AutoPEFT: Automatic Configuration Search for Parameter-Efficient Fine-Tuning

Han Zhou[1,*] **Xingchen Wan**[2,*] **Ivan Vulić**[1] **Anna Korhonen**[1]
[1]Language Technology Lab, University of Cambridge, UK
[2]Machine Learning Research Group, University of Oxford, UK
{hz416, iv250, alk23}@cam.ac.uk
xwan@robots.ox.ac.uk

Abstract

Large pretrained language models are widely used in downstream NLP tasks via task-specific fine-tuning, but such procedures can be costly. Recently, Parameter-Efficient Fine-Tuning (PEFT) methods have achieved strong task performance while updating much fewer parameters than full model fine-tuning (FFT). However, it is non-trivial to make informed design choices on the *PEFT configurations*, such as their architecture, the number of tunable parameters, and even the layers in which the PEFT modules are inserted. Consequently, it is highly likely that the current, manually designed configurations are suboptimal in terms of their performance-efficiency trade-off. Inspired by advances in neural architecture search, we propose AutoPEFT for automatic PEFT configuration selection: We first design an expressive configuration search space with multiple representative PEFT modules as building blocks. Using multi-objective Bayesian optimization in a low-cost setup, we then discover a Pareto-optimal *set* of configurations with strong performance-cost trade-offs across different numbers of parameters that are also highly transferable across different tasks. Empirically, on GLUE and SuperGLUE tasks, we show that AutoPEFT-discovered configurations significantly outperform existing PEFT methods and are on par or better than FFT without incurring substantial training efficiency costs.

1 Introduction and Motivation

Pretrained language models (PLMs) are used in downstream tasks via the standard transfer learning paradigm, where they get fine-tuned for particular tasks (Devlin et al., 2019; Liu et al., 2019b). This achieves state-of-the-art results in a wide spectrum of NLP tasks, becoming a prevalent modeling paradigm in NLP (Raffel et al., 2020). Fine-tuning the PLMs typically requires a full update of their original parameters (i.e., the so-called *full-model fine-tuning (FFT)*); however, this is (i) computationally expensive and also (ii) storage-wise expensive as it requires saving a separate full model copy for each task-tuned model. With the ever-growing size of the PLMs (Brown et al., 2020; Sanh et al., 2022), the cost of full-model FT becomes a major bottleneck, due to its increasing demands as well as computational (time and space) non-efficiency.

Parameter-efficient fine-tuning (PEFT) delivers a solution for alleviating the issues with full-model FT (Houlsby et al., 2019). By freezing the majority of pretrained weights of PLMs, PEFT approaches only update a small portion of parameters for efficiently adapting the PLM to a new downstream task. Recent studies have shown that PEFT can achieve competitive task performance while being modular, adaptable, and preventing catastrophic forgetting in comparison to traditional FFT (Wang et al., 2022; Pfeiffer et al., 2023).

Recent developments have created diverse PEFT modules with distinctive characteristics (Pfeiffer et al., 2020b; Li and Liang, 2021), with one of the two main aims in focus: **1)** *improve task performance* over other PEFT approaches while *maintaining the same parameter budget* as the competitor PEFT methods; or **2)** *maintain task performance* while *reducing the parameter budget* needed. Existing PEFT modules, optimizing for one of the two aims, have been successfully applied to transfer learning tasks (Chen et al., 2022b; Pfeiffer et al., 2022). However, different tasks, with different complexity, show distinct sensitivity to the allocated parameter budget and even to the chosen PEFT approach (He et al., 2022). At the same time, most PEFT applications are

[*] Equal contribution.

Transactions of the Association for Computational Linguistics, vol. 12, pp. 525–542, 2024. https://doi.org/10.1162/tacl_a_00662
Action Editor: Jacob Eisenstein. Submission batch: 4/2023; Revision batch: 1/2024; Published 5/2024.

limited to a single PEFT architecture (e.g., serial adapters, prefix-tuning) with fixed decisions on its components (e.g., hidden size dimensionality, insertion layers) resulting in *potentially suboptimal PEFT configurations* across many tasks. Therefore, in this work, we propose a new, versatile, and unified framework that automatically searches for improved and task-adapted PEFT configurations, aiming to *effectively balance* between the two (often colliding goals) of (i) improving performance and (ii) keeping the desired low parameter budget for PEFT.

While recent research has started exploring more dynamic PEFT configurations, prior studies remain limited across several dimensions, including how they define the configuration search space. Namely, they typically focus only on a single PEFT architecture (e.g., adapters) or their simple combinations, or a single property (e.g., insertion layers—where to insert the module); see a short overview later in §3. Here, we propose a unified and more comprehensive framework for improved configuration search. It covers multiple standard PEFT modules (serial adapters, parallel adapters, and prefix-tuning) as building blocks, combined with the critical parameter budget-related decisions: the size of each constituent module and the insertion layers for the modules.

Our defined comprehensive search space is huge; consequently, traversing it effectively *and* efficiently is extremely challenging. To enable search over the large configuration space, we thus propose the novel AutoPEFT framework. It automatically configures multiple PEFT modules along with their efficiency-oriented design decisions, relying on a high-dimensional Bayesian optimization (BO) approach. Crucially, within the search space, we propose a multi-objective optimization which learns to balance simultaneously between maximizing the searched configurations' task performance *and* parameter efficiency.

We conduct extensive experiments on the standard GLUE and SuperGLUE benchmarks (Wang et al., 2018, 2019), with encoder-only and encoder-decoder models. We first study the transferability of the AutoPEFT-searched architecture by running AutoPEFT on a single task with a low-fidelity proxy (aiming to reduce computational cost), followed by transferring the found architecture to other tasks. Experimental results show that this architecture can outperform existing PEFT baselines while achieving on-par performance with the standard FFT. Further slight gains can be achieved with a larger computation budget for training, where we run AutoPEFT per task to find a task-adapted PEFT configuration. As revealed in Figure 1, AutoPEFT can find configurations that offer a solid trade-off between task performance and parameter efficiency, even outperforming FFT. We also provide ablation studies over the search space, validating that the AutoPEFT framework is versatile and portable to different search spaces.

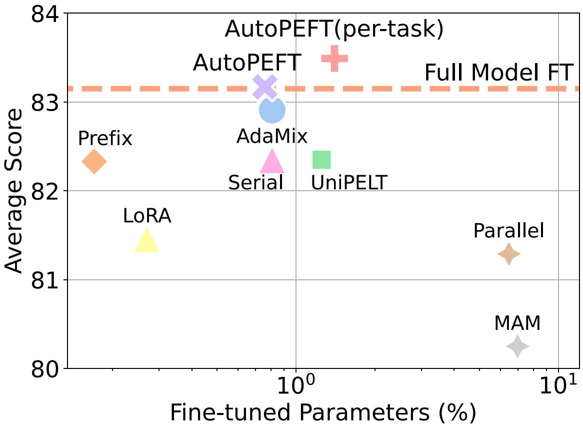

Figure 1: Performance of AutoPEFT-discovered configurations (`AutoPEFT` & `AutoPEFT(per-task)`; see details in Table 1) compared to other baseline PEFT methods (markers) and full model FT that updates 100% of parameters (dashed horizontal bar), averaged across 8 GLUE tasks. Our approach achieves the best trade-off between task performance and parameter efficiency.

Contributions. **1)** We propose the AutoPEFT search space containing diverse and expressive combinations of PEFT configurations from three representative PEFT modules as foundational building blocks and the binary decisions concerning Transformer layers for inserting these modules as searchable dimensions. **2)** To navigate the vast AutoPEFT search space and to discover a *set* of transferable PEFT configurations that optimally trade performance against cost across various parameter ranges *in a single run*, we further propose an effective search method based on multi-dimensional Bayesian optimization. **3)** We demonstrate that the one-time search cost of AutoPEFT is low, and AutoPEFT yields task-shareable configurations, outperforming existing PEFT modules while being transferable

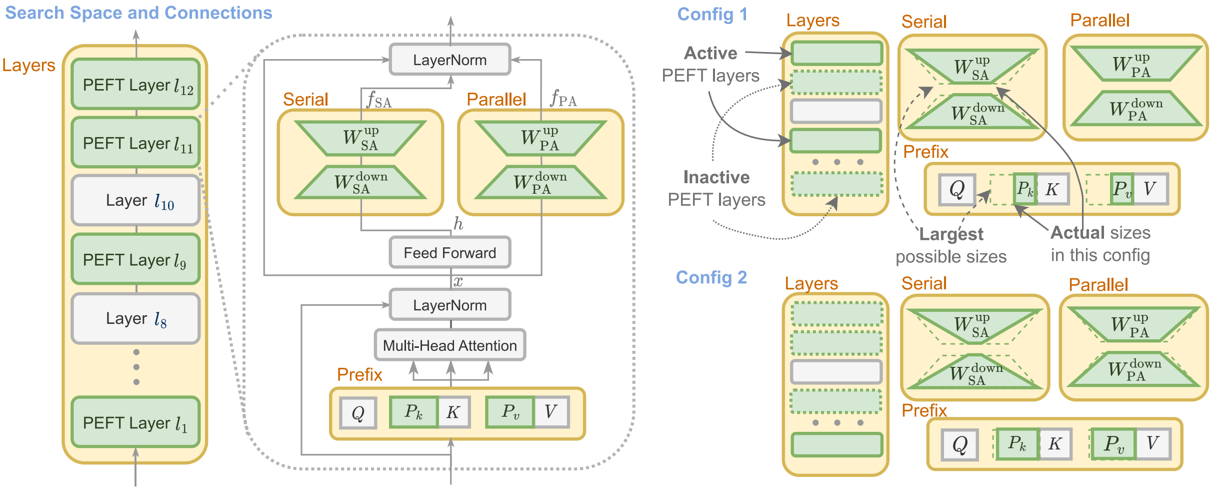

Figure 2: Illustration of the AutoPEFT *search space* which combines both layer-level (`Layers`) and within-layer (`Serial, Parallel, Prefix`) search, and the connections within a layer (**Left**). We further show two possible *configurations* in the search space (**Right**): note that some PEFT layers can be inactive altogether and the searchable module sizes (shaded in green), i.e., the bottleneck sizes in `Serial` and `Parallel` (D_{SA} and D_{PA}, respectively) and sizes of P_K, P_V in `Prefix` (L_{PT}), are dynamic.

across tasks. The AutoPEFT framework can also be easily extended to other and new PEFT modules. The code is available at `https://github.com/cambridgeltl/autopeft`.

2 AutoPEFT Framework

2.1 Designing the AutoPEFT Search Space

Inspired by the success of neural architecture search (NAS) methodology (Ru et al., 2020), we similarly start by designing a large and expressive configuration space. We additionally provide the motivation behind each decision to include a particular module and its components in the configuration space, along with a mathematical formulation.

The search space is known to be one of the most important factors in the performance of the configurations to be discovered subsequently (Ru et al., 2020; Xie et al., 2019; Li and Talwalkar, 2019; Dong and Yang, 2020; Yang et al., 2020). In order to simultaneously maximize task performance along with parameter efficiency, it is necessary to first define a 'parameter-reducible' search space, where each dimension within the space potentially contributes to reducing the parameter budget. Similarly, each dimension potentially impacts the performance positively without introducing redundancy in the space (Wan et al., 2022). Therefore, we propose the following search space with representative PEFT modules spanning a plethora of (non-redundant) configurations as illustrated in Figure 2:

PEFT Modules. Inspired by common practices in NAS of using known well-performing modules as building blocks, we include three distinctive PEFT designs to efficiently adapt different forwarding stages of hidden states in the PLM layers. We combine Serial Adapters (SA), Parallel Adapters (PA), and Prefix-Tuning (PT) as the three representative modules in the search space as the building blocks, where the PT module adapts the multi-head attention layer, and SA and PA interact with the FFN layer (Figure 2). Each configuration makes a decision on the PEFT modules in the insertion layer: all of them can be 'turned' on or off. We combine this binary decision with the actual non-binary decision on the module size (see next) so that the value of 0, in fact, denotes the absence of the modules in the layer(s). We note that other PEFT modules such as LoRA (Hu et al., 2022a) are scaled variants of PA with the same insertion form (He et al., 2022). As we empirically validate later, the resultant search space spanned by the selected building blocks is extremely expressive and flexible and enables the discovery of configurations that outscore any of the individual building blocks and other PEFT modules.

Size. Previous studies show that PEFT methods are highly sensitive to the number of tunable parameters: Adaptively setting their capacity in accordance with the target task is, therefore, essential for achieving good performance (Chen et al., 2022a). The number of tunable parameters depends on each particular module. The additional parameters introduced by both SA and PA are dominated by their bottleneck dimension D. Similarly, the size of the PT module is defined by its prefix length L_{PT}. Thus, we define a binary logarithmic search scale for the respective discrete sets D_{SA}, D_{PA}, and L_{PT}, spanning the values from 0 (absence of the module) to D_{h} where D_{h} is the dimensionality of the output embedding of the PLM (e.g., $D_{\text{h}} = 768$ for BERT$_{\text{base}}$).

Insertion Layers. Prior work has also shown that different layers in the PLMs store different semantic information (Vulić et al., 2020), where the higher layers produce more task-specific and contextualized representations (Tenney et al., 2019). Therefore, as another configuration dimension, we aim to search for the minimal number and the actual position of layers in which to insert the PEFT modules. We define a binary 'insertion' decision at each layer l_i.

Combining PEFT Modules. The SA module and the PA module share a bottleneck architecture. The SA receives hidden states from the FFN output as its inputs, adapting it with a down-projection matrix $W_{\text{SA}}^{\text{down}} \in \mathbb{R}^{D_{\text{h}} \times D_{\text{SA}}}$, followed by a non-linear activation function, and then an up-projection matrix $W_{\text{SA}}^{\text{up}} \in \mathbb{R}^{D_{\text{SA}} \times D_{\text{h}}}$:

$$f_{\text{SA}}(h) = \text{ReLU}(hW_{\text{SA}}^{\text{down}})W_{\text{SA}}^{\text{up}}. \tag{1}$$

PA, on the other hand, receives its inputs from hidden states before the FFN layer with the same formulation:

$$f_{\text{PA}}(x) = \text{ReLU}(xW_{\text{PA}}^{\text{down}})W_{\text{PA}}^{\text{up}}. \tag{2}$$

Therefore, it is able to act in parallel with the SA without interference. Note that the FFN hidden states $h = F(x)$ contain the task-specific bias learned in its pretrained weights. Therefore, by combining SA with PA, the following composition of functions is achieved:

$$\begin{aligned} f_{\text{SAPA}}(x) = \text{ReLU}(F(x)W_{\text{SA}}^{\text{down}})W_{\text{SA}}^{\text{up}} \\ + \text{ReLU}(xW_{\text{PA}}^{\text{down}})W_{\text{PA}}^{\text{up}}. \end{aligned} \tag{3}$$

The final composition should adapt effectively to both bias-influence hidden states and the original inputs before the pretrained FFN layer.[1]

Further, applying PEFT modules to interact with FFNs and multi-head attention should positively impact task performance (Mao et al., 2022; He et al., 2022). PT learns two prefix vectors, P_k and $P_v \in \mathbb{R}^{L_{\text{PT}} \times D_{\text{h}}}$, that are concatenated with the original multi-head attention's key and value vectors, which efficiently adapts the multi-head attention layer to fit the target task. Thus, we finally combine the SA and the PA (i.e., SAPA from above) with PT.

In sum, the overview of the dimensions spanning the final configuration space is provided in Figure 2. The combination of the different 'configuration dimensions' outlined above gives rise to a total of, e.g., 5,451,776 possible configurations with BERT$_{\text{base}}$ and $\sim 3 \times 10^{10}$ configurations with RoBERTa$_{\text{large}}$ (i.e. the number of configurations is $2^{|l|} \times |D_{\text{SA}}| \times |D_{\text{PA}}| \times |L_{\text{PT}}|$). While a large search space is crucial for expressiveness and to ensure that good-performing configurations are contained, it also increases the difficulty for search strategies to navigate the search space well while remaining sample- and thus computationally efficient. Furthermore, in the PEFT setting, we are also often interested in discovering a family of configurations that trade-off between performance and efficiency for general application in various scenarios with different resource constraints, thus giving rise to a multi-objective optimization problem where we simultaneously aim to maximize performance while minimizing costs. In what follows, we propose a search framework that satisfies all those criteria.

2.2 Pareto-Optimal Configuration Search

Multi-objective Optimization Formulation. The ultimate goal of AutoPEFT is to discover promising PEFT configuration(s) from the expressive search space designed in §2.1, which is itself challenging. In this paper, we focus on an even more challenging but practical goal: Instead of aiming to find a single, best-performing PEFT configuration, we aim to discover *a family of Pareto-optimal* PEFT configurations that trade performance against parameter-efficiency (or parameter cost) optimally: One of the most

[1] The PA module also acts as the low-rank reparameterisation of the learned SA and the frozen FFN layer to further match the intrinsic dimensionality of the target task.

impactful use cases of PEFT is its ability to allow fine-tuning of massive language models even with modest computational resources, and thus we argue that searching Pareto-optimal configurations is key as it allows tailored user- and scenario-specific PEFT deployment depending on the computational budget.

Formally, denoting the full AutoPEFT search space as $\mathcal{A}$ and a single configuration $a \in \mathcal{A}$ with trainable weights W, without loss of generality, assuming our objective is to maximize (i) a performance metric $f(a, W)$ (e.g., the accuracy on the dev set) and to (ii) minimize a cost metric $g(a)$ (e.g., the number of parameters in a), a search method aims to solve the bi-level, bi-objective optimization problem:

$$\begin{aligned} &\max_{a \in \mathcal{A}} \Big(f(a, W^*(a)), -g(a) \Big); \\ &\text{s.t.} W^*(a) = \arg\min_{W} \mathcal{L}_{\text{train}}(a, W), \end{aligned} \tag{4}$$

where the inner loop optimization problem is the optimization of the configuration *weights* achieved by fine-tuning the configuration a itself over the training loss $\mathcal{L}_{\text{train}}$. Given the bi-objective nature of the problem, there is, in general, no single maximizer of Eq. (4) but a *set* of Pareto-optimal configurations $A^* = \{a_1^*, \ldots, a_{|A^*|}^*\}$ that are *non-dominated*: We say that a configuration a dominates another a' (denoted $\boldsymbol{f}(a') \prec \boldsymbol{f}(a)$) if $\mathcal{L}_{\text{val}}(a, W^*) \leq \mathcal{L}_{\text{val}}(a', W^*)$ *and* $g(a) \leq g(a')$ and either $\mathcal{L}_{\text{val}}(a, W^*) < \mathcal{L}_{\text{val}}(a', W^*)$ or $g(a) < g(a')$. Denoting $\boldsymbol{f}(a) := [\mathcal{L}_{\text{val}}(a, W^*), g(a)]^\top$, the set of Pareto-optimal architectures A^* are those that are mutually non-dominated: $A^* = \{a_i^* \in \mathcal{A} \mid \nexists\, a' \in \mathcal{A} \text{ s.t. } \boldsymbol{f}(a') \prec \boldsymbol{f}(a_i^*)\}$. The Pareto front (PF) $\mathcal{P}^*$ is the image of the Pareto set of architectures: $\mathcal{P}^* = \{\boldsymbol{f}(a) \mid a \in A^*\}$.

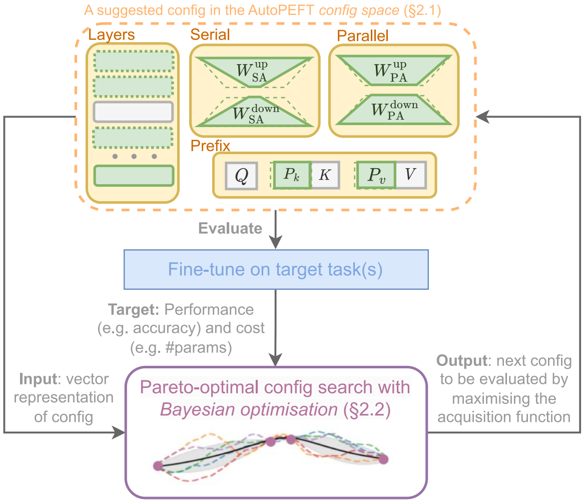
A suggested config in the AutoPEFT config space (§2.1)
Layers
Serial
Parallel
$W_{\text{SA}}^{\text{up}}$
$W_{\text{SA}}^{\text{down}}$
$W_{\text{PA}}^{\text{up}}$
$W_{\text{PA}}^{\text{down}}$
Prefix
Q
P_k
K
P_v
V
Evaluate
Fine-tune on target task(s)
Target: Performance (e.g. accuracy) and cost (e.g. #params)
Input: vector representation of config
Pareto-optimal config search with Bayesian optimisation (§2.2)
Output: next config to be evaluated by maximising the acquisition function

Figure 3: Illustration of the Pareto-optimal search with multi-objective Bayesian optimization (BO; §2.2): The BO agent trains on the vector representations of the evaluated configurations as inputs and their performance under a low-fidelity setup (e.g., accuracy—obtained by fine-tuning the language model with the PEFT configuration for a small number of iterations) and cost (e.g., number of parameters) as targets. The BO agent then iteratively suggests new configurations until convergence.

Bayesian Optimization (BO). To solve Eq. (4), we adopt a BO approach, illustrated in Figure 3. On a high level, BO consists of a *surrogate model* that sequentially approximates the objective function based on the observations so far and an *acquisition function* that is optimized at each iteration to actively select the next configuration to evaluate. Typically, the surrogate model is a Gaussian process (GP), a flexible and non-parametric model with well-principled and closed-form uncertainty estimates: Given an observed set of n configurations and their evaluated performance: $\mathcal{D}_n = \{\big(a_i, \boldsymbol{f}(a_i)\big)\}_{i=1}^n$, the GP surrogate model gives a closed form posterior distribution $\mathbb{P}(\boldsymbol{f}(a)|\mathcal{D}_n)$ over the true, unobserved function values $\boldsymbol{f}$ potentially over configurations that have *not* been evaluated before. The acquisition function $\alpha : \mathcal{A} \rightarrow \mathbb{R}$, on the other hand, uses the posterior distribution of the surrogate model to assign a utility value to possible configuration candidates in $\mathcal{A}$, typically balancing exploitation (i.e., querying near configurations in $\{a_i\}_{i=1}^n$ that were previously observed to be strong) and exploration (i.e., the configurations far from $\{a_i\}_{i=1}^n$ and are those we do not have knowledge on and can potentially be even better configurations). At each step of BO, the acquisition function is optimized (note that while evaluating $\boldsymbol{f}(a)$ is expensive, evaluating $\alpha(a|\mathcal{D})$, which only uses the posterior distribution from the surrogate model, is not) to select the next configuration (or batch of configurations) $a_{n+1} = \arg\max_{a \in \mathcal{A}} \alpha(a|\mathcal{D}_n)$ to evaluate. For a detailed overview of BO, we refer the readers to Garnett (2023) and Frazier (2018).

Rationales for Using BO. We argue that BO is well-suited to the task in principle and has various advantages over alternative, viable approaches such as those based on differentiable

NAS (DARTS) (Liu et al., 2019a), which typically utilize a *continuous relaxation* of the discrete configurations, thereby allowing a to be *jointly* optimized with the model weights W in Eq. 4 with a supernet.

First, unlike the DARTS-based approach, by treating the optimization problem defined in Eq. 4 as a *black box*, BO decouples the optimization of the weights W and the optimization of architecture a, and solves the latter problem with no gradient information at all (White et al., 2021; Ru et al., 2021). This makes a BO-based solution more parallelizable and more amenable to a distributed setup, which modern large PLMs often rely on, as multiple configuration evaluations may take place simultaneously in different client machines as long as they can relay the evaluation results $\boldsymbol{f}$ back to a central server running the BO. This further contributes to memory efficiency, as unlike the DARTS-based method that optimizes a supernet (a heavily over-parameterized network that can be deemed as a weighted superposition of all configurations in $\mathcal{A}$), each parallel evaluation in BO trains a single configuration only; we argue that this point is particularly important for PEFT given its main promise on *parameter efficiency*.

Second, as discussed, it is often desirable to discover a *family* of configurations with different trade-offs between performance and parameters in different application scenarios. As we will show, while BO generalizes elegantly to handle vector-valued objective functions and may generate a PF of configurations *in a single run*, competing methods, such as supernet-based NAS methods, typically require a scalar objective function and thus are limited to discovering a single best-performing configuration (Eriksson et al., 2021; Izquierdo et al., 2021); this means that one typically needs to run the NAS pipeline multiple times for different cost budgets in these methods.

Lastly, while one of the main arguments favoring differentiable techniques is its lighter computational expense as one only needs to train the supernet once rather than repeatedly training different candidate configurations, as we will later show, the sample-efficient nature of BO and strong transferability of the discovered configurations also ensure that the computational cost of our proposed method remains tractable. As we will show in §4, while DARTS-based NAS is indeed a plausible approach for PEFT configuration search, we show that our approach performs competitively to S^{3}PET (Hu et al., 2022b), a DARTS-based method.

Algorithm 1 Overall AutoPEFT search pipeline.

```
Input: number of randomly initialising points $N_0$, maximum number of config evaluations $N > N_0$, AutoPEFT search space $\mathcal{A}$.
Output: a set of Pareto-optimal configs $A^*$.
Initialise by sampling randomly at $N_0$ configurations $a \sim \mathcal{A}$ and fine-tune the PLM to obtain $\boldsymbol{f}(\cdot)$ of the corresponding configs. Initialise $\mathcal{D}_0 \leftarrow \{(a_i, \boldsymbol{f}(a_i))\}_{i=1}^{N_0}$ and fit a SAAS-GP model on $\mathcal{D}_0$.
for $n = N_0, \ldots, N$ do
    Select the next configuration(s) to evaluate $a_n$ by maximizing the NEHVI acquisition function $a_n = \mathrm{argmax}_{a \in \mathcal{A}} \alpha(a|\mathcal{D}_{n-1})$.
    Fine-tune the PLM with candidate configuration(s) $a$ (possibly with low-fidelity estimates) to obtain $\boldsymbol{f}(a)$ // Inner-loop optimization in Eq. 4.
    Augment the observation data $\mathcal{D}_n \leftarrow \mathcal{D}_{n-1} \cup (a_t, \boldsymbol{f}(a_t))$ and update the SAAS-GP model.
end for
Return the set of non-dominated configurations $A^* \subseteq \{a_i\}_{i=1}^{N}$.
```

Adapting BO to the AutoPEFT Task. Adapting BO to the high-dimensional and combinatorial AutoPEFT search space is non-trivial. To address the challenges, we customize both components of BO, and the overall pipeline is shown in Algorithm 1. Instead of a standard GP, we propose to use a *Gaussian process with sparse axis-aligned subspaces* (SAAS-GP) (Eriksson and Jankowiak, 2021) as the surrogate model: As an intuitive explanation, SAAS-GP places strong, sparsity-inducing priors on the GP hyperparameters to alleviate the difficulty in modeling high-dimensional data by assuming that despite the high nominal dimensionality, *some* search dimensions contribute much more significantly to the variation of the objective function than others—this assumption is shown to hold in related problems of *NAS in computer vision* (Wan et al., 2022) and *discrete prompt search in PLMs* (Zhou et al., 2023), and we expect similar findings in our particular case.

For the acquisition function, we use the noisy expected hypervolume improvement (NEHVI) (Daulton et al., 2021) to handle the multi-objective setting: Unlike the commonly used scalarisation approach that transforms the vector-valued objective function to a scalar weighted sum (which corresponds to *a single point* on the PF), NEHVI is capable of automatically exploring all parts of the PF in a single run. Lastly, we additionally use *low-fidelity* approximations, a popular low-cost performance estimation strategy in NAS (Elsken et al., 2019), to manage the search cost: At search-time, instead of fine-tuning each candidate PEFT configuration in full, we only fine-tune with a much smaller number of iterations (5% of full)—this is possible as we are only interested in the *relative ranking* (rather than the performance itself) of the different configurations during search. Consistent with NAS literature, we also find the low-fidelity estimate to provide a reliable ranking, with the best-performing configurations in low fidelity also performing the best under fine-tuning with the full number of iterations. As we will show in §5, using the low-fidelity search pipeline, in combination with the strong transferability of the discovered configurations, AutoPEFT only incurs an additional *one-off*, **1.9%** of the total GLUE fine-tuning cost, but delivers significant performance gains.

3 Related Work

PEFT Methods in NLP. Standard PEFT methods can be divided into two main groups (Pfeiffer et al., 2023). **1)** Some methods fine-tune a small portion of pretrained parameters (Zhao et al., 2020; Guo et al., 2021). For instance, Ben Zaken et al. (2022) propose to fine-tune the PLM's bias terms, while Sung et al. (2021) and Ansell et al. (2022) fine-tune sparse subnetworks withing the original PLM for a particular task. **2)** Other methods fine-tune an additional set of parameters (Liu et al., 2022). Since there is no interference with the pretrained parameters, this class of PEFT modules, besides offering strong task performance, is arguably more modular; we thus focus on this class of PEFT methods in this work. The original *adapter modules* (Houlsby et al., 2019; Pfeiffer et al., 2020b) have a bottleneck *serial* architecture which can be inserted into every Transformer layer, see Figure 2. LoRA (Hu et al., 2022a) assumes the low-rank intrinsic dimensionality of the target task and performs low-rank updates (Mahabadi et al., 2021). Li and Liang (2021) propose the Prefix-Tuning method that appends a learnable vector to the attention heads at each Transformer layer. Similarly, prompt-tuning (Lester et al., 2021) only appends this vector to the input embedding. UniPELT (Mao et al., 2022) integrates multiple PEFT modules with a dynamic gating mechanism. He et al. (2022) provide a unified formulation of existing PEFT modules and propose a *parallel* adapter module, along with a combined 'Mix-and-Match Adapter (MAM)' architecture that blends parallel adapters and prefix-tuning. Wang et al. (2022) propose the mixture-of-adaptations (AdaMix) architecture with weight averaging for a mixture of adapters.

Optimizing Parameter Efficiency in PEFT. Recent work further aims to optimize the parameter efficiency of existing PEFT modules while maintaining task performance. The standard approach is to insert (typically serial) adapters into all Transformer layers, which still requires a sizeable parameter budget. Rücklé et al. (2021) address this question by randomly dropping adapters from lower-level layers, displaying only a small decrease in task performance. Adaptable Adapters (AA) (Moosavi et al., 2022) generalize this idea by learning gates that switch on or off adapters in particular Transformer layers. Neural Architecture Search (NAS) methods aim to automate the design of neural net architectures themselves, and NAS has seen great advances recently, with performance often surpassing human expert-designed architectures in various tasks (Zoph and Le, 2017; Ren et al., 2021; Elsken et al., 2019). Concerning NLP tasks and PEFT, Hu et al. (2022b) propose S^{3}PET, which adapts Differentiable Architecture Search (DARTS) (Liu et al., 2019a) to learn the positions for inserting the PEFT modules. This work is closest in spirit to ours and is empirically compared to in §4. Conceptually, however, as discussed in detail in §2, we argue that our method offers a spectrum of advantages over S^{3}PET and other related PEFT work, including but not limited to the ability to automatically discover a family of PEFT configurations across parameter budgets in a single run, better parallelisability and memory efficiency. Other concurrent work (Valipour et al., 2023; Zhang et al., 2023) also approaches the same problem by dynamic budg et allocation mechanisms on a single PEFT module within a limited search space. Nonetheless, this field still lacks a compact solution for automatically configuring a complex space of PEFT modules (Chen et al., 2023).

4 Experimental Setup

Evaluation Data. We follow prior PEFT research and base our evaluation on the standard and established GLUE and SuperGLUE benchmarks. For GLUE, we include 4 types of text classification tasks, including linguistic acceptability: CoLA; similarity and paraphrase: STS-B, MRPC, QQP; sentiment analysis: SST-2; natural language inference: RTE, QNLI, MNLI. We exclude WNLI following previous work (Houlsby et al., 2019; Mao et al., 2022). We also include CB, COPA, WiC, and BoolQ from SuperGLUE to further validate the transferability of AutoPEFT-found configuration across different tasks and datasets.

Baselines. We compare the performance of the AutoPEFT-found configurations to the standard full model FT and each individual PEFT module (SA, PA, PT) from the search space used in their default setup from their respective original work. We also compare with the LoRA module to provide a comparison to low-rank decomposition methods. To compare with recent methods that also integrate multiple PEFT modules (see §3), we further include the UniPELT and the MAM adapter in their default settings. We reproduce AdaMix for a comparison to a mixture of homogeneous adaptations. In ablations on insertion layers, we also include the Adaptable Adapter (AA) as a baseline that proposes a differentiable gate learning method to select the insertion layer for PEFT modules (i.e. serial adapters originally). On T5 (Raffel et al., 2020) models, we also compare against S^{3}PET (Hu et al., 2022b), one of the most similar works to us that use differentiable NAS for configuration search.

Implementation Details. Following previous work on the GLUE benchmark, we report the best GLUE dev set performance (Ben Zaken et al., 2022) and use 20 training epochs with an early stopping scheme of 10 epochs for all *per-task* experiments. We use AdapterHub (Pfeiffer et al., 2020a) as the codebase and conduct extensive experiments with the uncased BERT$_{\text{base}}$ (Devlin et al., 2019) as the main backbone model. We report main experiments with the mean and standard deviation over 5 different random seeds. Following Pfeiffer et al. (2020b), we use a recommended learning rate of 10^{-4} for all PEFT experiments. We use the learning rate of 2×10^{-5} for full model FT according to Mao et al. (2022). We use batch sizes 32 and 16 for all BERT and RoBERTa experiments, respectively. The optimizer settings for each PEFT module follow the default settings in AdapterHub (Pfeiffer et al., 2020a). We implement the BO search algorithm in BoTorch (Balandat et al., 2020) and use the recommended settings from Eriksson and Jankowiak (2021) for the surrogate. For acquisition function optimization, we use a local search method similar to previous literature with a similar setup (Wan et al., 2021; Eriksson et al., 2021): At each search iteration (after the initial randomly sampled points), we collect the *Pareto-optimal* architectures up to this point. From this collection of Pareto-optimal architectures, we perform a local search by evaluating the acquisition function values of their neighbors and move the current point to a neighbor with a higher acquisition function value, and this process is repeated until convergence. Due to the relatively noisy nature of the problem, we use 100 random initialization points for all experiments, followed by 100 BO iterations. We further show results using RoBERTa$_{\text{large}}$ (Liu et al., 2019b) in Table 5, which shows findings that are consistent with the BERT$_{\text{base}}$. In experiments with RoBERTa$_{\text{large}}$ as the underlying PLM, we report the RTE results with a learning rate of 2×10^{-5} for AutoPEFT^{MRPC} and AutoPEFT^{CoLA}; 10^{-4} for AutoPEFT^{RTE}. We use batch size 16 and a learning rate of 3×10^{-4} for T5$_{\text{base}}$ experiments by AutoPEFT with the SAPA space; 10^{-5} for STS-B. We reproduce S^{3}PET results with batch size 8 in the same experimental setup as AutoPEFT.

5 Results and Discussion

Discussion of Main Results. The main results on BERT are summarized in Table 1, where we evaluate the AutoPEFT-found configurations searched from RTE, the most low-resource and challenging task, on the full GLUE suite. We further report selected GLUE tasks on T5 in Table 4 (where we also compare against S^{3}PET) and RoBERTa$_{\text{large}}$ in Table 5. For simplicity, we report a single configuration that leads to the highest task performance in a predefined, user-specified parameter budget from the discovered Pareto-optimal set in Table 1, whereas the full Pareto-optimal set is evaluated in Figure 4. On BERT (Table 1, we find that using only 0.76% of parameters, AutoPEFT^{RTE} outperforms all the

Method	#Param.	RTE	MRPC	STS-B	CoLA	SST-2	QNLI	QQP	MNLI	Avg.
FFT	100%	$71.12_{1.46}$	$85.74_{1.75}$	$89.00_{0.45}$	$59.32_{0.62}$	$\mathbf{92.57}_{0.24}$	$\underline{91.50}_{0.08}$	$91.52_{0.04}$	$\mathbf{84.43}_{0.22}$	83.15
Prefix	0.17%	$70.54_{0.49}$	$85.93_{0.89}$	$88.76_{0.15}$	$58.88_{1.15}$	$91.93_{0.45}$	$90.76_{0.14}$	$89.12_{0.07}$	$82.78_{0.16}$	82.33
LoRA	0.27%	$65.85_{1.49}$	$84.46_{1.04}$	$88.73_{0.08}$	$57.58_{0.78}$	$92.06_{0.38}$	$90.62_{0.22}$	$89.41_{0.04}$	$83.00_{0.07}$	81.46
Serial	0.81%	$68.01_{1.34}$	$84.75_{0.45}$	$88.61_{0.11}$	$59.73_{0.62}$	$91.93_{0.33}$	$91.06_{0.12}$	$90.52_{0.05}$	$84.18_{0.22}$	82.35
AdaMix	0.81%	$70.11_{0.62}$	$86.86_{1.12}$	$\underline{89.12}_{0.11}$	$59.11_{1.00}$	$92.06_{0.22}$	$\mathbf{91.52}_{0.15}$	$90.22_{0.04}$	$84.25_{0.14}$	82.91
UniPELT	1.25%	$67.07_{1.82}$	$84.22_{0.78}$	$88.84_{0.11}$	$60.13_{0.46}$	$\underline{92.52}_{0.24}$	$91.09_{0.13}$	$90.69_{0.11}$	$\underline{84.28}_{0.18}$	82.35
Parallel	6.46%	$68.52_{3.44}$	$86.52_{0.96}$	$88.90_{0.28}$	$58.72_{1.69}$	$92.13_{0.35}$	$90.83_{0.22}$	$\underline{90.74}_{0.08}$	$73.93_{19.24}$	81.29
MAM	6.97%	$69.10_{1.76}$	$\underline{87.16}_{0.74}$	$89.01_{0.48}$	$47.87_{23.97}$	$83.94_{16.52}$	$90.85_{0.22}$	$\mathbf{90.76}_{0.05}$	$83.31_{0.17}$	80.25
AutoPEFT^{RTE}	0.76%	$\underline{72.20}_{0.72}$	$\underline{87.16}_{0.83}$	$88.77_{0.07}$	$\underline{60.30}_{1.24}$	$92.22_{0.30}$	$90.90_{0.10}$	$90.37_{0.06}$	$83.46_{0.21}$	$\underline{83.17}$
AutoPEFT$^{task}_{Avg.}$	$\overline{1.40}$%	$\mathbf{72.35}_{0.94}$	$\mathbf{87.45}_{0.87}$	$\mathbf{89.17}_{0.24}$	$\mathbf{60.92}_{1.47}$	$92.22_{0.30}$	$91.12_{0.13}$	$90.64_{0.05}$	$84.01_{0.10}$	**83.49**

Table 1: Results on the GLUE benchmark with BERT$_{\text{base}}$ (tasks are ranked in ascending order of training resources required from left to right). For AutoPEFT^{RTE}, we search on RTE with a low-fidelity proxy, training for 1 epoch per iteration, *only at a search cost of 1.9% (in terms of additional fine-tuning steps required) over the full GLUE experiment.* We report the $\overline{\text{average}}$ fine-tuned parameters of *per-task* AutoPEFT, where we conduct additional *per-task* searches on RTE, MRPC, STS-B, and CoLA, and take best-found configurations for the remaining tasks. We report Spearman's Correlation for STS-B, Matthew's Correlation for CoLA, and accuracy for all other tasks (matched accuracy for MNLI). The percentage of parameters is the ratio of the number of additional parameters to the pretrained parameters. We reproduce all baselines and report the mean and standard deviation of all results for 5 random seeds. The **best** and $\underline{\text{second-best}}$ results are marked in **bold** font and underlined, respectively.

PEFT baselines (more than 2% on RTE). The AutoPEFT-found configuration also outperforms the full-model FT baseline on the RTE task by more than 1%. These results indicate the effectiveness of the AutoPEFT framework in optimizing both task performance and parameter efficiency. Transferring the RTE-based configurations to other tasks, we find that strong performance is maintained across the target tasks, with more benefits on the medium-resource tasks (MRPC, STS-B, CoLA), but the configuration remains competitive also for higher-resource tasks (e.g., QQP, MNLI). Finally, we find the strength of AutoPEFT to persist in RoBERTa and T5 as a representative of the encoder-decoder model families. It is particularly noteworthy that in addition to outperforming the baseline PEFT methods without configuration search, AutoPEFT also performs competitively compared to S^{3}PET *with configuration search* under a comparable parameter count, even though S^{3}PET was *exclusively developed and tested on the T5 search space* and that the *S^{3}PET search space was designed with meticulous hand-tuning*, where the authors manually excluded several building blocks that did not lead to empirical gain; this provides further empirical support to the strength of a BO-based search strategy described in §2.2.

Table 2 specifies the composition of the found configuration, indicating the exact task-active layers while allocating more parameter budget to the efficient and effective PA module. On average, the AutoPEFT^{RTE} configuration shows a comparable fine-tuning performance (83.17) to FFT (83.15) by only updating 0.76% of parameters. With strong transferability across similar tasks, AutoPEFT provides distinct advantages in parameter efficiency; the search algorithm itself, coupled with the transfer, becomes more sample-efficient within limited training resources.

Task	%Param.	Active PEFT Layers l_i	Submodule	Value
RTE	0.76%	3, 4, 8, 9, 10	D_{SA} (Serial)	12
			D_{PA} (Parallel)	96
			L_{PT} (Prefix)	1

Table 2: Specification of the discovered configuration reported in Table 1 (AutoPEFT^{RTE}) using BERT$_{\text{base}}$.

Method	CB	COPA	WiC	BoolQ	Avg.
FFT	$\mathbf{71.43}_{1.13}$	$51.80_{3.76}$	$\underline{68.62}_{1.93}$	$\mathbf{72.17}_{0.86}$	$\underline{66.01}$
LoRA	$67.14_{2.42}$	$\underline{55.80}_{1.47}$	$68.56_{1.11}$	$69.09_{0.42}$	65.15
Serial	$67.86_{1.13}$	$54.20_{7.68}$	$67.34_{0.61}$	$70.00_{0.85}$	64.86
Ours^{RTE}	$\underline{71.07}_{2.86}$	$\mathbf{56.40}_{6.83}$	$\mathbf{68.87}_{1.06}$	$\underline{70.86}_{0.89}$	**66.80**

Table 3: Results on SuperGLUE tasks with AutoPEFT-discovered configurations *searched on RTE* with BERT$_{\text{base}}$ as the underlying PLM. We split 10% of the training set as the new validation set and report the AutoPEFT^{RTE}-found configuration transfer results on the evaluation set over five random seeds.

Method	#Param.	RTE	MRPC	STS-B	CoLA	SST-2	QNLI	QQP	MNLI	Avg.
LoRA	0.40%	80.1	**89.5**	89.2	59.9	94.4	**93.6**	**91.0**	86.5	85.5
Serial	0.79%	78.0	88.2	89.1	60.6	**94.6**	93.1	90.7	86.4	85.1
S^{3}PET^{RTE}	0.30%	79.8	89.0	**90.2**	58.6	94.2	93.3	90.6	86.5	85.3
AutoPEFT^{RTE}	0.33%	**82.7**	89.0	89.6	**61.7**	**94.6**	93.3	90.8	**86.7**	**86.1**

Table 4: Experimental results on GLUE with T5$_{\text{base}}$. We report comparisons of in-task search performance and transfer performance between the architectures found by AutoPEFT and the state-of-the-art baseline S^{3}PET in a constrained parameter budget. Consistent with Table 1, we report AutoPEFT and S^{3}PET results searched on RTE in full-resource settings that are then transferred to all other included GLUE tasks.

Method	#Param.	RTE	MRPC	STS-B	CoLA	SST-2	QNLI	Avg.
FFT†	100%	86.6	90.9	**92.4**	68.0	96.4	94.7	88.2
LoRA‡	0.22%	85.2	90.2	92.3	68.2	96.2	**94.8**	87.8
Serial	0.89%	84.8	90.2	92.0	66.8	96.3	94.7	87.5
AutoPEFT^{RTE}	0.03%	**88.1**	89.5	92.3	67.0	96.0	94.6	87.9
AutoPEFT$^{task}_{Avg.}$	0.88%	**88.1**	**92.2**	**92.4**	**70.6**	**96.8**	94.6	**89.1**

Table 5: Experimental results on GLUE with RoBERTa$_{\text{large}}$. We report the full model fine-tuning† results from Liu et al. (2019b) with Pearson correlation for STS-B. We include the LoRA‡ module performance from Hu et al. (2022a). We exclude QQP and MNLI tasks due to the high computation cost of RoBERTa$_{\text{large}}$. Consistent with Table 1, we again report AutoPEFT results searched on RTE in full-resource settings that are then transferred all included GLUE tasks (AutoPEFT^{RTE}) and per-task AutoPEFT (AutoPEFT$^{task}_{Avg.}$) but on RoBERTa$_{\text{large}}$.

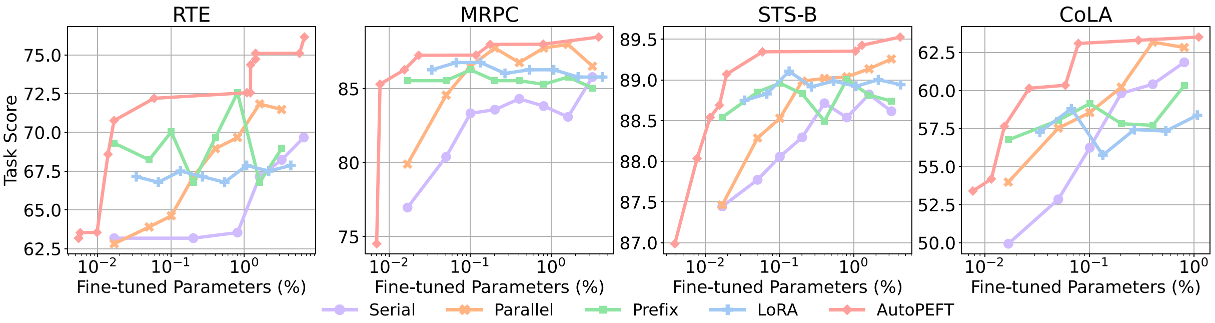

Figure 4: Pareto fronts of AutoPEFT on four tasks compared to baselines on BERT$_{\text{base}}$, over varying parameter budgets. We report the single-seed task score but otherwise follow the settings in Table 1.

Extending AutoPEFT to More Tasks. We next 'stress-test' the ability of AutoPEFT-found configuration in a more challenging scenario, experimenting on a completely new set of dissimilar tasks. Table 3 reports the results of transferring AutoPEFT^{RTE} from Table 1 to four SuperGLUE tasks. In terms of *parameter efficiency*, we observe consistent patterns as in Table 1 before, where our *plug-and-play* PEFT configuration outperforms existing PEFT baselines by a substantial margin (2%) on average while being comparable to the costly full-model FT.[2] In terms of *search cost*, we recall that through the use of low-fidelity proxy and the strong transferability, AutoPEFT^{RTE} in Table 1 only requires an additional, one-off 1.9% in terms of training time (or equivalently the number of fine-tuning steps) of that of single-seed training of the GLUE training sets. Furthermore,

[2] With the AutoPEFT-found off-the-shelf configuration, this requires no additional search cost and enables a more efficient and effective tuning approach for new tasks.

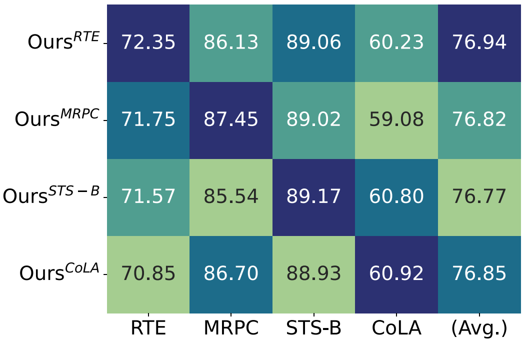

Figure 5: Pairwise transferability study of AutoPEFT-discovered configurations: each **row** (`Ours`$^{\texttt{[task]}}$) denotes the performances of the AutoPEFT configuration searched from `[task]` (e.g., RTE) to the task itself and 3 other GLUE tasks. The results suggest that AutoPEFT performance is largely robust to the choice of which task to search on.

Figure 5 demonstrates the robustness of our framework to the choice of the source task to search on. Therefore, our framework is task-agnostic with a cheap one-time cost but yields 'permanent' improvement towards all efficiency metrics for PEFT: space, time, and memory.

***Per-Task* Search.** We further conduct full-resource per-task AutoPEFT searches. While naturally more expensive, we argue this setup is useful if, for example, one is interested in finding absolutely the best configurations *for that particular task* and where search cost is less of a concern. Due to computational constraints, we search per-task on RTE, MPRC, STS-B, and CoLA, then port the small set of best configurations to the remaining higher-resource tasks (SST-2, QNLI, QQP, MNLI). We observe consistent gain in all tasks we search on over the best-performing PEFT baselines, e.g., MRPC (87.16% (*best baseline*) to 87.45% (*ours*)) and CoLA (60.13% to 60.92%), and also the transferred configuration AutoPEFT^{RTE} in Table 1. One interpretation is that while configurations are highly transferable, the optimal configurations may nonetheless differ slightly across tasks such that while transferred AutoPEFT configurations (e.g., the one reported in Table 1) perform *well*, searching per-task performs the *best*. Crucially, we also find per-task AutoPEFT in this setup to even *outperform FFT, despite only using 1.4% of all parameters*, except for the high-resources task where we mostly perform on par; this is consistent with our observations that similar to the baselines, due to the richness of training resources, the performance may be mostly saturated and PEFT methods often achieve on-par performance to FFT at most.

Analyzing the 'Behavior' of BO and the Discovered Configurations. Figure 7 shows the distribution of AutoPEFT-found configurations when we conduct its search experiment on RTE. Recalling that the search strategy (§2.2) starts with random initialization, we compare the behaviors of the random explorations and the BO-suggested configurations: Whereas the random search baseline is purely exploratory and discovers less parameter-efficient configurations, BO succeeds in discovering configurations towards the regions with improved parameter efficiency. The superiority of BO over the random search baseline is further demonstrated quantitatively by Figure 8 where we compare the evolution of the *hypervolume*, which measures the size of the space enclosed by the Pareto front over a reference point (set to the nadir point of the optimization trajectory) (Zitzler and Thiele, 1998), discovered by BO and random search as a function of the number of configurations evaluated; it is clear that as optimization proceeds, BO finds a better Pareto set with a better trade-off between performance and cost in the end. BO eventually discovers a rich family of PEFT configurations across a wide range of parameters, whereas previous approaches typically fail to explore the entire PF. This is a critical strength motivating our BO search strategy.

Figure 6, on the other hand, visualizes the discovered sets in different tasks: we observe that *within* the Pareto-optimal configuration set of the same task, some layers are consistently enabled (e.g., Layer 2 in CoLA) whereas some are consistently disabled (e.g., Layer 1 across all tasks) even under very different cost budgets; this suggests PEFT modules in different layers are not equally important, and by *selectively* enabling them, AutoPEFT is capable of making better use of the parameter budgets by allocating them to the more beneficial Transformer layers only. We observe the unanimity of preference or disinclination towards certain layers extends even *across* tasks that are unlikely to stem from randomness only: For example, we found Layers 2 and 10 are enabled in 71.2% and 69.2% in all Pareto-optimal configurations over all tasks, whereas Layers 1 and

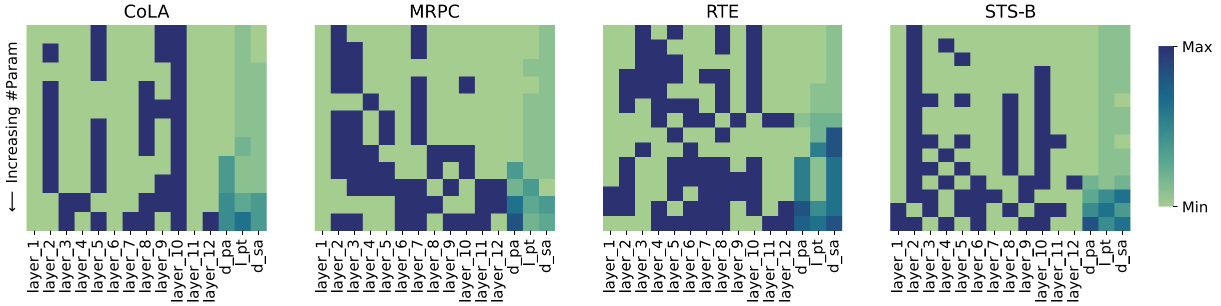

Figure 6: Visualisation of the BO discovered Pareto-optimal sets of configurations A^* in different tasks (i.e., the configurations on the PFs in Figure 4) in ascending order of parameter budget. `layer_i` denotes the binary choice of whether the PEFT module is active in the i-th layer of the PLM. The final 3 columns denote D_{SA}, D_{PA} and L_{PT} respectively, and feature a range of possible values from 0 to 768.

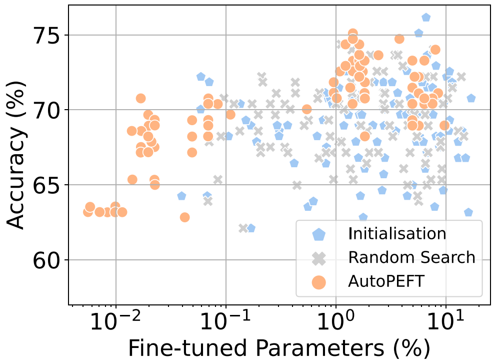

Figure 7: The distribution of the discovered configurations via BO (orange), described in §2.2 and random search (grey) using the same total number of evaluations (200). Both searches use the same 100 random initialising points (blue) on RTE. Note that BO-generated configurations typically have much better parameter efficiency for configurations with similar accuracy.

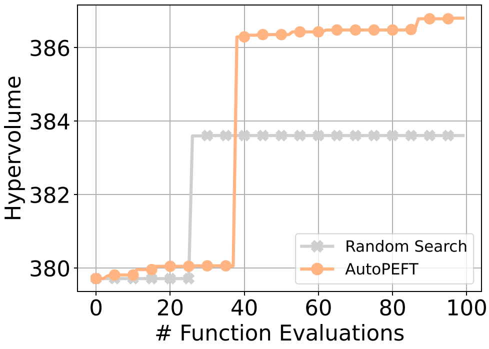

Figure 8: The hypervolumes of the Pareto-optimal configurations discovered by BO (orange) and random search (grey) as a function of the number of configurations evaluated.

12 are enabled in only 7.7% and 13.4% of the time, respectively. We also observe that across all tasks, a common trend is that sequential and prefix adapters are universally preferred in low-budget ranges, and parallel adapters are only enabled when we have a more lenient budg et allowance; these commonalities in high-performing configurations may, to some extent, account for the strong transferability of the discovered configurations, as shown in Figure 5.

Ablation of the Configuration Space. To provide a finer-grained analysis of factors that bring positive impact to AutoPEFT, we ablate the AutoPEFT search space from the full configuration space: 1) to the basic enumeration of the bottleneck size D_{SA} of the SA only (the `SA` space); 2) a naïve baseline where instead of searching for each search dimension independently, we vary a single, common coefficient that generates a family of configurations of different sizes by scaling from the largest PEFT configuration in our search space (SA-PA-PT) over D_{SA}, D_{PA} and L_{PT}. We then include the Transformer layer and the SA size into the search space (the `SA-Layer` space) to validate the usefulness of layer selection as one configuration dimension. We can then also expand the search space by adding another module (e.g., PA yields the `SA-PA-Layer` space). Figure 9 plots the performance over the ablated configuration spaces and different parameter budgets. Several key findings emerge. First, combining multiple single PEFT modules has a positive impact on AutoPEFT in general (c.f. full AutoPEFT vs. `SA-PA-Layer` vs `SA-Layer`). Second, simply scaling all search dimensions by a common scaling factor is sub-optimal. This is likely because not all parameters are equally important, necessitating a configuration search. Relying on layer selection also brings benefits

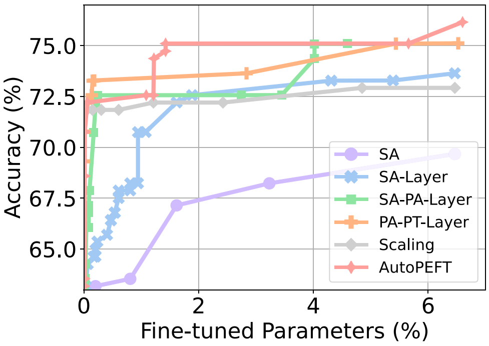

Figure 9: The performance of AutoPEFT with ablation of search space on RTE on BERT$_{\text{base}}$. The SA results refer to the Pfeiffer adapter (Pfeiffer et al., 2020b) with an enumeration of its bottleneck size. The `Scaling` results refer to the PF where smaller configurations are obtained by simply scaling the largest configuration in $\mathcal{A}$ over all search dimensions. We report the PF of AutoPEFT-found configurations, where `SA-PA-PT-Layer` forms the search space of AutoPEFT.

(c.f. `SA` vs. `SA-Layer`). The comparison indicates that *leaving out Transformer layers while increasing the capacity of the PEFT module* is a straightforward method to improve the parameter efficiency and task performance of the PEFT module within a fixed parameter budget. The ablation results also demonstrate that AutoPEFT is search space-agnostic, capable of effectively operating over configuration spaces of different granularity.

Layer Selection. The ability to disable some PEFT layers altogether is a key novelty of the AutoPEFT search space, and to further compare different layer selection approaches, we conduct a controlled experiment with the SA module on BERT$_{\text{large}}$ (24 Transformer layers) under a predefined parameter budget. In Table 6, we compare against AdapterDrop, which simply drops the adapters for the first 11 layers while doubling their bottleneck sizes, and, within the same architecture, we also include the Adaptable Adapter with selected layers from switch learning (3 and 10 layers from the first 12 and the other 12 layers, respectively). We show that AutoPEFT outperforms existing layer selection baselines activating fewer PEFT layers, leading to better parameter efficiency (12.5% fewer parameters in relative terms) yet achieving better performance. It indicates that selecting the best insertion layer is non-trivial, and AutoPEFT can efficiently learn the correlation between layers.

Method	#Layers	Size D_{SA}	RTE
Serial	24	64	$72.56_{0.76}$
Adaptable Adapter	13	128	$73.36_{0.80}$
AdapterDrop	13	128	$73.50_{1.40}$
AutoPEFT$^{SA}_{Layer}$	**10**	128	$\mathbf{73.86}_{0.94}$

Table 6: Comparing AutoPEFT to layer selection baselines with the same parameter budget on BERT$_{\text{large}}$. We report the Pfeiffer adapter for all 24 layers (`Serial`), specialised `AdapterDrop` (Rücklé et al., 2021) that inserts SA for the last 13 layers, and AA^{uni} (Moosavi et al., 2022) without its rational activation function with 13 selected layers (`Adaptable Adapter`). We run our AutoPEFT under the comparable search space of 24 layers and approximately match the size of `Serial`.

6 Conclusion

We proposed AutoPEFT, a novel search framework for automatically configuring parameter-efficient fine-tuning (PEFT) modules of pretrained language models. AutoPEFT features both a large and expressive, newly designed configuration *search space* and an effective *search method* featuring Bayesian optimization that discovers a Pareto-optimal set of novel PEFT configurations with promising performance-efficiency trade-offs. Empirically, we demonstrated that AutoPEFT-discovered configurations transfer strongly across different GLUE and SuperGLUE tasks, outperforming various strong PEFT baselines and being competitive to full model fine-tuning.

Limitations and Future Work

AutoPEFT search inevitably incurs a search cost since it requires iterative optimization at search time. However, we mitigate this by (i) using a low-fidelity proxy of 1-epoch training and (ii) leveraging strong transferability by generalising from low-resource and, thus, quick-to-train tasks. While the search itself can be seen as a *one-time* cost yielding a *permanent* well-performing and shareable configuration for particular tasks, we plan to delve deeper into further optimizing the search cost in future work.

Furthermore, while we conduct extensive experiments on the search space that contains three existing PEFT modules as building blocks, novel

PEFT modules may emerge. However, AutoPEFT framework is general, so we may easily integrate these forthcoming new modules. We defer thorough investigations to future work.

Acknowledgments

Han Zhou is supported by the UK Research and Innovation (UKRI) Frontier Research Grant EP/Y031350/1 (the UK government's funding guarantee for ERC Advanced Grants) awarded to Anna Korhonen at the University of Cambridge. Xingchen Wan is supported by the Clarendon Scholarship at the University of Oxford. The work has been supported in part by a Royal Society University Research Fellowship (no 221137; 2022-) awarded to Ivan Vulić, and by the UK EPSRC grant EP/T02450X/1. We thank TACL editors and anonymous reviewers for their constructive feedback that enabled us to strengthen our work.

References

Alan Ansell, Edoardo Ponti, Anna Korhonen, and Ivan Vulić. 2022. Composable sparse fine-tuning for cross-lingual transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1778–1796, Dublin, Ireland. Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.acl-long.125

Maximilian Balandat, Brian Karrer, Daniel Jiang, Samuel Daulton, Ben Letham, Andrew G. Wilson, and Eytan Bakshy. 2020. Botorch: A framework for efficient monte-carlo bayesian optimization. *Advances in Neural Information Processing Systems*, 33:21524–21538.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, Dublin, Ireland. Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.acl-short.1

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6–12, 2020, virtual.*

Guanzheng Chen, Fangyu Liu, Zaiqiao Meng, and Shangsong Liang. 2022a. Revisiting parameter-efficient tuning: Are we really there yet? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2612–2626, Abu Dhabi, United Arab Emirates Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.emnlp-main.168

Jiaao Chen, Aston Zhang, Xingjian Shi, Mu Li, Alex Smola, and Diyi Yang. 2023. Parameter-efficient fine-tuning design spaces. In *The Eleventh International Conference on Learning Representations.*

Shoufa Chen, Chongjian Ge, Zhan Tong, Jiangliu Wang, Yibing Song, Jue Wang, and Ping Luo. 2022b. Adaptformer: Adapting vision transformers for scalable visual recognition. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022.*

Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. 2021. Parallel bayesian optimization of multiple noisy objectives with expected hypervolume improvement. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6–14, 2021, virtual*, pages 2187–2200.

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. https://doi.org/10.18653/v1/N19-1423

Xuanyi Dong and Yi Yang. 2020. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26–30, 2020*.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017. https://doi.org/10.1007/978-3-030-05318-5_11

David Eriksson, Pierce I-Jen Chuang, Samuel Daulton, Peng Xia, Akshat Shrivastava, Arun Babu, Shicong Zhao, Ahmed A. Aly, Ganesh Venkatesh, and Maximilian Balandat. 2021. Latency-aware neural architecture search with multi-objective bayesian optimization. In *8th ICML Workshop on Automated Machine Learning (AutoML)*.

David Eriksson and Martin Jankowiak. 2021. High-dimensional bayesian optimization with sparse axis-aligned subspaces. In *Uncertainty in Artificial Intelligence*, pages 493–503. PMLR.

Peter I. Frazier. 2018. A tutorial on bayesian optimization. *CoRR*, abs/1807.02811v1. https://doi.org/10.48550/arXiv.1807.02811

Roman Garnett. 2023. *Bayesian Optimization*. Cambridge University Press. https://doi.org/10.1017/9781108348973

Demi Guo, Alexander Rush, and Yoon Kim. 2021 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. Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.acl-long.378

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Towards a unified view of parameter-efficient transfer learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25–29, 2022*.

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 *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9–15 June 2019, Long Beach, California, USA*, pages 2790–2799.

Edward J. Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022a. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Shengding Hu, Zhen Zhang, Ning Ding, Yadao Wang, Yasheng Wang, Zhiyuan Liu, and Maosong Sun. 2022b. Sparse structure search for delta tuning. In *Advances in Neural Information Processing Systems*.

Sergio Izquierdo, Julia Guerrero-Viu, Sven Hauns, Guilherme Miotto, Simon Schrodi, André Biedenkapp, Thomas Elsken, Difan Deng, Marius Lindauer, and Frank Hutter. 2021. Bag of baselines for multi-objective joint neural architecture search and hyperparameter optimization. In *8th ICML Workshop on Automated Machine Learning (AutoML)*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.emnlp-main.243

Liam Li and Ameet Talwalkar. 2019. Random search and reproducibility for neural architecture search. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22–25, 2019*, pages 367–377.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. 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 4582–4597. Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.acl-long.353

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019a. DARTS: Differentiable architecture search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6–9, 2019*.

Haokun Liu, Derek Tam, Muqeeth Mohammed, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *Advances in Neural Information Processing Systems*.

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. *CoRR*, abs/1907.11692v1.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6–14, 2021, virtual*, pages 1022–1035.

Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Scott Yih, and Madian Khabsa. 2022. UniPELT: A unified framework for parameter-efficient language model tuning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6253–6264, Dublin, Ireland. Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.acl-long.433

Nafise Moosavi, Quentin Delfosse, Kristian Kersting, and Iryna Gurevych. 2022. Adaptable adapters. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3742–3753, Seattle, United States. Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.naacl-main.274

Jonas Pfeiffer, Naman Goyal, Xi Lin, Xian Li, James Cross, Sebastian Riedel, and Mikel Artetxe. 2022. Lifting the curse of multilinguality by pre-training modular transformers. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3479–3495, Seattle, United States. Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.naacl-main.255

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020a. AdapterHub: A framework for adapting transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 46–54. Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-demos.7

Jonas Pfeiffer, Sebastian Ruder, Ivan Vulić, and Edoardo Ponti. 2023. Modular deep learning. *Transactions on Machine Learning Research*. Survey Certification. https://doi.org/10.48550/arXiv.2302.11529

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020b. MAD-X: An adapter-based framework for multi-task cross-lingual transfer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673. Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-main.617

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:140:1–140:67.

Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. 2021. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)*, 54(4):1–34. https://doi.org/10.1145/3447582

Bin Xin Ru, Xingchen Wan, Xiaowen Dong, and Michael A. Osborne. 2021. Interpretable neural architecture search via bayesian optimisation with weisfeiler-lehman kernels. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3–7, 2021.*

Robin Ru, Pedro M. Esperança, and Fabio Maria Carlucci. 2020. Neural architecture generator optimization. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6–12, 2020, virtual.*

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2021. AdapterDrop: On the efficiency of adapters in transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7930–7946, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.emnlp-main.626

Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M. Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal V. Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Févry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M. Rush. 2022. Multitask prompted training enables zero-shot task generalization. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25–29, 2022.*

Yi-Lin Sung, Varun Nair, and Colin Raffel. 2021. Training neural networks with fixed sparse masks. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6–14, 2021, virtual*, pages 24193–24205.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601. Florence, Italy. Association for Computational Linguistics. https://doi.org/10.18653/v1/P19-1452

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. 2023. DyLoRA: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 3274–3287, Dubrovnik, Croatia. Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.eacl-main.239

Ivan Vulić, Edoardo Maria Ponti, Robert Litschko, Goran Glavaš, and Anna Korhonen. 2020. Probing pretrained language models for lexical semantics. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7222–7240. Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-main.586

Xingchen Wan, Vu Nguyen, Huong Ha, Binxin Ru, Cong Lu, and Michael A. Osborne. 2021. Think global and act local: Bayesian optimisation over high-dimensional categorical and mixed search spaces. In *International Conference on Machine Learning*, pages 10663–10674. PMLR.

Xingchen Wan, Binxin Ru, Pedro M. Esperança, and Zhenguo Li. 2022. On redundancy and diversity in cell-based neural architecture search. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25–29, 2022.* OpenReview.net.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems 32: Annual Conference* *on Neural Information Processing Systems 2019, NeurIPS 2019, December 8–14, 2019, Vancouver, BC, Canada*, pages 3261–3275.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics. https://doi.org/10.18653/v1/W18-5446

Yaqing Wang, Sahaj Agarwal, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. 2022. AdaMix: Mixture-of-adaptations for parameter-efficient model tuning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5744–5760, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.emnlp-main.388

Colin White, Willie Neiswanger, and Yash Savani. 2021. BANANAS: Bayesian optimization with neural architectures for neural architecture search. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2–9, 2021*, pages 10293–10301. https://doi.org/10.1609/aaai.v35i12.17233

Saining Xie, Alexander Kirillov, Ross B. Girshick, and Kaiming He. 2019. Exploring randomly wired neural networks for image recognition. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 1284–1293. IEEE. https://doi.org/10.1109/ICCV.2019.00137

Antoine Yang, Pedro M. Esperança, and Fabio Maria Carlucci. 2020. NAS evaluation is frustratingly hard. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26–30, 2020*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*.

Mengjie Zhao, Tao Lin, Fei Mi, Martin Jaggi, and Hinrich Schütze. 2020. Masking as an efficient alternative to finetuning for pretrained language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2226–2241. Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-main.174

Han Zhou, Xingchen Wan, Ivan Vulić, and Anna Korhonen. 2023. Survival of the most influential prompts: Efficient black-box prompt search via clustering and pruning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 13064–13077, Singapore. Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.findings-emnlp.870

Eckart Zitzler and Lothar Thiele. 1998. Multiobjective optimization using evolutionary algorithms - A comparative case study. In *Parallel Problem Solving from Nature - PPSN V, 5th International Conference, Amsterdam, The Netherlands, September 27–30, 1998, Proceedings*, volume 1498 of *Lecture Notes in Computer Science*, pages 292–304. Springer. https://doi.org/10.1007/BFb0056872

Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings*.