters of adapters will increase proportionally. In Figure 3, we demonstrate the results of the RTE and SST-2 tasks. We can see that the advantages of our Learned Adapter framework become more prominent with lower tunable parameter budgets. The results demonstrate that our framework can

[4]https://github.com/thunlp/OpenDelta

Source	Target RTE	Target SST-2
RTE	**80.4** (1.3)	94.6 (0.1)
SST-2	80.3 (1.4)	**94.9** (0.3)

Table 3: Architecture transfer from source datasets to target datasets. The target datasets are in the column names, and the source datasets are in the row names.

Search space	**RTE (acc)**	**SST-2 (acc)**
$\mathcal{S}$	**80.4** (1.3)	**94.9** (0.3)
$\mathcal{S}_1$	80.1 (1.5)	94.6 (0.3)
$\mathcal{S}_2$	79.8 (1.0)	94.4 (0.4)
Parallel Adapter	79.1 (0.5)	94.2 (0.3)

Table 4: Experimental results for the ablation study of our Learned Adapter search space.

effectively deliver the most proper architectures under the given parameter budgets and boost the performance of adapter-based tuning.

Architectures' Transferability We now evaluate the transferability of the searched structures by the Learned Adapter.The RTE and SST-2 datasets are used as source and target datasets. We search the adapter architectures on the source dataset and train the searched architectures from scratch on the target task, and report the average and standard deviation of scores over 5 random seeds on table 3. We can see from Table 3 that the searched architectures are highly transferable. The transferred architectures can already achieve comparable or better performances than most baseline models (Table 2). The transferability guarantees the reusability of the searched adapter architectures.

Ablation studies on the search space We now conduct an ablation study of our search space by reducing our search space $\mathcal{S}$ to a singleton step-by-step : (a) reduce the activation cells by only keeping the **ReLU** activation for g_1 and the **NullAct** for g_2 ($\mathcal{S}_1$); (b) further reduce the encoder cell to only include **skip-connect** ($\mathcal{S}_2$); (c) further reduce the adapter placement cell to only include FFN adapter, and now the search space only contains Parallel Adapter (He et al., 2021). Table 4 reports the search results on different search spaces, showing that that dropping any components of the whole search space results in performance losses. The results demonstrate that each search cell in our search space design is necessary and beneficial.

Working with other PTM backbones To verify the general applicability of our Learned Adapter

Method	**Tunable Params**	**RTE (acc)**	**SST-2 (acc)**
DeBERTa-large backbone			
Fine-tuning	406M	82.1 (1.1)	95.7 (0.2)
Adapters	1.6M	80.7 (1.3)	93.8 (0.1)
Parallel Adapters	279K	81.5 (1.2)	93.9 (0.3)
Learned Adapter	245K	**82.0** (0.9)	**94.5** (0.3)
GPT2-large backbone			
Fine-tuning	774M	79.5 (0.8)	95.5 (0.1)
Adapters	2.7M	78.1 (0.9)	93.9 (0.1)
Parallel Adapters	462K	78.5 (1.2)	93.8 (0.1)
Learned Adapter	482K	79.1 (0.6)	94.2 (0.2)

Table 5: Results on 2 GLUE tasks using DeBERTa-large and GPT2-large models as the backbone. Bold indicates the best PETuning results.

framework, we also conduct experiments on two other widely used PTM backbones, DeBERTa-large (He et al., 2020), and GPT2-large (Radford et al., 2019). The results are shown in Table 5. Our Learned Adapter successfully outperforms the adapter-based tuning baselines on both pre-trained backbones. This result enhances the reliability of our framework.

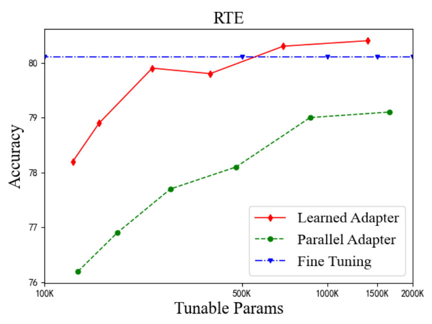
We now validate our Learned Adapter framework on other pre-trained backbone: DeBERTa-large (He et al., 2020) and GPT2-large (Radford et al., 2019). The results are presented in Table 5.

5.6 Discussions on the search method

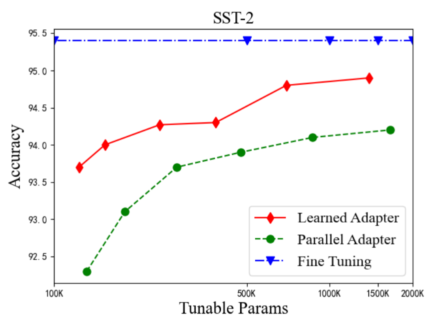
Search efficiency of GDNAS We use the RTE task to demonstrate the search efficiency. Running the RTE task with DARTS takes 1.5h (70.5min for bi-level optimization for 25 epochs and 21.6min for re-training with 25 epochs). Since GDNAS does not require re-training, it requires 1.2h (73.3min for training the hyper-network for $k_1 + 3 * K_2 = 26$ epochs). Our method consumes around three times the training time of Parallel Adapter (He et al., 2021), which is affordable compared to manually designing different architectures and running numerous evaluations.

Ablation study of search methods We now run Learned Adapter with GDNAS with single-level optimization, the original DARTS (Liu et al., 2019a) and ENAS (Cai et al., 2018). The results are shown in Table 6. The results demonstrate that our GDNAS are effective in discovering better adapter architectures. In addition, the results demonstrate that bi-level optimization obtains slightly better results.

Performance on a NAS benchmark To further



(a) RTE



(b) SST-2

Figure 3: Performances under different PHM rank n. The x-axis represents the number of tunable parameters, and the y-axis represents the performance. The performance of full-model fine-tuning is in dotted horizontal line.

Search method	**RTE (acc)**	**SST-2 (acc)**
GDNAS	**80.4** (1.3)	**94.9** (0.3)
GDNAS (single-level)	80.2 (0.9)	94.7 (0.2)
DARTS	79.8 (1.2)	94.5 (0.5)
ENAS	80.3 (1.0)	94.3 (0.4)

Table 6: Experimental results for the ablation study of the search methods.

validate our GDNAS method, we conduct experiments on the widely studied neural architecture search benchmark dataset, CIFAR-10 (Krizhevsky, 2009). The results are reported in Table 8 of Appendix C. Our GDNAS method achieves 2.52% test error, which manageable search cost of 0.6 GPU days.

6 Conclusion

In this work, we propose the Learned Adapter framework, which automatically optimizes the adapter architectures. First, we design a unified search space for adapters, taking into account the recent works of manual adapter designs. Second, in light of the issues in the DARTS method, we propose a novel GDNAS method that can deliver better adapter architectures and requires no re-training of the learned adapter architectures. We run extensive experiments and analyses on the GLUE benchmark, demonstrating that our Learned Adapter framework can achieve better tuning performances than the baselines while maintaining parameter efficiency.

Limitations

We showed that our proposed method can greatly improve the performance of parameter efficient tuning on diverse NLU tasks and three different pretrained models (i.e., RoBERTa-large, DeBERTa-large and GPT2-large). However, we acknowledge the following limitations: (a) the more super-sized pretrained models with tens of billions of or more parameters were not studied due to limited computation resources. (b) Other tasks in natural language processing, like the text generation tasks, were also not considered. But our framework can be easily transferred to other backbone architectures and different types of tasks. It would be of interest to investigate if the superiority of our method holds for other backbone models and types of tasks. And we will explore it in future work.

Ethics Statement

The finding and proposed method aims to improve the adapter based tuning in terms of tuning parameters and performances. The used datasets are widely used in previous work and, to our knowledge, do not have any attached privacy or ethical issues.

References

Abien Fred Agarap. 2018. Deep learning using rectified linear units (relu). *ArXiv*, abs/1803.08375.

Elad Ben-Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *ArXiv*, abs/2106.10199.

Kaifeng Bi, Changping Hu, Lingxi Xie, Xin Chen, Longhui Wei, and Qi Tian. 2019. Stabilizing DARTS with Amended Gradient Estimation on Architectural Parameters. *arXiv e-prints*, page arXiv:1910.11831.

Kaifeng Bi, Lingxi Xie, Xin Chen, Longhui Wei, and Qi Tian. 2020. Gold-nas: Gradual, one-level, differentiable. *ArXiv*, abs/2007.03331.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. 2018. Efficient architecture search by network transformation. In *AAAI*.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. 2021. Progressive darts: Bridging the optimization gap for nas in the wild. *ArXiv*, abs/1912.10952.

Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. 2021. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 12219–12228.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. 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, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

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

Xiangxiang Gao, Wei Zhu, Jiasheng Gao, and Congrui Yin. 2023. F-pabee: Flexible-patience-based early exiting for single-label and multi-label text classification tasks. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE.

Yuan Gao, Haoping Bai, Zequn Jie, Jiayi Ma, Kui Jia, and Wei Liu. 2020. Mtl-nas: Task-agnostic neural architecture search towards general-purpose multitask learning. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11540–11549.

Demi Guo, Alexander Rush, and Yoon Kim. 2021a. Parameter-efficient transfer learning with diff pruning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4884–4896, Online. Association for Computational Linguistics.

Zhao Guo, Yuan Ni, Keqiang Wang, Wei Zhu, and Guotong Xie. 2021b. Global attention decoder for Chinese spelling error correction. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1419–1428, Online. Association for Computational Linguistics.

Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Liang Zhang, Wentao Han, Minlie Huang, Qin Jin, Yanyan Lan, Yang Liu, Zhiyuan Liu, Zhiwu Lu, Xipeng Qiu, Ruihua Song, Jie Tang, Ji rong Wen, Jinhui Yuan, Wayne Xin Zhao, and Jun Zhu. 2021. Pre-trained models: Past, present and future. *ArXiv*, abs/2106.07139.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *ArXiv*, abs/2110.04366.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced bert with disentangled attention. *ArXiv*, abs/2006.03654.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv: Learning*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Shibo Jie and Zhifang Deng. 2022. Convolutional bypasses are better vision transformer adapters. *ArXiv*, abs/2207.07039.

A. Krizhevsky. 2009. Learning multiple layers of features from tiny images.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84 – 90.

Tuan Le, Marco Bertolini, Frank No'e, and Djork-Arné Clevert. 2021. Parameterized hypercomplex graph neural networks for graph classification. In *International Conference on Artificial Neural Networks*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Hector J. Levesque, Ernest Davis, and L. Morgenstern. 2011. The winograd schema challenge. In *International Conference on Principles of Knowledge Representation and Reasoning*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Xiepeng Li, Zhexi Zhang, Wei Zhu, Zheng Li, Yuan Ni, Peng Gao, Junchi Yan, and Guotong Xie. 2019. Pingan smart health and SJTU at COIN - shared task: utilizing pre-trained language models and commonsense knowledge in machine reading tasks. In *Proceedings of the First Workshop on Commonsense Inference in Natural Language Processing*, pages 93–98, Hong Kong, China. Association for Computational Linguistics.

Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. 2021. Pruning and Quantization for Deep Neural Network Acceleration: A Survey. *arXiv e-prints*, page arXiv:2101.09671.

Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Loddon Yuille, Jonathan Huang, and Kevin P. Murphy. 2018. Progressive neural architecture search. In *ECCV*.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019a. Darts: Differentiable architecture search. *ArXiv*, abs/1806.09055.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2021. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *ArXiv*, abs/2110.07602.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Annual Meeting of the Association for Computational Linguistics*.

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

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

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. In *NeurIPS*.

Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Wen tau Yih, and Madian Khabsa. 2021. Unipelt: A unified framework for parameter-efficient language model tuning. *ArXiv*, abs/2110.07577.

Nafise Sadat Moosavi, Quentin Delfosse, Kristian Kersting, and Iryna Gurevych. 2022. Adaptable adapters. In *North American Chapter of the Association for Computational Linguistics*.

Niv Nayman, Asaf Noy, T. Ridnik, Itamar Friedman, Rong Jin, and Lihi Zelnik-Manor. 2019. Xnas: Neural architecture search with expert advice. *ArXiv*, abs/1906.08031.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *North American Chapter of the Association for Computational Linguistics*.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online. Association for Computational Linguistics.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient neural architecture search via parameter sharing. In *ICML*.

Alec Radford and Karthik Narasimhan. 2018. Improving language understanding by generative pretraining.

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

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

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. 2017. Swish: a self-gated activation function. *arXiv: Neural and Evolutionary Computing*.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. 2018. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence*.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2020. Adapterdrop: On the efficiency of adapters in transformers. In *Conference on Empirical Methods in Natural Language Processing*.

David R. So, Chen Liang, and Quoc V. Le. 2019. The evolved transformer. *ArXiv*, abs/1901.11117.

Haixia Sun, Jin Xiao, Wei Zhu, Yilong He, Sheng Zhang, Xiaowei Xu, Li Hou, Jiao Li, Yuan Ni, and Guotong Xie. 2020. Medical knowledge graph to enhance fraud, waste, and abuse detection on claim data: Model development and performance evaluation. *JMIR Med Inform*, 8(7):e17653.

Tianxiang Sun, Xiangyang Liu, Wei Zhu, Zhichao Geng, Lingling Wu, Yilong He, Yuan Ni, Guotong Xie, Xuanjing Huang, and Xipeng Qiu. 2022. A simple hash-based early exiting approach for language understanding and generation. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2409–2421, Dublin, Ireland. Association for Computational Linguistics.

Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. 2022. Lst: Ladder side-tuning for parameter and memory efficient transfer learning. *ArXiv*, abs/2206.06522.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *ArXiv*, abs/1706.03762.

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 *BlackboxNLP@EMNLP*.

Yujing Wang, Yaming Yang, Yiren Chen, Jing Bai, Ce Zhang, Guinan Su, Xiaoyu Kou, Yunhai Tong, Mao Yang, and Lidong Zhou. 2020. Textnas: A neural architecture search space tailored for text representation. In *AAAI*.

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

Zhuofeng Wu, Sinong Wang, Jiatao Gu, Rui Hou, Yuxiao Dong, V. G. Vinod Vydiswaran, and Hao Ma. 2022. Idpg: An instance-dependent prompt generation method. In *North American Chapter of the Association for Computational Linguistics*.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. 2018. Snas: Stochastic neural architecture search. *ArXiv*, abs/1812.09926.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. 2019. Snas: Stochastic neural architecture search. *ArXiv*, abs/1812.09926.

Yuhui Xu, Lingxi Xie, Wenrui Dai, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Hongkai Xiong, and Qi Tian. 2021. Partially-connected neural architecture search for reduced computational redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43:2953–2970.

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. 2019. Understanding and robustifying differentiable architecture search. *ArXiv*, abs/1909.09656.

Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q. Weinberger, and Yoav Artzi. 2020. Revisiting few-sample bert fine-tuning. *ArXiv*, abs/2006.05987.

Zhen Zhang, Wei Zhu, Jinfan Zhang, Peng Wang, Rize Jin, and Tae-Sun Chung. 2022. PCEE-BERT: Accelerating BERT inference via patient and confident early exiting. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 327–338, Seattle, United States. Association for Computational Linguistics.

Zhexi Zhang, Wei Zhu, Junchi Yan, Peng Gao, and Guotong Xie. 2021. Automatic student network search for knowledge distillation. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 2446–2453. IEEE.

Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. 2019. Bayesnas: A bayesian approach for neural architecture search. *ArXiv*, abs/1905.04919.

Wei Zhu. 2021a. Autonlu: Architecture search for sentence and cross-sentence attention modeling with redesigned search space. In *Natural Language Processing and Chinese Computing: 10th CCF International Conference, NLPCC 2021, Qingdao, China, October 13–17, 2021, Proceedings, Part I 10*, pages 155–168. Springer.

Wei Zhu. 2021b. AutoRC: Improving BERT based relation classification models via architecture search. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: Student Research Workshop*, pages 33–43, Online. Association for Computational Linguistics.

Wei Zhu. 2021c. LeeBERT: Learned early exit for BERT with cross-level optimization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2968–2980, Online. Association for Computational Linguistics.

Wei Zhu, Yilong He, Ling Chai, Yunxiao Fan, Yuan Ni, Guotong Xie, and Xiaoling Wang. 2021a. paht_nlp @ MEDIQA 2021: Multi-grained query focused multi-answer summarization. In *Proceedings of the 20th Workshop on Biomedical Language Processing*, pages 96–102, Online. Association for Computational Linguistics.

Wei Zhu, Yuan Ni, Xiaoling Wang, and Guotong Xie. 2021b. Discovering better model architectures for medical query understanding. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Papers*, pages 230–237, Online. Association for Computational Linguistics.

Wei Zhu, Peng Wang, Xiaoling Wang, Yuan Ni, and Guotong Xie. 2023. Acf: Aligned contrastive finetuning for language and vision tasks. In *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5.

Category	Datasets	|train|	|dev|	|test|	$\|\mathcal{Y}\|$	Type	Labels
Single-sentence	SST-2	66349	1000	872	2	sentiment	positive, negative
	CoLA	8551	521	522	2	linguistic acceptability	acceptable, not acceptable
Sentence-pair	MNLI	391702	1000	19647	3	NLI	entailment, neutral, contradiction
	MRPC	2668	1000	408	2	paraphrase	equivalent, not equivalent
	QNLI	103743	1000	5463	2	NLI	entailment, not entailment
	QQP	362846	1000	40430	2	paraphrase	equivalent, not equivalent
	RTE	1490	1000	277	2	NLI	entailment, not entailment
	STS-B	5750	750	751	-	semantic similarity	-

Table 7: The dataset statistics of the GLUE benchmark tasks evaluated in this work. For MNLI task, the number of samples in the test set is summed by matched and mismatched samples. $|\mathcal{Y}|$ is the number of classes for a classification task.

Wei Zhu, Xiaoling Wang, Yuan Ni, and Guo Tong Xie. 2021c. Gaml-bert: Improving bert early exiting by gradient aligned mutual learning. In *EMNLP*.

Wei Zhu, Xiaoling Wang, Yuan Ni, and Guotong Xie. 2021d. Autotrans: Automating transformer design via reinforced architecture search. In *Natural Language Processing and Chinese Computing*, pages 169–182, Cham. Springer International Publishing.

Wei Zhu, Xiaofeng Zhou, Keqiang Wang, Xun Luo, Xiepeng Li, Yuan Ni, and Guotong Xie. 2019. PANLP at MEDIQA 2019: Pre-trained language models, transfer learning and knowledge distillation. In *Proceedings of the 18th BioNLP Workshop and Shared Task*, pages 380–388, Florence, Italy. Association for Computational Linguistics.

Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning. *ArXiv*, abs/1611.01578.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2017. Learning transferable architectures for scalable image recognition. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8697–8710.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2018. Learning transferable architectures for scalable image recognition. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8697–8710.

Yuhui Zuo, Wei Zhu, and Guoyong GUET Cai. 2022. Continually detection, rapidly react: Unseen rumors detection based on continual prompt-tuning. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 3029–3041, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

A Datasets and evaluation metrics

A.1 Dataset splits

The detailed statistics of the GLUE benchmark is presented in Table 7.

A.2 Evaluation metrics

For MNLI, we report the average of the accuracy scores on the matched and mis-matched test set. For MRPC and QQP, we report acc-f1, which is the average of accuracy and F1 scores. For STS-B, we report corr, which denotes the average of the Pearson and Spearman correlation coefficients. For CoLA, we report mcc, which is the Matthews correlation. For all other tasks, we report accuracy.

B Appendix for experimental settings

Details of experiments We run our methods and baseline method on the GLUE (Wang et al., 2018) following the previous works. All datasets are downloaded via the HuggingFace Datasets[5] library. Since the test split of these tasks are invisible to the researchers, we randomly split off 1k samples from the training set as validation set for large datasets(QQP, QNLI, SST2, MNLI), and use the remaining as the training set, and use the original validation set as the test set. For other datasets, we randomly split the original validation set in half as the validation set and the test set, and use the original train set as our train set. The same dataset is split differently with different random seeds.

For each experiment setting, we repeat the experiment with 5 seeds, and report the average score and standard deviation. In all experiments, the maximum sequence length is 128 for the tasks. We mainly use the RoBERTa-large model (355M parameters) as the backbone model, but we also adopt the GPT2-large and DeBERTa-large the backbone models for ablation studies. We freeze the pre-trained parameters in all experiments except full-model finetuning. We use AdamW as the optimizer with a linear learning rate decay schedule and 6% of the training steps for warm-up.

[5]https://huggingface.co/docs/datasets/index

We train the hyper-network on the train set D_{train} following (Liu et al., 2019a). For training epochs, we set $K_1 = 5$ and $K_2 = 1$ on large datasets (SST-2, QNLI, QQP and MNLI), and $K_1 = 20$ and $K_2 = 5$ on low-resource datasets. We will run the search procedure once for each task.

After the hyper-network is fully discretized, instead of retraining from scratch, we further train the remained network for K_2 epochs, and we evaluate the model on the dev set and save the model checkpoint every I_{eval} steps. The best checkpoint on the dev set is used to run predictions on the test set. We report the average scores on the test set and standard deviations across 5 random seeds.

Other hyper-parameters We do pilot experiments on SST-2 using learning rates in {2e-5, 5e-5, 1e-4, 2e-4}, and find that 1e-4 performs the best. For fine-tuning, we try learning rates in {1e-5, 2e-5, 5e-5} and find that 2e-5 performs the best. The number of training epochs for the baselines is set as $K = 5$ on large datasets (SST-2, QNLI, QQP and MNLI), and $K = 20$ on smaller datasets. We apply these hyper-parameters to all baselines, and no further hyperparameter-tuning are conducted. Therefore, the comparison is fair for all methods.

C Experimental results on the CIFAR-10 task

To further validate that our GDNAS method can obtain better search performances than the DNAS baselines, we now conduct experiments in the general NAS setting. Following DARTS (Liu et al., 2019a), we conduct neural architecture search on the CIFAR-10 dataset (Krizhevsky, 2009) based on the search space of DARTS. We keep all the search settings identical to DARTS. We first train the hyper-network with frozen architectural weights for 50 epochs. After selecting the operation on an edge, we tune the hyper-net for 8 epochs to let the modified hyper-network to adjust. Following DARTS (Liu et al., 2019a), we run the search and architecture selection phase with four random seeds and report both the best and average test errors of the obtained architectures. The results are reported in Table 8, which compares GDNAS with the DNAS baseline methods. Our GDNAS method achieves 2.52% test error, which manageable search cost of 0.6 GPU days. The results of GDNAS is comparable to other method (like P-DARTS (Chen et al., 2021)) with more complex procedures.

D Learned architectures on the GLUE tasks

In this section, we present the learned adapter architectures on the RTE and SST-2 tasks. The learned adapter architecture are presented in Table 9, 10, respectively.

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	Search Framework
NASNet-A (Zoph et al., 2017)	2.65	3.3	2000	RL
AmoebaNet (Real et al., 2018)	3.34	3.2	3150	evolution
ENAS (Pham et al., 2018)	2.89	4.6	0.5	RL
DARTS (Liu et al., 2019a)	3.02	3.3	0.4	DNAS
SNAS (Xie et al., 2018)	2.85	2.8	1.5	DNAS
BayesNAS (Zhou et al., 2019)	2.81	3.4	0.2	DNAS
P-DARTS (Chen et al., 2021)	2.50	3.4	0.3	DNAS
R-DARTS (Zela et al., 2019)	2.95	-	1.6	DNAS
GDNAS (ours)	2.52	3.3	0.6	DNAS

Table 8: The search results on the CIFAR-10 task.

Layer index	Adapter placement	Activation g_1	Activation g_2	Encoder operation 1
1	FFN	elu	null_act	mha_8
2	-	-	-	-
3	-	-	-	-
4	FFN	elu	null_act	mha_2
5	Block	tanh	null_act	mha_2
6	FFN	gelu_new	leaky_relu	conv_3
7	FFN	null_act	tanh	mha_2
8	-	-	-	-
9	Attn	elu	relu	conv_3
10	-	-	-	-
11	Attn	gelu_new	relu	conv_1
12	Attn	gelu_new	relu	conv_3
13	Block	relu	leaky_relu	conv_1
14	Attn	swish	relu	conv_3
15	FFN	leaky_relu	relu	conv_3
16	Block	leaky_relu	relu	conv_5
17	FFN	leaky_relu	relu	conv_1
18	Attn	leaky_relu	null_act	skip_connect
19	FFN	relu	relu	conv_3
20	FFN	gelu_new	null_act	conv_3
21	-	-	-	-
22	Block	tanh	null_act	mha_8
23	Attn	tanh	null_act	conv_3
24	FFN	tanh	null_act	mha_2

Table 9: The learned adapter architectures on the RTE task when the PTM backbone is RoBERTa-large. If an adapter's architecture contains only "-", it means our Learned Adapter framework choose the **null** encoder operation, and equivalently, dropping this layer's adapter.

Layer index	Adapter placement	Activation g_1	Activation g_2	Encoder operation 1
1	Attn	null_act	null_act	conv_5
2	-	-	-	-
3	FFN	elu	null_act	mha_2
4	Attn	tanh	null_act	mha_2
5	Attn	elu	null_act	skip_connect
6	FFN	gelu_new	relu	conv_5
7	Block	leaky_relu	leaky_relu	conv_3
8	-	-	-	-
9	Block	leaky_relu	leaky_relu	conv_5
10	FFN	relu	relu	conv_3
11	Block	tanh	null_act	mha_8
12	-	-	-	-
13	Block	tanh	null_act	skip_connect
14	-	-	-	-
15	FFN	swish	swish	conv_1
16	Block	relu	relu	conv_1
17	FFN	relu	swish	conv_3
18	FFN	leaky_relu	relu	conv_3
19	Attn	gelu_new	relu	mha_8
20	FFN	swish	relu	mha_8
21	Attn	swish	null_act	conv_3
22	Attn	elu	null_act	conv_3
23	Attn	null_act	null_act	mha_8
24	Attn	elu	null_act	mha_2

Table 10: The learned adapter architectures on the SST-2 task when the PTM backbone is RoBERTa-large. If an adapter's architecture contains only "-", it means our Learned Adapter framework choose the **null** encoder operation, and equivalently, dropping this layer's adapter.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Left blank.

☑ A2. Did you discuss any potential risks of your work?
Left blank.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Left blank.

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☒ Did you use or create scientific artifacts?

Left blank.

☐ B1. Did you cite the creators of artifacts you used?
No response.

☐ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
No response.

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
No response.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
No response.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
No response.

☐ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
No response.

C ☑ Did you run computational experiments?

Section 6

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Section 6.2

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Section 6.2, Appendix C

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Section 6.4, Section 6.5, Section 6.6

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Section 6.2

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?
Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.