# Neural Architecture Search for Parameter-Efficient Fine-tuning of Large Pre-trained Language Models

**Neal Lawton**[1*] **Anoop Kumar**[2] **Govind Thattai**[2] **Aram Galstyan**[2] **Greg Ver Steeg**[2]
[1]Information Sciences Institute [2]Amazon Alexa AI
lawton@isi.edu {anooamzn,thattg,argalsty,gssteeg}@amazon.com

## Abstract

Parameter-efficient tuning (PET) methods fit pre-trained language models (PLMs) to downstream tasks by either computing a small compressed update for a subset of model parameters, or appending and fine-tuning a small number of new model parameters to the pre-trained network. Hand-designed PET architectures from the literature perform well in practice, but have the potential to be improved via automated neural architecture search (NAS). We propose an efficient NAS method for learning PET architectures via structured and unstructured pruning. We present experiments on GLUE demonstrating the effectiveness of our algorithm and discuss how PET architectural design choices affect performance in practice.

## 1 Introduction

Fine-tuning a large pre-trained language model is a popular method for solving many downstream natural language processing (NLP) tasks. *Full fine-tuning* involves fine-tuning all parameters of the base PLM, resulting in a fine-tuned copy of the model. However, full fine-tuning becomes cumbersome when fine-tuning on multiple downstream tasks due to the massive size of state-of-the-art language models, which range from the millions (Devlin et al., 2018; Liu et al., 2019) to billions (Brown et al., 2020) and now trillions (Fedus et al., 2022) of parameters. Full fine-tuning also carries a risk of *catastrophic forgetting* (Jang et al., 2021; Chen et al., 2022), wherein the PLM's learned useful representation of natural language data is forgotten during fine-tuning.

To address those problems, recent research has focused on *parameter-efficient tuning* (PET). Rather than fine-tuning all parameters of the base PLM, PET methods choose a small subset of parameters to fine-tune (Zaken et al., 2021; Guo et al., 2020), or compute compressed parameter updates (Hu et al., 2021; Mahabadi et al., 2021), or append and fine-tune a small subset of new parameters (Houlsby et al., 2019; Li and Liang, 2021; Hambardzumyan et al., 2021; He et al., 2021). Each of these methods has their own advantages and disadvantages, but one question relevant to all these methods is *which parts of the network are most efficient to fine-tune, and what is the most parameter-efficient way to fine-tune them*?

Here we answer this question by designing and applying a fine-grain NAS method for learning PET architectures. Our method uses a first order approximation of the loss function and is computationally efficient. We compare our approach with several hand-designed PET methods and find that the architectures learned by our method generally achieve comparable or higher development set performance on GLUE tasks (Wang et al., 2018) for the same number of parameters. We conclude by examining the PET architectures learned by our method and discuss the affect of architecture design choices on parameter efficiency.

## 2 Related work

Many different PET methods exist in the literature. *Adapter networks* insert and fine-tune small adapter modules to a base PLM. Rebuffi et al. (2017) introduced adapter networks to the visual domain, and Houlsby et al. (2019) introduced adapters to transformers. Adapters have been applied to text generation (Lin et al., 2020), translation (Bapna et al., 2019), and multi-task learning (Pfeiffer et al., 2020c,a). Peters et al. (2019) compare adaptation with full fine-tuning. AdapterHub (Pfeiffer et al., 2020b) enables easy sharing of adapter models. Additionally, Mosbach et al. (2020) propose best practices for producing strong full fine-tuning baselines.

Prompt-tuning methods fine-tune a PLM by inserting prompt tokens into the input sequence. Continuous prompts (Li and Liang, 2021; Lester

[*]Work done while at Amazon Alexa AI



*Findings of the Association for Computational Linguistics: ACL 2023*, pages 8506–8515

et al., 2021; Hambardzumyan et al., 2021) or discrete prompts (Shin et al., 2020) can be learned or engineered (Brown et al., 2020). Gu et al. (2021) demonstrate the effectiveness of pre-training prompts for low resource tasks.

Some methods fine-tune a subset of parameters (Zaken et al., 2021; Guo et al., 2020), or compute compressed parameter updates (Hu et al., 2021; Mahabadi et al., 2021). These methods fine-tune the PLM without increasing test-time inference latency. He et al. (2021) and Mao et al. (2021) combine multiple PET methods.

Beyond parameter-efficient tuning, NAS has previously been used to discover more parameter-efficient base language models. Cheong and Daniel (2019) use magnitude pruning to reduce the number of parameters in BERT. Many efforts at pruning BERT have focused on pruning attention heads from the multi-head attention (MHA) modules (Michel et al., 2019; Voita et al., 2019; Li et al., 2021). Sajjad et al. (2020) evaluate different ad-hoc strategies for shrinking the depth of a BERT encoder. So et al. (2019) use an evolutionary NAS method to learn an improved transformer cell. In contrast to NAS, distillation can be used to compress language models (Sanh et al., 2019; Jiao et al., 2019; Sun et al., 2020).

In our experiments section, we examine the architectures learned by our algorithm and consider what they say about which parts of the network are most parameter-efficient to fine-tune. Merchant et al. (2020) explore a similar question, probing the network activations to understand how the network's representation of natural language data changes during full fine-tuning.

## 3 Method

The architecture search space we choose for our NAS method is based on BitFit (Zaken et al., 2021) and LoRA (Hu et al., 2021), two of the most popular methods for parameter-efficient fine-tuning in the literature. We consider both structured and unstructured variants of each of these, where the non-zero pattern of the learned PET parameters is restricted or unrestricted, respectively. Specifically, our search space consists of the following:

1. Learning an update $\Delta b$ for each vector of bias parameters $b$. In *structured bias-tuning*, for each PLM module, the NAS algorithm must choose whether $\Delta b = 0$ or not. In *unstructured bias-tuning*, for each PLM module, the NAS algorithm must choose which components of $\Delta b$ should be zero or non-zero.

2. Learning a low-rank (LoRA Hu et al., 2021) update $\Delta W = UV^\top$ for each user-specified parameter matrix $W$. The maximum possible rank for the update is also user-specified. In *structured LoRA*, for each parameter matrix $W$, the NAS algorithm must decide what the rank of the update $UV^\top$ should be. In *unstructured LoRA*, the NAS algorithm must decide which components of $U$ and $V$ should be non-zero.

The collection of updates $\Delta b$ and $\Delta W$ are the PET parameters. In this search space, any number of the above PET modules can be applied to a base PLM without increasing the latency of inference, just like BitFit (Zaken et al., 2021) and LoRA (Hu et al., 2021).

### 3.1 Pruning

We perform NAS via pruning. Our NAS method begins by training a PET architecture of a maximum user-specified size: for each bias tuning module, we fine-tune all bias parameters, and for each LoRA update module, we learn a dense low-rank update with a user-specified rank (in all our experiments, we use rank-16 initial LoRA updates). After training the initial PET architecture, our method decides which PET parameters to prune and which to keep. Then we re-initialize and re-train the pruned architecture before evaluating on the validation set.

The criteria that we use to decide which PET parameters to prune is based on a first-order approximation of the change in training loss that results from pruning a PET parameter $\theta$:

$$-\theta \cdot \frac{\partial \mathcal{L}}{\partial \theta}.$$

Note that this is a common pruning criterion, e.g., see Molchanov et al. (2016). This criterion is straight forward to use when deciding whether to prune a single PET parameter, as in unstructured bias-tuning and unstructured LoRA. For structured bias-tuning, we sum this criterion over the entire bias update $\Delta b$, and for structured LoRA, when considering what column of $U$ and $V$ to prune, we sum the criterion over each column of $U$.

Pruning via evaluating the criterion at the end of training does not yield better-than-random architectures. We observe that the value of the pruning

| Method | #params | MNLI | SST-2 | MRPC | CoLA | QNLI | QQP | RTE | STS-B | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|
| FFT | 355M | 90.6 | 96.0 | 89.2 | 66.8 | 94.6 | 91.6 | 85.2 | 91.5 | 88.2 |
| BitFit | 273k | 89.2 | 95.6 | 88.2 | 65.0 | 93.9 | 88.1 | 81.9 | 91.4 | 86.7 |
| Adapters† | 3.0M | 90.2 | 96.1 | 90.2 | 68.3 | 94.8 | 91.9 | 83.8 | 92.1 | 88.4 |
| LoRA | 3.4M | 90.7 | 95.3 | 89.7 | 65.1 | 93.8 | 90.3 | 84.8 | 91.7 | 87.7 |
| MaM | 3.4M | **90.6** | 95.3 | 89.7 | 65.1 | 93.8 | 90.3 | 84.8 | 91.7 | 87.7 |
| S-MaM | 3.4M | **90.6** | **95.9** | 90.4 | 66.3 | **94.5** | 90.6 | 85.2 | 91.6 | 88.1 |
| U-MaM | 3.4M | 90.3 | 95.8 | **90.7** | **66.8** | 94.1 | **90.8** | **85.9** | **91.8** | **88.3** |
| WARP† | 25k | 88.2 | **96.0** | **90.8** | 60.6 | **93.5** | 84.5 | **75.8** | 88.6 | **84.8** |
| S-BitFit | 25k | 84.1 | 94.2 | 70.6 | 40.2 | 88.9 | 83.8 | 56.0 | 76.8 | 74.3 |
| U-BitFit | 25k | **88.8** | 95.5 | 85.3 | **62.1** | **93.5** | **87.7** | 74.0 | **90.3** | 84.6 |

Table 1: GLUE development set score for learned and hand-crafted PET architectures. We report the result for WARP† from Hambardzumyan et al. (2021) and for Adapters† from Hu et al. (2021).

criterion may change drastically from one stochastic gradient descent (SGD) step to the next. To maximally smooth the noise introduced by SGD, we instead average the pruning criterion over all training SGD steps. This yields the most consistent indication of which PET parameters are efficient to prune.

Our NAS algorithm takes as input a parameter budget specifying the desired maximum number of parameters in the learned PET architecture. After training the initial PET architecture and evaluating each pruning criterion, we apply each pruning operation in increasing criterion order until the number of parameters in the PET architecture falls below the parameter budget. This way, pruning operations that are estimated to increase the training loss the least are applied first.

### 3.2 Initialization

Correct initialization is important for successfully applying this algorithm. After pruning, we re-initialize and re-train the learned PET architecture before evaluating on the validation set. We find that it is important to use the same initialization after pruning as before. We believe this is a consequence of the lottery ticket hypothesis (Frankle and Carbin, 2018).

We always initialize bias parameter updates as zero, as do other works, and find this works well. However, we find that the initialization for LoRA given in the original paper (Hu et al., 2021), which initializes the matrix $U$ with zeros and $V$ with a Gaussian distribution, is not ammenable to unstructured LoRA pruning. Because the parameters in the matrix $U$ are initialized zero, the magnitudes of those parameters are likely to remain small throughout training relative to the magnitudes of the parameters in $V^\top$. Consequently, the pruning criterion for unstructured LoRA updates is likely to favor pruning parameters from $U$ over $V$, leading to an unbalanced, parameter-inefficient LoRA update. Instead, following the same reasoning given for Kaiming initialization (He et al., 2015), we recommend the following initialization:

$$U \sim \mathcal{N}(0, 1/\sqrt{m}) \quad V \sim \mathcal{N}(0, 1/\sqrt{n}), \quad (1)$$

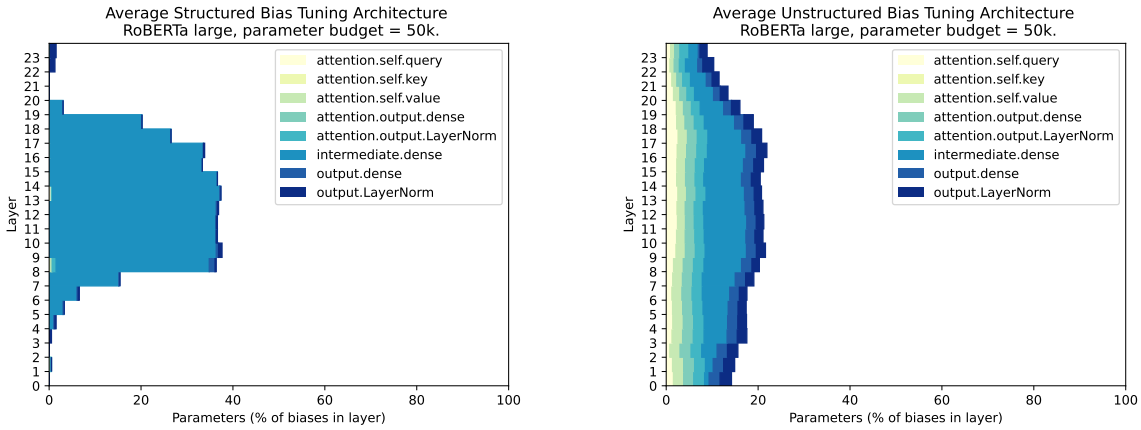
where $m$ is the first dimension of the matrix $U$ (i.e., the "fan-in"), and $n$ is the second dimension of the matrix $V^\top$ (i.e., the "fan-out"). With this initialization, the expected square gradients for the parameters of $U$ and $V$ are equal.

## 4 Experiments

Details of our experimental setup, including hyperparameter choices, are available in the appendix. In all experiments we report median validation score at the end of training over 5 random initializations using the GLUE development set for validation.

### 4.1 Comparing to Full Fine-tuning

Here we present results for training larger PET architectures with the aim of achieving performance similar to full fine-tuning, but with fewer parameters. In addition to structured or unstructured bias-tuning, our learned PET architectures add structured or unstructured LoRA updates to the MHA query modules, key modules, and the `dense` feed forward network (FFN) modules. In Table 1, our learned structured PET architecture is labeled `S-MaM`, and our learned unstructured PET architecture is labeled `U-MaM`. We compare our method with




(a) (b)

Figure 1: Average learned architecture for (a) structured bias-tuning and (b) unstructured bias-tuning.

a LoRA baseline (Hu et al., 2021) and a baseline similar to Mix-and-Match (MaM) (He et al., 2021). Our LoRA baseline fine-tunes all bias parameters and adds rank-8 updates to all MHA query and key modules. Our MaM-like baseline fine-tunes all bias parameters and adds rank-8 updates to all MHA query and key modules and all FFN modules.

Results for this experiment with parameter budget 3.4M are in Table 1. In our `S-MaM` and `U-MaM` experiments, we prune from an initial architecture with 6.8M parameters. We observe that our `S-MaM` architecture achieves slightly higher average GLUE (Wang et al., 2018) validation score over our MaM-like baseline, and our `U-MaM` architecture slightly higher average GLUE validation score over our `S-MaM` architecture. We conclude that structured architecture search provides a small positive benefit over the uniform-rank baseline architecture, and that unstructured architecture search provides a small positive benefit over structured architecture search. We also observe our `U-MaM` architecture achieves average GLUE validation score on par with full fine-tuning while fine-tuning approximately 100 times fewer parameters.

### 4.2 Very Small PETs

Here we examine our learned PET architectures with parameter budget less than the total number of bias parameters in the base PLM. For `roberta-large`, this is about 273k.

We use our method to learn structured and unstructured bias-tuning architectures. We compare our method with WARP (Hambardzumyan et al., 2021) using parameter budget 25k in Table 1, and report results for our method for other parameter budgets in the appendix. Our learned structured and unstructured bias-tuning architectures are labeled `S-BitFit` and `U-BitFit`, respectively. In our `S-BitFit` and `U-BitFit` experiments, we prune from a PET architecture with 273k parameters that fine-tuens all bias parameters, the same as BitFit. We observe that the unstructured bias-tuning architecture achieves significantly higher validation performance than the structured bias-tuning architecture with the same parameter budget. We conclude that the subset of bias parameters that are "good" to fine-tune are not concentrated in a few modules, but rather are distributed throughout the network. Our learned unstructured bias-tuning architecture with $< 50$k parameters fine-tunes only $18\%$ of all bias parameters while achieving validation GLUE score only slightly less than fine-tuning all bias parameters ($86.5$ versus $86.7$). We conclude that a vast majority of bias parameters do not need to be fine-tuned to achieve performance comparable to fine-tuning all bias parameters. With a parameter budget of 25k, unstructured bias tuning achieves similar performance compared to WARP, beating or tying WARP on a majority of GLUE tasks but achieving slightly worse average performance. We conclude that both methods are about equally effective.

### 4.3 Interpreting Learned Architectures

Here we examine the architectures learned by our algorithm and consider what they say about which parts of the network are most parameter-efficient to fine-tune. Each illustration discussed in this section averages the architectures learned by our method over all GLUE tasks and five random initializa-

tions per task. Figure 1a illustrates the architecture learned by our method for structured bias-tuning with parameter budget 50k. We observe a clear preference by our algorithm for fine-tuning the biases of the `intermediate.dense` modules in the middle of the network. Figure 1b illustrates the architecture learned by our method for unstructured bias tuning with parameter budget 50k. We observe a weak preference for fine-tuning the bias parameters of modules in the middle of the network, but not for any particular module type within each transformer block. We conclude that the biases that are most parameter-efficient to fine-tune are in the middle layers of the network.

## 5 Conclusion

In this paper, we considered the question *which parts of the network are most efficient to fine-tune, and what is the most parameter-efficient way to fine-tune them*? To answer that question, we developed a NAS algorithm based on structured and unstructured pruning. We presented experimental results on RoBERTa Large demonstrating the effectiveness of our algorithm, achieving GLUE validation performance similar to WARP at 25k parameters (9% of all biases), similar to BitFit at 50k parameters (18% of all biases), and similar to full fine-tuning at 3.4M parameters (10% of all parameters). From our learned architectures we observed that the bias parameters in the middle layers of the network are most efficient to fine-tune. We conclude that it is important to consider *where* to fine-tune as well as *how*.

## Limitations

Differences in experimental setup may make it difficult to accurately and fairly compare published results. For example, to prevent data leakage, we report validation performance at the end of training and do not perform early stopping. This is in contrast to most other papers which report peak validation performance. Results reported for other methods are reproduced in the same learning environment as our method unless explicitly stated otherwise. This takes into account recent work demonstrating problems with fairly and accurately evaluating PET methods that use early stopping improperly (Chen et al., 2022).

Although many pruning criteria exist in the literature, in this paper we only consider one pruning criterion. Although not presented in this paper, experiments we conducted with various formulations of magnitude pruning did not produce better results.

Although prompt tuning is a popular PET method, we do not perform NAS for prompt tuning to determine the most efficient positions for inserting prompt tokens into the input. Pruning may or may not prove to be a successful strategy for this problem.

Other NAS strategies exist in the literature besides pruning, such as evolutionary, reinforcement learning, and DARTS (Liu et al., 2018). However, our pruning method seems to give a good trade-off between validation performance and computational expense.

## Ethics Statement

Powerful language models can be used for unethical purposes, such as generating offensive or deceptive content. Although researchers today are making a greater effort to establish protections against the unethical use of their models, bad actors may still find ways to circumvent those protections. One avenue for attack could involve fine-tuning a PLM on a nefarious dataset to produce unethical content. In this paper, we showed that a PLM can be successfully fine-tuned on a downstream task by fine-tuning a small number of parameters, or adding a low-rank update to a few select parameter matrices. Thus researchers should consider the risk posed by unethical parameter-efficient fine-tuning before publishing a fine-tuneable version of their model.

## References


Ankur Bapna, Naveen Arivazhagan, and Orhan Firat. 2019. Simple, scalable adaptation for neural machine translation. *arXiv preprint arXiv:1909.08478*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Guanzheng Chen, Fangyu Liu, Zaiqiao Meng, and Shangsong Liang. 2022. Revisiting parameter-efficient tuning: Are we really there yet? *arXiv preprint arXiv:2202.07962*.

Robin Cheong and Robel Daniel. 2019. transformers. zip: Compressing transformers with pruning and quantization. *Technical report, tech. rep., Stanford University, Stanford, California*.

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

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270.

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

Yuxian Gu, Xu Han, Zhiyuan Liu, and Minlie Huang. 2021. PPT: Pre-trained prompt tuning for few-shot learning. *arXiv preprint arXiv:2109.04332*.

Demi Guo, Alexander M Rush, and Yoon Kim. 2020. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv:2012.07463*.

Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. 2021. WARP: Word-level adversarial reprogramming. *arXiv preprint arXiv:2101.00121*.

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

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

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*.

Joel Jang, Seonghyeon Ye, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun Kim, Stanley Jungkyu Choi, and Minjoon Seo. 2021. Towards continual knowledge learning of language models. *arXiv preprint arXiv:2110.03215*.

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

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

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

Jiaoda Li, Ryan Cotterell, and Mrinmaya Sachan. 2021. Differentiable subset pruning of transformer heads. *Transactions of the Association for Computational Linguistics*, 9:1442–1459.

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

Zhaojiang Lin, Andrea Madotto, and Pascale Fung. 2020. Exploring versatile generative language model via parameter-efficient transfer learning. *arXiv preprint arXiv:2004.03829*.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. DARTS: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

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*.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. *arXiv preprint arXiv:2106.04647*.

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 preprint arXiv:2110.07577*.

Amil Merchant, Elahe Rahimtoroghi, Ellie Pavlick, and Ian Tenney. 2020. What happens to BERT embeddings during fine-tuning? *arXiv preprint arXiv:2004.14448*.

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

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*.

Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. 2020. On the stability of fine-tuning BERT: Misconceptions, explanations, and strong baselines. *arXiv preprint arXiv:2006.04884*.

Matthew E Peters, Sebastian Ruder, and Noah A Smith. 2019. To tune or not to tune? Adapting pretrained representations to diverse tasks. *arXiv preprint arXiv:1903.05987*.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2020a. AdapterFusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020b. AdapterHub: A framework for adapting transformers. *arXiv preprint arXiv:2007.07779*.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020c. MAD-X: An adapter-based framework for multi-task cross-lingual transfer. *arXiv preprint arXiv:2005.00052*.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2017. Learning multiple visual domains with residual adapters. *Advances in neural information processing systems*, 30.

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

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

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Eliciting knowledge from language models with automatically generated prompts. *arXiv preprint arXiv:2010.15980*.

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

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

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

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. *arXiv preprint arXiv:1804.07461*.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

## A Experiment Setup

In all experiments we use the Adam optimizer (Kingma and Ba, 2014) and a linear learning rate scheduler with $6\%$ warm-up steps. We observe that training with a higher peak learning rate works best when fine-tuning a small number of parameters. We use different peak learning rates for different experiments depending on the maximum number of parameters being fine-tuned, ranging from $10^{-5}$ for full fine-tuning to $3 \times 10^{-4}$ for training our smallest PETs. We also train for a different number of epochs for each GLUE tasks. We train for 20 epochs on MRPC, RTE, CoLA, and STSB; 5 epochs on SST-2 and QNLI; and 2 epochs for MNLI and QQP. We observe that extending the number of training epochs beyond these limits does not substantially affect validation performance. In all experiments, we use batch size 16 and maximum sequence length 128.

## B Additional Experimental Results

We report results for our learned structured and unstructured bias-tuning architecture with parameter budgets 10k, 25k, 50k, 100k, and 200k in Table 2. We observe that unstructured bias-tuning holds an advantage over structured bias-tuning across all parameter budgets. We also observe that the performance of unstructured bias-tuning begins to fall off after decreasing the parameter budget below 50k. WARP with a parameter budget of 11k significantly outperforms our `U-BitFit` method with a parameter budget of 10k on the MRPC and COLA tasks. This difference might be explained by the difference in experimental setup (e.g., Hambardzumyan et al. (2021) reports peak validation score whereas we report end-of-training validation score), or the small difference in parameter budget. We believe that our method can be improved in the very small parameter budget regime using iterative, rather than one-shot, pruning.

| Method | #params | MNLI | SST-2 | MRPC | CoLA | QNLI | QQP | RTE | STS-B | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|
| WARP† | 11k | 87.6 | 93.0 | 83.8 | 72.9 | 95.4 | 85.6 | 57.4 | 81.0 | 82.1 |
| WARP† | 25k | 88.2 | 96.0 | 90.8 | 60.6 | 93.5 | 84.5 | 75.8 | 88.6 | 84.8 |
| S-BitFit | 10k | 70.1 | 92.1 | 70.6 | 0.0 | 73.1 | 73.3 | 52.7 | 22.2 | 56.8 |
| S-BitFit | 25k | 84.1 | 94.2 | 70.6 | 40.2 | 88.9 | 83.8 | 56.0 | 76.8 | 74.3 |
| S-BitFit | 50k | 87.1 | 94.3 | 72.1 | 51.5 | 91.4 | 86.2 | 59.6 | 86.9 | 78.6 |
| S-BitFit | 100k | 88.2 | 95.0 | 87.7 | 58.8 | 92.4 | 87.4 | 78.7 | 90.4 | 84.8 |
| S-BitFit | 200k | 89.1 | 95.6 | 88.2 | 63.1 | 93.8 | 87.9 | 81.9 | 91.4 | 86.4 |
| U-BitFit | 10k | 87.4 | 95.1 | 71.1 | 58.8 | 92.2 | 86.3 | 59.6 | 88.3 | 79.8 |
| U-BitFit | 25k | 88.8 | 95.5 | 85.3 | 62.1 | 93.5 | 87.7 | 74.0 | 90.3 | 84.6 |
| U-BitFit | 50k | 89.1 | 95.8 | 88.5 | 64.8 | 93.8 | 88.0 | 80.9 | 91.1 | 86.5 |
| U-BitFit | 100k | 89.3 | 95.8 | 88.5 | 63.6 | 93.9 | 87.7 | 81.9 | 91.3 | 86.5 |
| U-BitFit | 200k | 89.4 | 95.6 | 88.5 | 64.8 | 93.9 | 86.5 | 81.9 | 91.4 | 86.5 |

Table 2: GLUE development set score for structured (S-BitFit) and unstructured (U-BitFit) bias-tuning architectures learned by our method for different parameter budgets. The results for WARP† are reported from Hambardzumyan et al. (2021).

## ACL 2023 Responsible NLP Checklist

### A For every submission:

☑ A1. Did you describe the limitations of your work?
*This is discussed in the section titled "Limitations" after section 5.*

☑ A2. Did you discuss any potential risks of your work?
*We provide an Ethics Statement after section 5.*

☑ A3. Do the abstract and introduction summarize the paper's main claims?
*The abstract is presented before section 1.*

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

### B ☑ Did you use or create scientific artifacts?

*We used the GLUE datasets in our experiments discussed in section 4.*

☑ B1. Did you cite the creators of artifacts you used?
*We provide the citation for GLUE on line 227 in section 4.*

☒ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
*We did not discuss license for GLUE due to space constraints.*

☒ 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)?
*We did not discuss the intended use for GLUE as we properly use GLUE for its intended use and because GLUE is a widely known dataset.*

☒ 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?
*We did not discuss whether GLUE contains any non-anonymized or offensive data because GLUE is a widely known dataset.*

☒ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
*We did not provide documentation for GLUE because GLUE is a widely known dataset.*

☒ 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.
*We did not present train/test/dev split counts for GLUE because of the tight space constraint and because we used the default train/test/dev split for each GLUE task.*

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

**C ☑ Did you run computational experiments?**

*Section 4*

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
*We reported the number of parameters used, but not the computational budget or the computing infrastructure.*

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
*We discuss experimental setup in the appendix.*

☑ 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?
*At the beginning of section 4, we specify that we report the median of 5 runs.*

☑ 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.)?
*We specify the hugging face model that we use and specific modules within that model in section 4.*

**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.?
*Not applicable. Left blank.*

☐ 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)?
*Not applicable. Left blank.*

☐ 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?
*Not applicable. Left blank.*

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
*Not applicable. Left blank.*

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
*Not applicable. Left blank.*