Unlocking the Global Synergies in Low-Rank Adapters

Anonymous ACL submission

Abstract

Low-rank adaption (LoRA) has been the de-facto parameter-efficient fine-tuning technique for large language models. We present *HeteroLoRA*, a light-weight search algorithm that leverages zero-cost proxies to allocate the limited LoRA trainable parameters across the model for better fine-tuned performance. In addition to the allocation for the standard LoRA-adapted models, we also demonstrate the efficacy of HeteroLoRA by performing the allocation in a more challenging search space that includes LoRA modules and LoRA-adapted shortcut connections. Experiments show that HeteroLoRA enables improvements in model performance given the same parameter budge. For example, on RTE, we see an improvement of 6.7% in accuracy with a similar training parameter budget compared to a variety of state-of-the-art methods. We will open-source our algorithm once the paper is accepted.

1 Introduction

Recently, large language models (LLMs) have shown impressive performance in a range of natural language processing tasks (Qiu et al., 2020). Yet, fine-tuning pre-trained language models (PLMs) is computationally demanding and memory-intensive, and this problem is exacerbated by the ongoing trend of scaling up LLMs (Rae et al., 2022). To mitigate this, *parameter-efficient tuning* (PET) methods have been developed to fine-tune a small number of (extra) model parameters instead of the entire model (Houlsby et al., 2019).

Low-rank adaptation (LoRA) (Hu et al., 2021) is now the de-facto PET method. LoRA injects two low-rank matrices $A \in \mathbb{R}^{r \times d_{\text{in}}}$ and $B \in \mathbb{R}^{d_{\text{out}} \times r}$ with rank $r \ll \min(d_{\text{in}}, d_{\text{out}})$, to update the pre-trained weights $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. Unlike full fine-tuning, LoRA updates only the injected A and B with the pre-trained weights W unchanged. After fine-tuning, the update weights $\Delta W = BA$ fuse back to the pre-trained weights $W' = W + BA$, incurring no additional latency. LoRA achieves performance levels similar to full fine-tuning while drastically reducing memory usage. Yet, we identify the following limitations of LoRA.

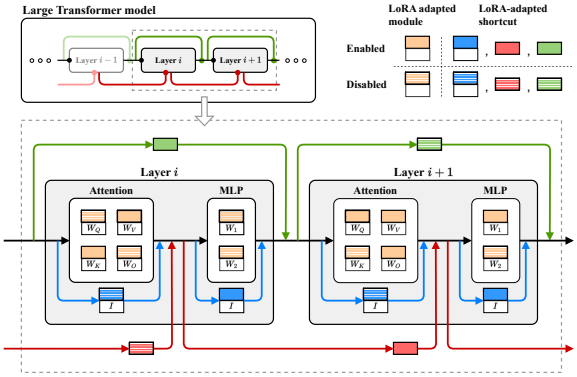


Figure 1: An illustration of the HeteroLoRA search space in a Transformer model. Given a fixed number of trainable parameters, HeteroLoRA finds an efficient heterogeneous LoRA configuration for a model on a specific task. Each of the standard LoRA modules and LoRA-adapted shortcuts can be enabled or disabled.

- Existing methods configure LoRA modules within a model uniformly with the same rank r, thus each LoRA module consumes an identical number of trainable parameters, regardless of its potentially varying contributions to the overall model performance.
- Current LoRA implementations predominantly adhere to the Transformer architecture. However, there has been limited exploration into extending the model architecture to enhance performance. This leads to the broader question of whether it is necessary to incorporate LoRA modules under these constraints and whether LoRA modules would be more effective with specific new connections, such as shortcut connections (He et al., 2015; Huang et al., 2018).

In this work, we introduce *HeteroLoRA*, a new lightweight framework designed to autonomously allocate the LoRA module across the entire LLM given a parameter budget. Furthermore, we perform HeteroLoRA within an expanded search space including LoRA-adapted shortcut connections (He et al., 2015) as illustrated in Figure 1. Specifically, we make the following contributions:

- We propose HeteroLoRA, a novel LoRA configuration search algorithm to solve the rank allocation problem within a limited trainable parameter budget. HeteroLoRA leverages zero-cost proxies (Abdelfattah et al., 2021) to avoid the high cost of brute-force search for finding effective allocations.
- We introduce static and dynamic versions of HeteroLoRA. The dynamic variant permits periodic enabling and disabling of LoRA modules. We empirically demonstrate that dynamic HeteroLoRA achieves superior performance through a flexible allocation scheme.
- We further prove the efficacy of the LoRA-adapted shortcut connection and combine it with HeteroLoRA to improve global synergies. The shortcuts suggested by HeteroLoRA enable more gains in model performance given the same parameter budget. For instance, on RTE, we see an improvement of 6.7% in accuracy with similar model size budgets.

2 Related Works

We explain the methodology of LoRA in Section 2.1, and examine previous uses of shortcut connections in Section 2.2. In Section 2.3, we summarise previous works in configuration searching on PET methods.

2.1 Low-Rank Adaptation

Conducting a full fine-tuning on an LLM is often parameter-inefficient, since every downstream task can require a large set of its own tuned parameters (Qiu et al., 2020). To resolve this issue, a series of special fine-tuning methods referred to as *parameter-efficient tuning* (PET) was studied. PET methods usually introduce a small set of extra trainable parameters ("adapter") to be trained on a downstream task with the pre-trained model parameters frozen.

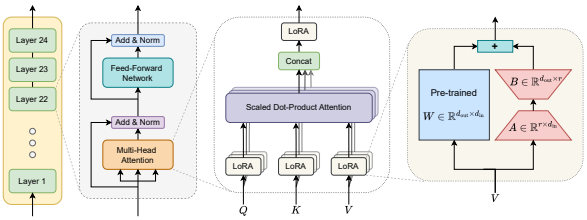


Figure 2: Overview of LoRA applied to a Transformer model. LoRA can be applied to each Transformer block. Taking a multi-head attention submodule in a block as an example, LoRA can be applied to linear projections transforming Q, K, V and the concatenated output. For each LoRA module, the input is transformed by both a pre-trained weight W and two low-rank matrices A and B, whose results are added and returned.

Low-rank adaptation (LoRA) (Hu et al., 2021) is a PET method that exploits the intrinsic low-rank structures in deep learning. Mathematically, LoRA hypothesises that the updates ΔW to the pre-trained weights during fine-tuning have a low "intrinsic rank". The update can be represented as the product of two rectangular low-rank decomposition matrices, yielding:

$$W_0 + \Delta W = W_0 + BA$$

where $W_0 \in \mathbb{R}^{d \times k}$ denotes the pre-trained matrix, $\Delta W \in \mathbb{R}^{d \times k}$ denotes the update, and $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$ with rank $r \ll \min(d, k)$ represents the low-rank decomposition matrices. During fine-tuning, only B and A are updated while W_0 is frozen. The updated B and A are fused into W_0 after fine-tuning.

2.2 Shortcut Connections

The residual connections in the Transformer architecture, as well as other types of shortcut connections, have been well explored to improve model performance.

ResNet (He et al., 2015) proposes residual connection for image processing tasks. The residual connection acts as an identity shortcut, taking the input to a function and adding it back to the function's output. By fitting a residual mapping directly, ResNet addresses the optimisation difficulty as model depth increases. *SENet* (Hu et al., 2019) introduces "squeeze-and-excitation" operation after a block of convolutional transformations to capture cross-channel relations. *DenseNet* (Huang et al., 2018) also offers an intuitive shortcut-adapted architecture. It divides a deep convolutional network into blocks based on feature map sizes. Within

each block, the input to the current layer includes outputs from all preceding layers. Unlike ResNet's residual connections, DenseNet concatenates feature maps from previous layers rather than summing them.

ResLoRA (Shi et al., 2024a), inspired by ResNet, incorporates three types of shortcuts into LoRA, namely input-shortcut, block-shortcut, and middle-shortcut. ResLoRA's experiments indicate that these LoRA-adapted shortcuts improve model performance over the original LoRA by reducing gradient vanishing and explosion.

2.3 Configuration Search in PET

There have been numerous PET methods showing performance improvements. However, determining the most suitable method and configuration for a given task can be complex and time-consuming. To address this challenge, various automated approaches to PET search have been proposed.

Prior research has applied neural architecture search (NAS) methods to automate the search for PET strategies. For instance, *AutoPEFT* (Zhou et al., 2023) utilised Bayesian optimisation (Frazier, 2018) to identify Pareto-optimal configurations for models adapted to Serial Adapters (Houlsby et al., 2019), Parallel Adapters (He et al., 2022), and prefix tuning (Li and Liang, 2021). Similarly, *S3PET* (Hu et al., 2022), employed DARTS (Liu et al., 2019a), a gradient-based NAS technique, across various PET methods. NAS4PET (Lawton et al., 2023) integrates parameter pruning into LoRA and BitFit (Zaken et al., 2022).

Specifically, several methods work on the automatic selection of an optimal rank for LoRA. In *DyLoRA* (Valipour et al., 2023), nested dropout is used to enforce ordered representations in the low-rank A and B, allowing simultaneous training of LoRA modules across a range of ranks r. This enables seamless switching between different rank configurations without the need for re-training. In contrast, *AdaLoRA* (Zhang et al., 2023) approximates the update matrix using singular value decomposition $\Delta W = P\Lambda Q$ such that singular values in Λ are pruned in each training step based on certain importance scores. Moreover, *AutoLoRA* (Zhang et al., 2024) expresses the update matrix as the summation of r rank-1 matrices and estimates the importance of these matrices. After fine-tuning, the optimal rank of each LoRA module is determined by thresholding the importance, then the model is re-trained with the optimal LoRA ranks.

Table 1: Comparison between HeteroLoRA and previous works over features including (a) single-shot NAS, where the search is accomplished in a single training run; (b) fixed GPU memory usage, where the searching process does not require more trainable parameters than the target parameter limitation; and (c) shortcut-enabled, where the search considers shortcut connections.

Features	Single-shot Search	Fixed GPU Memory Usage	Shortcut-enabled
AutoPEFT (Zhou et al., 2023)	✗	✓	✓
S3PET (Hu et al., 2022)	✓	✗	✗
NAS for PET (Lawton et al., 2023)	✗	✗	✗
DyLoRA (Valipour et al., 2023)	✓	✗	✗
AdaLoRA (Zhang et al., 2023)	✓	✓	✗
AutoLoRA (Zhang et al., 2024)	✗	✗	✗
ResLoRA (Shi et al., 2024a)	✗	✗	✓
HeteroLoRA (Ours)	✓	✓	✓

Table 2: Saliency Proxies for LoRA modules. We follow the definition of three zero-cost proxies, $s_{\texttt{snip}}(\cdot)$ for SNIP, $s_{\texttt{synflow}}(\cdot)$ for SYNFLOW, and $s_{\texttt{gradnorm}}(\cdot)$ for GRAD-NORM, to build the saliency scores for LoRA modules ($S_{\texttt{snip}}(\cdot)$, $S_{\texttt{synflow}}(\cdot)$, and $S_{\texttt{gradnorm}}(\cdot)$). A constant proxy is considered as random search baseline. Detailed introduction to zero-cost proxies (Abdelfattah et al., 2021) is included in Appendix C.

Proxy	Saliency score of LoRA-adapted module
Constant	$S_{\texttt{constant}}(M) = 1$
GRAD-NORM (Abdelfattah et al., 2021)	$S_{\texttt{gradnorm}}(M) = s_{\texttt{gradnorm}}(A) + s_{\texttt{gradnorm}}(B)$
SNIP (Lee et al., 2019)	$S_{\texttt{snip}}(M) = \sum_{\theta \in A} s_{\texttt{snip}}(\theta) + \sum_{\phi \in B} s_{\texttt{snip}}(\phi)$
SYNFLOW (Tanaka et al., 2020)	$S_{\texttt{synflow}}(M) = \sum_{\theta \in A} s_{\texttt{synflow}}(\theta) + \sum_{\phi \in B} s_{\texttt{synflow}}(\phi)$

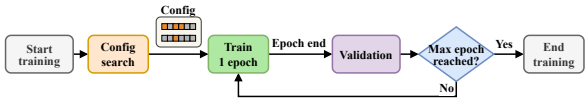
However, these methods either necessitate *computationally intensive full NAS, or require additional trainable parameters to determine the LoRA configuration*, while being confined to the existing architecture search space. We propose HeteroLoRA to address these limitations. Table 1 summarises the distinctions between HeteroLoRA and previous methods, highlighting the novelty of our work.

3 HeteroLoRA

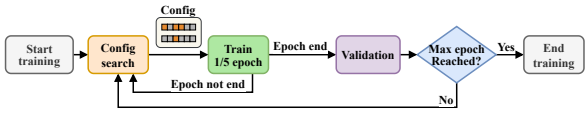
We adopt zero-cost proxies to estimate the importance of LoRA modules in Section 3.1, and discuss two ways to integrate the HeteroLoRA search into the existing PET pipeline in Section 3.2. In Section 3.3, we introduce LoRA-adapted shortcuts to enable exploration of global synergies.

3.1 Saliency Estimation using Proxies

Given a limited number of active LoRA modules, turning on a subset of modules could potentially be more effective. For instance, turning on all LoRA



(a) Static HeteroLoRA



(b) Dynamic HeteroLoRA

Figure 3: Training pipeline of (a) static HeteroLoRA, where LoRA modules are enabled/disabled at the start of training, and (b) dynamic HeteroLoRA, where LoRA modules are enabled/disabled periodically, *e.g.*, every $1/5$ epoch, during the training.

modules with $r = 2$ vs. turning on 25% of all modules with $r = 8$, the latter may achieve higher model performance. We consider such LoRA configuration assignment as a "LoRA rank allocation" problem and propose HeteroLoRA to solve it.

We estimate the saliency (importance) of a single LoRA module as a reference for HeteroLoRA searches. Modules with higher saliency scores will be enabled during training, and tie-breaking will be done by uniform random sampling. Three saliency proxies plus a random allocation baseline are shortlisted in Table 2. The detailed introduction to SNIP (Lee et al., 2019), SYNFLOW (Tanaka et al., 2020), and GRAD-NORM (Abdelfattah et al., 2021) are included in the Appendix C. It is worth mentioning that, as illustrated in Table 2, the saliency is applied to both A and B.

Note that the saliency proxies are applied to the whole LoRA W' instead of $\Delta W = BA$ for two reasons. First, at the start of training, the update component $\Delta W = BA$ is initialised as zeros, hence saliencies do not make sense by then. Second, the update component ΔW has a strong correlation with the pre-trained weight W, indicating that the features that ΔW amplified are already in W (Hu et al., 2021). Therefore, it is reasonable to include the pre-trained weight in the saliency for deciding the "on/off" of the LoRA modules.

3.2 Static and Dynamic HeteroLoRA

Static HeteroLoRA A straightforward way to incorporate HeteroLoRA search into the training pipeline is to compute the saliency proxy at the beginning of training, enabling or disabling LoRA modules accordingly. This is applied on a handful of training samples at the start that only introduces minimal search cost, taking around 2-5% of one epoch training. We then maintain the same rank allocation throughout training, as depicted in Figure 3a. This approach mirrors zero-cost NAS (Abdelfattah et al., 2021), where a lightweight search for optimal configurations is conducted initially, followed by complete fine-tuning.

Algorithm 1 Dynamic HeteroLoRA training

Require: $model$, $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{val}}$, $max\_epoch > 0$,
TRAIN(model, train_set),
VALIDATE(model, validation_set),
$saliency \in \{\texttt{CONSTANT}, \texttt{SNIP}, \texttt{SYNFLOW}, \texttt{GRAD-NORM}\}$,
$enable\_rate \in (0, 100]$, ▷ *percentage of LoRA and shortcut modules to be enabled*
HETEROLORA_SEARCH(model, saliency, enable_rate), ▷ *configuration search function*
$search\_T > 0$. ▷ *number of HeteroLoRA search to perform in every epoch*

$subepoch \leftarrow \left\lceil \frac{|\mathcal{D}_{\text{train}}|}{search\_T} \right\rceil$
$epoch \leftarrow 0$
while $epoch < max\_epoch$ **do**
  for $i := 0$ to $search\_T - 1$ **do**
    HETEROLORA_SEARCH($model$, $saliency$, $enable\_rate$) ▷ *find a LoRA rank allocation*
    TRAIN($model$, $\mathcal{D}_{\text{train}}[i \times subepoch : (i+1) \times subepoch]$) ▷ *train the model on a fraction of the training set*
  VALIDATE($model$, $\mathcal{D}_{\text{val}}$) ▷ *validate the model*
  $epoch \leftarrow epoch + 1$

Dynamic HeteroLoRA After several training steps, the optimizer may find that some enabled LoRA modules are not as important as measured initially. Therefore, we introduce dynamic HeteroLoRA, which periodically updates the rank allocation at the start of each training epoch, as shown in Figure 3b. Dynamic HeteroLoRA offers an opportunity to inspect the importance of each LoRA module through the frequency it has been enabled.

We detail the algorithms for both static HeteroLoRA (in Appendix D) and dynamic HeteroLoRA (in Algorithm 1). Algorithm 1 outlines the utilization of various saliency metrics by HeteroLoRA, where the search process is triggered $search\_T$ times within a single epoch of training. Dynamic HeteroLoRA then requires $search\_T \times max\_epochs$ times more search budget, however, we also show in later sections how this method provides a better fine-tuned model.

3.3 Extending the Search Space with LoRA-Adapted Shortcut Connections

We introduce LoRA-adapted shortcut connections to extend the search space, which is later integrated with HeteroLoRA to foster global synergies between LoRA modules. A LoRA-style low-rank

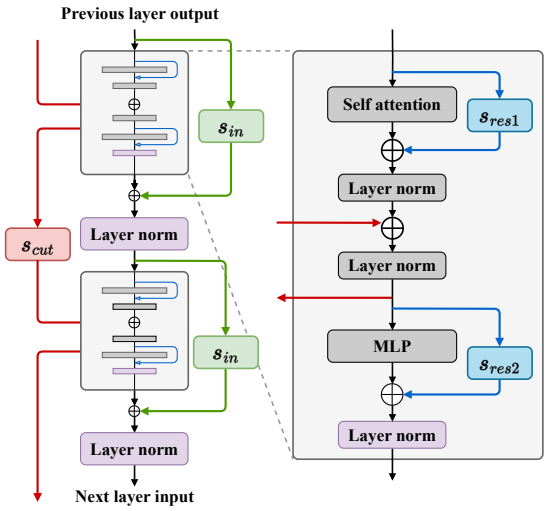


Figure 4: LoRA-adapted shortcut architecture on two Transformer layers with post-layer-normalisation. Blue blocks are the residual shortcuts s_{res1} and s_{res2}. Green blocks are the s_{in} same-layer style "cross-layer" shortcuts. Red blocks are the s_{cut} "cut-layer" style cross-layer shortcuts. These shortcuts, combined with the standard LoRA modules present in the layers, constitute the HeteroLoRA search space.

linear transformation is applied to each shortcut:

$$W = W_0 + \frac{\alpha}{r} BA$$

where W_0 is the initial weight of the linear projection, depending on the type of the shortcut; α is a pre-defined scaling factor; and r is the rank of A and B. A and B are initialised similarly to LoRA modules to ensure $W = W_0$ at the start of training. We refer to a combination of $\langle W_0, A, B \rangle$ as a "shortcut module". A layer normalisation (Ba et al., 2016) is appended after the addition of the shortcut to improve the training stability in post-layer-normalisation models (*e.g.*, OPT-350M). This is, however, not employed in pre-layer-normalisation models (*e.g.*, RoBERTa and Llama3) as we have observed enhanced performance without the extra layer normalisation.

As shown in Figure 4, we focus on two types of shortcut connections, residual shortcut and cross-layer shortcut, to keep the search space tractable:

1. Residual shortcut: Shortcut is applied to the micro-architecture by replacing the two original residual connections within the Transformer block (as the blue blocks in Figure 4). We refer to them as "residual shortcuts" s_{res1} and s_{res2}. The initial weight W_0 of these shortcut modules is the identity matrix I.

2. Cross-layer shortcut: Shortcut connection is applied to the macro-architecture by linking two points at different Transformer blocks. We refer to this as the "cross-layer shortcut". A cross-layer shortcut skips multiple Transformer blocks. We concentrate on cross-layer shortcuts that skip one Transformer block for simplicity, as the green blocks (s_{in}) and the red blocks (s_{cut}) in Figure 4. The initial weights W_0 of the cross-layer shortcut modules are initialised to zeros because these shortcuts do not exist in the original architecture. More details are summarised in Appendix E.

4 Experiments

We outline our basic experiment setup in Section 4.1. In Section 4.2, we identify the most promising saliency proxy and verify that HeteroLoRA achieves a better performance than the standard homogeneous LoRA. In Section 4.3, we demonstrate that LoRA-adapted shortcuts enable an additional performance gain. In Section 4.4 we use HeteroLoRA to allocate the rank in a search space that includes both standard LoRA moduels and LoRA-adapted shortcuts, highlighting the efficacy of HeteroLoRA. Section 4.5 subsequently evaluates the performance of Dynamic HeteroLoRA within the shortcut-enabled search space relative to state-of-the-art methods.

4.1 Experiment Setup

We perform experiments with OPT-350M (Zhang et al., 2022), RoBERTa-base, and RoBERTa-large (Liu et al., 2019b) on the GLUE benchmark (Wang et al., 2019), which contains English natural language understanding (NLU) tasks. For each experiment, the mean and standard deviation of the performance are calculated over three independent runs of different random seeds. We apply LoRA to the query layer W^Q and the value layer W^V as experiments show that applying these LoRA modules to these two layers effectively improves model performance (see Appendix A). We use a fixed number of trainable parameters for all groups in the same experiment subsection. To make our experimental results reproducible, detailed rank and training hyperparameters are summarised in Appendix B and our code would be open-sourced if the paper is accepted.

Table 3: Performance of the salience proxies with static and dynamic HeteroLoRA. Accuracy on MRPC and the difference with the baseline performance are reported. The baseline, in which all LoRA modules are enabled with rank $r = 2$, achieves 83.8% accuracy. We observe that the combination of GRAD-NORM and dynamic HeteroLoRA achieves the highest accuracy.

$r = 8$	CONSTANT	GRAD-NORM	SNIP	SYNFLOW
Static	83.8 (+0.0)	82.8 (-1.0)	82.8 (-1.0)	78.7 (-5.1)
Dynamic	82.4 (-1.4)	**84.1 (+0.3)**	82.4 (-1.4)	82.4 (-1.4)

Table 4: Performance gain of LoRA-adapted shortcut connections. L-only is the LoRA-only baseline. $r_S = r_L$ denotes the model in which both LoRA modules and LoRA-adapted shortcuts are applied with the same rank. $r_S > r_L$ represent the model in which the LoRA module rank is fixed and the rest of ranks are allocated to the shortcut. We observe that the LoRA-adapted shorcuts combined with LoRA modules achieve higher accuracy than the LoRA-only group given the same number of trainable parameters.

#Trainable	Group	MRPC	RTE	SST-2	Avg.
2.3M	L-only	83.4 ± 1.1	72.1 ± 1.2	93.4 ± 0.4	92.1
	$r_S = r_L$	**84.6 ± 0.5**	**73.5 ± 1.4**	**93.9 ± 0.3**	**92.6**
	$r_S > r_L$	84.6 ± 1.9	70.9 ± 2.2	93.7 ± 0.3	92.4
9.4M	L-only	83.4 ± 0.4	69.4 ± 3.2	93.3 ± 0.5	91.9
	$r_S = r_L$	83.8 ± 0.6	**72.6 ± 1.3**	**94.1 ± 0.7**	**92.7**
	$r_S > r_L$	**84.1 ± 0.9**	71.8 ± 3.0	93.8 ± 0.3	92.5

4.2 Determining the Proxy and Comparing Static and Dynamic HeteroLoRA

We first apply HeteroLoRA to the original LoRA. Table 3 compares the four proxies (Constant, Grad-Norm, SNIP and Synflow) under both Static and Dynamic HeteroLoRA on OPT-350M. In each experiment, 25% of LoRA modules are enabled with $r = 8$. The eight combinations are also compared to a baseline, in which all LoRA modules are enabled with $r = 2$, so the numbers of trainable parameters are the same.

We observe that *Dynamic HeterLoRA achieves better performances than static HeteroLoRA*, with GRAD-NORM performing the best and surpassing the baseline. Therefore, we use Dynamic HeteroLoRA with the GRAD-NORM proxy in the following experiments.

4.3 The Performance Gain of Enabling Shortcut Connections

We conduct controlled experiments in two different configurations to validate the effectiveness of the extended search space with shortcut connections described in Section 3.3. Table 4 presents three different search configurations (L-only, $r_S = r_L$ and $r_S > r_L$) on OPT-350M under two different number of trainable parameters constraints:

- L-only: Only the standard LoRA is applied to the model. This group serves as the baseline.
- $r_S = r_L$: Both standard LoRA and LoRA-adapted shortcuts are applied to the model and the rank of shortcuts is the same as the standard LoRA.
- $r_S > r_L$: Both LoRA and shortcut are applied, but the LoRA module rank is fixed and the rest of the ranks are allocated to the shortcut.

We ensure that the three groups have the same number of trainable parameters for a fair comparison. The detailed experiment setup is in Appendix B. Table 4 demonstrates that the *shortcut-adapted architecture generally outperforms the LoRA-only architecture*, meanwhile with a more prominent advantage as the budget grows larger (at 9.4M trainable parameters). This observation indicates that the linear projections on the shortcuts have larger "intrinsic ranks" than the LoRA update matrices. When performance "saturates" in LoRA modules, shortcuts foster further performance improvement by developing global synergies across layers.

Table 5: Dynamic HeteroLoRA combined with LoRA-adapted shortcuts. The S (LoRA-adapted Shortcut) & L (Standard LoRA) baselines have all modules enabled. The DH (Dynamic HeteroLoRA) & S & L have 25% of LoRA models and LoRA-adapted shortcuts enabled. They have the same number of trainable parameters for a fair comparison. We observe that DH & S & L outperforms the baseline, meaning HeteroLoRA finds a more optimal rank allocation.

#Trainable	Setup	MRPC	RTE	SST-2	Avg.
2.3M	S & L	83.7 ± 0.8	**73.4 ± 2.2**	93.6 ± 0.5	83.5
	DH & S & L	**84.3 ± 1.0**	72.9 ± 1.8	**93.9 ± 0.5**	**83.7**
9.4M	S & L	84.6 ± 0.5	73.5 ± 1.4	**93.8 ± 0.4**	84.0
	DH & S & L	**85.0 ± 1.6**	**73.6 ± 1.1**	93.6 ± 0.1	**84.1**

4.4 Dynamic HeteroLoRA with an Extended Search Space

Finally, we integrate the components and design choices discussed above in Section 4.2 and Section 4.3. We combine Dynamic HeteroLoRA and the GRAD-NORM proxy.

Table 5 compares the LoRA-adapted OPT-350M model with/without Dynamic HeteroLoRA under the same trainable parameter budget:

Table 6: Search time costs of HeteroLoRA configuration searches for an OPT-350M model training on SST-2, where the number of trainable parameters is constrained to 2.3M. Percentage is the search time relative to the total training time.

Method	Static HeteroLoRA	Dynamic HeteroLoRA
Time cost	50 seconds	200 seconds per epoch
Percentage	0.25%	7.75%

- S & L denotes all the LoRA modules and LoRA-adapted shortcuts are enabled with the same rank.
- For DH & S & L, Dynamic HeteroLoRA sorts the saliency scores of standard LoRA and LoRA-adapted shortcuts to determine which module to enable/disable.

As illustrated in Table 5, we observe that Dynamic HeteroLoRA further improves model performance over S & L, indicating that HeteroLoRA finds a more optimal rank allocation. Figure 5 displays the frequency of each LoRA or shortcut module being enabled over the 20 training epochs on MRPC and the 10 training epochs on SST-2. The frequency of each LoRA module denotes its importance to performance; the frequency of each shortcut module characterises its efficacy to global synergies. A noticeable preference for value projections over query projections indicates that the value transformation updates generally contribute more to the performance. Move results on different datasets are available in Appendix F.

As stated in Section 3.2, Static HeteroLoRA incurs minimal search cost at the start of PEFT, whereas Dynamic HeteroLoRA bears a higher cost due to its periodic search during PEFT. Table 6 reveals that the cost for Dynamic HeteroLoara, although very low, extends the training duration marginally (by 7.75%). It's also important to note that the relative overhead is comparable for other models and datasets, since this cost is proportional to the total training expenditure.

4.5 Comparing with SoTAs

One piece of work that is related to our Dynamic HeteroLoRA is ResLoRA (Shi et al., 2024b), which also incorporated the residual connections into the LoRA trainable region but they have conducted an allocation manually for the adaptors. We then focus on a comparison with ResLoRA in Table 7 on the RoBERTa-large model. It is worth mentioning that ResLoRA manually assigns trainable parameters to certain blocks, and is, bs and ms means the shortcuts are added to inputs (input-shortcuts), blocks (block-shortcuts) or middle (middle-shortcuts). In contrast, we incorporate these design options in a search space as explained in Section 3.3. Table 7 illustrates that Dynamic HeteroLoRA can effectively navigate the extended search space, evidenced by a notable performance increase over the manually designed ResLoRA methods within a similar PEFT design space. On these three considered datasets (CoLA, RTE and MRPC), Dynamic HeteoLoRA outperforms the best ResLoRA design by 0.3 in terms of averaged accuracy.

Table 7: Comparing with ResLoRA on RoBERTa-large, where the model size is constrained to 0.4M trainable parameters for both ResLoRA and Dynamic HeteroLoRA.

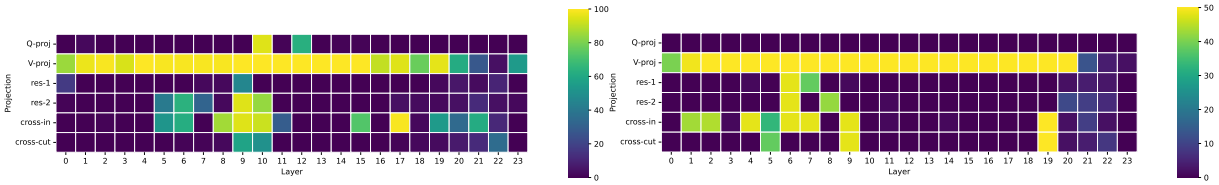
Method	CoLA	RTE	MRPC	Avg.
ResLoRA$_{is}$	65.5	83.0	92.4	80.3
ResLoRA$_{bs}$	65.4	82.3	91.3	79.7
ResLoRA$_{ms}$	65.8	82.0	91.6	79.8
Dynamic HeteroLoRA	66.1	85.9	90.0	80.7

Another research direction related to HeteroLoRA involves various LoRA configuration search algorithms, including AutoLoRA (Zhang et al., 2024) and AdaLoRA (Zhang et al., 2023). We then pivot to a comparative analysis with these methods in Table 8 focusing on RoBERTa-base, as this is a model that these methods report performance on. In addition to AdaLoRA and AutoLoRA, we also report the standard LoRA's performance at a rank value $r = 8$ as a baseline. In Table 8, all methods are kept to have the same number of trainable parameters (0.3M). We evaluated performance across eight different downstream tasks and have shown that HeteroLoRA outperforms other LoRA configuration search methods.

5 Discussion

5.1 Proxies for PET

The concept of employing saliency metrics to identify critical components of a neural network is well-established in the field of Efficient AI and has been extensively researched in domains like network compression (to determine which parts to prune (Lee et al., 2019)) and network architecture search (to design networks guided by saliency measures (Abdelfattah et al., 2021)). Though it was natural to anticipate that proven proxies like SNIP (Lee et al., 2019) and Synflow (Tanaka et al.,



(a) MRPC (b) SST-2

Figure 5: Frequency of linear projections in every model layer being enabled in the Dynamic HeteroLoRA trained on (a) MRPC and (b) SST-2 for LoRA and shortcut modules. A noticeable preference for value projections over query projections indicates that the value update generally contributes more to the fine-tuned performance.

Table 8: Comparing with AdaLoRA (Zhang et al., 2023) and AutoLoRA (Zhang et al., 2024) on RoBERTa-base on different GLUE downstream tasks, where the model size is constrained to 0.3M trainable parameters.

Method	Params	MNLI	SST-2	MRPC	CoLA	QNLI	RTE	QQP	Avg
Full fine-tuning	125M	86.7	94.8	89.3	61.6	92.8	76.9	90.3	84.6
LoRA ($r = 8$)	0.3M	86.9	94.5	89.1	59.0	92.9	75.8	89.6	84.0
AdaLoRA	0.3M	87.0	94.0	89.4	58.8	93.0	75.9	89.9	84.0
AutoLoRA	0.3M	87.0	94.9	89.4	61.3	92.9	77	90.3	84.7
Dynamic HeteroLoRA	0.3M	85.9	**94.9**	**89.7**	**64.5**	91.5	**82.5**	89.6	**85.5**

2020) would be advantageous, given their effectiveness in pruning-at-initialisation and zero-cost NAS, our results in Section 4.2 revealed that the simple Grad-Norm proxy is more effective for PET configuration search. This finding suggests that proxy designs for PET might differ from those previously proven effective in other domains, and there is potential for future research to develop more specialised and mathematically grounded proxies tailored for PET search.

5.2 The Impact of PET on Different Modules

A common practice in PET, particularly when employing LoRA, is to uniformly apply trainable modules with the same rank value, for instance, using a consistent rank r. Our findings depicted in Figure 6 indicate that Dynamic HeteroLoRA generally favours assigning trainable parameters to the value projection layers in transformers. Notably, HeteroLoRA also tends to allocate resources to residual layers, particularly in the middle layers. Conversely, we observe that HeteroLoRA infrequently assigns training resources to the query projection layers. Intuitively, the calculations within a transformer layer can be seen as constructing weighted attention for each entry in the value sequence. Consequently, for PET, modifying the value projection may be the most straightforward way for fine-tuning the network to downstream tasks. We believe these observations might provide useful insights for future research in the direction of novel PET methods or even novel transformer architectures.

6 Conclusions

We propose dynamic HeteroLoRA, a framework automatically determining the "on/off" for the LoRA modules in LLM fine-tuning. Then we verify that LoRA-adapted shortcuts improve model performance. In the end, we demonstrate that Dynamic HeteroLoRA effectively solves the rank allocation problem in a challenging search space including both LoRA modules and shortcuts. HeteroLoRA offers a cost-effective way to allocate trainable parameters within a limited training budget.

Limitations

In comparison to the original LoRA method and other extensions to LoRA such as ResLoRA, the shortcut connections in our method do introduce additional computational and memory overhead during inference, although the overhead is relatively subtle. The HeteroLoRA configuration search also inevitably incurs additional time costs. However, we mitigate this by employing zero-cost proxies and dynamic configuration search.

Furthermore, a set of more fine-grained rank values for LoRA modules, rather than enable/disable, can be considered in configuration searches. However, a more sophisticated allocation problem may be involved.

References

Mohamed S. Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas D. Lane. 2021. Zero-cost proxies for lightweight nas. *Preprint*, arXiv:2101.08134.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *Preprint*, arXiv:1607.06450.

Peter I. Frazier. 2018. A Tutorial on Bayesian Optimization. *Preprint*, arXiv:1807.02811.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Towards a unified view of parameter-efficient transfer learning. *Preprint*, arXiv:2110.04366.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *Preprint*, arXiv:1512.03385.

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. *Preprint*, arXiv:1902.00751.

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. *Preprint*, arXiv:2106.09685.

Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. 2019. Squeeze-and-excitation networks. *Preprint*, arXiv:1709.01507.

Shengding Hu, Zhen Zhang, Ning Ding, Yadao Wang, Yasheng Wang, Zhiyuan Liu, and Maosong Sun. 2022. Sparse Structure Search for Parameter-Efficient Tuning. *Preprint*, arXiv:2206.07382.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2018. Densely connected convolutional networks. *Preprint*, arXiv:1608.06993.

Neal Lawton, Anoop Kumar, Govind Thattai, Aram Galstyan, and Greg Ver Steeg. 2023. Neural Architecture Search for Parameter-Efficient Fine-tuning of Large Pre-trained Language Models. *Preprint*, arXiv:2305.16597.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. 2019. Snip: Single-shot network pruning based on connection sensitivity. *Preprint*, arXiv:1810.02340.

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

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019a. DARTS: Differentiable Architecture Search. *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. 2019b. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *Preprint*, arXiv:1907.11692.

XiPeng Qiu, TianXiang Sun, YiGe Xu, YunFan Shao, Ning Dai, and XuanJing Huang. 2020. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, 63(10):1872–1897.

Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimpoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew Johnson, Blake Hechtman, Laura Weidinger, Iason Gabriel, William Isaac, Ed Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stanway, Lorrayne Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. 2022. Scaling language models: Methods, analysis & insights from training gopher. *Preprint*, arXiv:2112.11446.

Shuhua Shi, Shaohan Huang, Minghui Song, Zhoujun Li, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, and Qi Zhang. 2024a. Reslora: Identity residual mapping in low-rank adaption. *Preprint*, arXiv:2402.18039.

Shuhua Shi, Shaohan Huang, Minghui Song, Zhoujun Li, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, and Qi Zhang. 2024b. Reslora: Identity residual mapping in low-rank adaption. *Preprint*, arXiv:2402.18039.

Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins, and Surya Ganguli. 2020. Pruning neural networks without any data by iteratively conserving synaptic flow. *Preprint*, arXiv:2006.05467.

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. *Preprint*, arXiv:2210.07558.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Glue: A multi-task benchmark and analysis platform for natural language understanding. *Preprint*, arXiv:1804.07461.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2022. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *Preprint*, arXiv:2106.10199.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *Preprint*, arXiv:2303.10512.

Ruiyi Zhang, Rushi Qiang, Sai Ashish Somayajula, and Pengtao Xie. 2024. Autolora: Automatically tuning matrix ranks in low-rank adaptation based on meta learning. *Preprint*, arXiv:2403.09113.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. Opt: Open pre-trained transformer language models. *Preprint*, arXiv:2205.01068.

Han Zhou, Xingchen Wan, Ivan Vulić, and Anna Korhonen. 2023. AutoPEFT: Automatic Configuration Search for Parameter-Efficient Fine-Tuning. *Preprint*, arXiv:2301.12132.

A Primary LoRA Experiments

To make the search space manageable, we first decide which linear layers in attention are helpful if they are LoRA-adapted. We sweep the LoRA combinations and fine-tune them on MRPC. As shown in Tables 9 and 10, we find that applying LoRA to W_Q and W_V is most helpful for improving model performance.

Table 9: Performance of LoRA applied to different combinations of linear projections in the attention submodule under the same numbers of trainable parameters. Adapting W^V and (W^Q, W^V) perform the best.

Projections	W^Q	W^K	W^V	W^O	W^Q, W^K	W^Q, W^V	W^Q, W^K, W^V, W^O
Rank r	8	8	8	8	4	4	2
MRPC (acc)	80.2±1.6	80.3±0.5	**84.3±1.3**	83.2±0.1	81.6±1.1	**84.1±1.1**	83.3±1.8

Table 10: Comparison of LoRA applied to attention linear projections, FFN linear projections, and both. Due to the larger feed-forward network module dimension, adapting FFN projections W_1 or W_2 costs more trainable parameters than the attention projections.

Projections	W^Q, W^V	W_1	W_2	W_1, W_2	W^Q, W^V, W_1, W_2
Rank r	4	8	8	4	2
# Trainable	394K		984K		689K
MRPC (acc)	**84.1±1.1**	83.3±0.6	82.3±1.2	84.0±0.9	83.5±0.5

B Hyperparameters

Experiments are run on a server with 4 NVIDIA QUADRO RTX 8000 GPUs, as well as an HPC cluster with servers of 3 NVIDIA A100 40 GB RAM Tensor Core GPUs, 2 AMD EPYC 7742 (Rome) 2.25 GHz 64-core processors, and 512 GB DDR4-3200 RAM.

For experiments with different model configurations and different datasets, the optimal training hyperparameters may vary. Due to the expensive computational cost, searching for the optimal hyperparameters in every experiment is infeasible. Therefore, learning rate searches are conducted for experiments while other hyperparameters are set by referencing the original RoBERTa paper (Liu et al., 2019b) and LoRA paper (Hu et al., 2021).

For experiments on the LoRA-searching training pipelines, the optimal hyperparameters tuned independently by beam search on *MRPC* are applied to all datasets, except for the learning rate, which is searched on each dataset respectively.

B.1 LoRA modules combined with LoRA-Adapted Shortcuts

In the dynamic HeteroLoRA training, a search is conducted every $1/5$ training epoch, in which the `GRAD-NORM` saliency score is evaluated over 32 batches of training data for each LoRA or shortcut module, and the modules ranking top 25% are enabled with rank $r = 8$ until the next HeteroLoRA search.

As shortcut connections have shown their effectiveness at larger ranks in Section 4.3, we further evaluate dynamic HeteroLoRA with LoRA and shortcut modules enabled with rank $r = 32$. The two HeteroLoRA training setups are compared to two baselines with all modules enabled but with $1/4$ ranks respectively, so the numbers of trainable parameters at any time in the training are equivalent respectively.

B.2 Training hyperparameters

Experiments were run on three random seeds (0, 13, 42), and the averaged results were reported. Table 11 and Table 12 display the training hyperparameters. The learning rates were determined through hyperpa-

Table 11: The hyperparameters for experiments on the shortcut-adapted OPT-350M model.

Method	Dataset	MRPC	RTE	SST-2
	Optimiser	AdamW		
OPT-350M with shortcuts	Batch size	8		
	# Epochs	20	20	10
	Learning rate	2e-4	2e-4	1e-4
	Weight decay	0.01		
	Max seq. len.	512		
	LoRA config.	$r_Q = r_V = 8$		
	LoRA α	16		
	Shortcut config.	$r_{\text{res1}} = r_{\text{res2}} = r_{\text{in}} = r_{\text{cut}} = 8$		
	Shortcut α	4		

Table 12: The hyperparameters for experiments on shortcut-adapted RoBERTa models.

Method	Dataset	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE
	Optimiser	AdamW						
RoBERTa-base	Batch size	16	16	16	32	32	16	32
	# Epochs	30	60	30	80	25	25	80
	Learning rate	5e-5	5e-4	4e-4	4e-4	4e-4	5e-4	5e-4
	Weight decay	0.01						
	Max seq. len.	512						
	LoRA config.	$r_Q = r_V = 16$						
	LoRA α	8						
	Shortcut config.	$r_{\text{res1}} = r_{\text{res2}} = r_{\text{in}} = r_{\text{cut}} = 16$						
	Shortcut α	8						
RoBERTa-large	Batch size	4	4	4	4	4	4	8
	# Epochs	10	10	20	20	10	20	20
	Learning rate	3e-4	4e-4	4e-4	2e-4	2e-4	3e-4	5e-4
	Weight decay	0.01						
	Max seq. len.	128	128	512	128	512	512	512
	LoRA config.	$r_Q = r_V = 32$						
	LoRA α	16						
	Shortcut config.	$r_{\text{res1}} = r_{\text{res2}} = r_{\text{in}} = r_{\text{cut}} = 32$						
	Shortcut α	16						

rameter searches from 1e-5 to 5e-3; other hyperparameters were decided according to previous works Liu et al. (2019b) and Hu et al. (2021).

B.3 Saliency hyperparameters

For comparison between saliency proxies, the following hyperparameters of the proxies were used:

- `CONSTANT`: no hyperparameter required.
- `SNIP`: the score was evaluated on the first 32 batches from the training set.
- `SYNFLOW`: no hyperparameter required.
- `GRAD-NORM`: the score was evaluated on the first 32 batches from the training set.

Experiments were conducted to compare different choices of the number of training batches to use for `SNIP` and `GRAD-NORM`. On MRPC, using 8 batches, 32 batches, and the entire training set produced the same training curve. Following the previous work (Abdelfattah et al., 2021), we used 32 training batches for the later experiments.

Furthermore, two hyperparameters were involved in the HeteroLoRA strategies:

- Enable rate: the percentage of LoRA and shortcut modules enabled at any time in the training was set to 25%.
- Frequency of dynamic HeteroLoRA search: for all experiments, the HeteroLoRA search was conducted 5 times per training epoch. Further ablation experiment results are shown in Appendix F.

The scaling factor α of shortcut modules was searched across 1 to 16. The best hyperparameters and configurations, as in Tables 11 and 12, were used for the two series of shortcut-adapted models in Section 4.3 and the dynamic HeteroLoRA training in Sections 4.4 and 4.5 unless particularly specified.

C Zero-Cost Proxies

The detailed definition of zero-cost proxies for the LoRA-adapted module/shortcut and trainable parameters are defined as follows

C.1 CONSTANT

A baseline proxy is designed as assigning score $S_{\texttt{constant}}(M) = 1$ to every LoRA module M. This enforces tie-breaking on all LoRA modules, so uniform random sampling is performed in every HeteroLoRA search.

C.2 SNIP

The SNIP (Lee et al., 2019) proxy aims to find the elements that degrade the performance the least when removed. It uses a weight mask $C \in \{0, 1\}^m$ applied to each block of parameters, with 0 at the positions of disabled parameters and 1 at the position of active parameters, and computes the loss gradient to the mask variables over a few minibatches of training data $\mathcal{D}$:

$$s_{\texttt{snip}}(\theta) = \frac{\partial \mathcal{L}(\mathcal{D}; C \odot W)}{\partial c_\theta}$$

where c_θ denotes the weight mask variable corresponding to parameter θ. In HeteroLoRA, since the LoRA rank allocation regards each LoRA module as a unit, we fill the weight mask C for each LoRA module with ones, and extend the saliency of a single parameter $s(\theta)$ the saliency of a LoRA module M by summation:

$$\begin{aligned} S_{\texttt{snip}}(M) &= \sum_{\theta \in M} s_{\texttt{snip}}(\theta) \\ &= \sum_{\theta \in A} s_{\texttt{snip}}(\theta) + \sum_{\phi \in B} s_{\texttt{snip}}(\phi) \end{aligned}$$

C.3 SYNFLOW

A minibatch of inputs of ones is fed to the model with weights taken as their absolute values. The SYNFLOW (Tanaka et al., 2020) score computes the product of a parameter value and the gradient of the sum of the losses on the minibatch to the parameter:

$$s_{\texttt{synflow}}(\theta) = \theta \cdot \frac{\partial \left(\sum_{\text{minibatch}} \mathcal{L}(\mathbb{1}; |W|)\right)}{\partial \theta}$$

We also extended SYNFLOW of a single parameter to the saliency of a LoRA module by summation:

$$\begin{aligned} S_{\texttt{synflow}}(M) &= \sum_{\theta \in M} s_{\texttt{synflow}}(\theta) \\ &= \sum_{\theta \in A} s_{\texttt{synflow}}(\theta) + \sum_{\phi \in B} s_{\texttt{synflow}}(\phi) \end{aligned}$$

C.4 GRAD-NORM

A minibatch of training data is fed to the model, and GRAD-NORM computes the Euclidean norm of the loss gradients on a block of parameters:

$$s_{\texttt{gradnorm}}(W) = \left\| \frac{\partial \mathcal{L}(\mathcal{D}; W)}{\partial W} \right\|_2$$

This marks how sensitive the loss is to each block of parameters. We extend this to the saliency of a LoRA module by taking the sum of GRAD-NORM over the matrices:

$$S_{\texttt{gradnorm}}(M) = s_{\texttt{gradnorm}}(A) + s_{\texttt{gradnorm}}(B)$$

D Static HeteroLoRA Algorithm

In Algorithm 2, we present the algorithm of static HeteroLoRA, where as explained previously, the search operation occurs only once at the start of training.

Algorithm 2 Static HeteroLoRA training

Require: $model$, $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{val}}$, $max\_epoch > 0$,
TRAIN(model, train_set),
VALIDATE(model, validation_set),
$saliency \in \{$CONSTANT, SNIP, SYNFLOW, GRAD-NORM$\}$,
$enable\_rate \in (0, 100]$, ▷ *percentage of LoRA and shortcut modules to be enabled*
HETEROLORA_SEARCH(model, saliency, enable_rate). ▷ *configuration search function*

HETEROLORA_SEARCH($model$, $saliency$, $enable\_rate$) ▷ *find a LoRA rank allocation*
$epoch \leftarrow 0$
while $epoch < max\_epoch$ **do**
  TRAIN($model$, $\mathcal{D}_{\text{train}}$) ▷ *train the model*
  VALIDATE($model$, $\mathcal{D}_{\text{val}}$) ▷ *validate the model*
  $epoch \leftarrow epoch + 1$

E LoRA-Adapted Shortcuts

The detailed definition of LoRA-adapted shortcuts is as follows.

Cross-Layer Shorcut This type of shortcut forwards the model's hidden state as:

$$h_{i+1} = s(h_i) + f_i(h_i)$$

where h_i denotes the input hidden state to the ith layer of modules, f_i denotes the function of the ith layer, and s denotes the current shortcut linear transformation. The function of the layer f_i, however, does not necessarily need to be an exact Transformer block as:

$$f_{\text{in}}(h) = \text{MLP}_i\left(\text{Attention}_i(h)\right)$$

but can also be a "layer" recomposed by the sub-modules cutting across a Transformer block boundary, like:

$$f_{\text{cut}}(h) = \text{Attention}_{i+1}(\text{MLP}_\text{i}(h))$$

The two corresponding styles of cross-layer shortcuts, referred to as s_{in} and s_{cut}, are employed in our shortcut-adapted models (as the green blocks and the red blocks in Figure 4).

LayerNorm Inserted After Shortcut Layer normalisation sometimes needs to be performed after the cross-layer shortcut output is merged into the original hidden state. In this project, the shortcuts are applied to the OPT-350M model, which employs post-layer-normalisation Transformer architecture (layer normalisation is performed after the original residual connection is merged back). For cross-layer shortcuts, given that the original layer normalisation of OPT-350M performs an element-wise affine transformation with pre-trained weights, performing another layer normalisation without affine transformation after the original will impact the original layer normalisation's effect, meanwhile, training new weights for a new affine transformation merely on a downstream dataset will be ineffective. Therefore, we re-perform the original layer normalisation after the cross-layer shortcut output is merged back to the hidden state.

Consequently, the original output hidden state h_{i+1} of the ith layer:

$$\begin{aligned} a_i &= \text{LN}_{1,i}\left(h_i + \text{Attn}_i(h_i)\right) \\ h_{i+1} &= \text{LN}_{2,i}\left(a_i + \text{FFN}_i(a_i)\right) \end{aligned}$$

is transformed by the shortcut connections into:

$$\begin{aligned} a_i &= \text{LN}_{1,i}\left[\text{LN}_{1,i}\left(s_{\text{res1},i}(h_i) + \text{Attn}_i(h_i)\right) + s_{\text{cut},i}(a_{i-1})\right] \\ h_{i+1} &= \text{LN}_{2,i}\left[\text{LN}_{2,i}\left(s_{\text{res2},i}(a_i) + \text{FFN}_i(a_i)\right) + s_{\text{in},i}(h_i)\right] \end{aligned}$$

where $s_{\text{res1},i}, s_{\text{res2},i}, s_{\text{in},i}, s_{\text{cut},i}$ denote the linear projections on the shortcuts, $\text{LN}_{1,i}$ and $\text{LN}_{2,i}$ represent the two layer normalisation layers in the ith layer, Attn_i represents the attention submodule, and FFN_i represents the feed-forward network submodule.

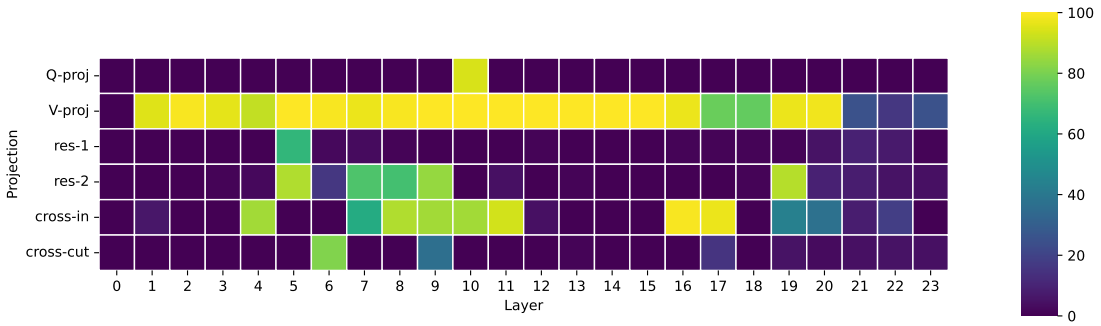
F Additional Experimental Results

Dynamic HeteroLoRA is experimented on MRPC, RTE and SST-2 with the same setup as in Section 4.4. Figure 6, Figure 7 and Figure 8 demonstrate the frequency of each LoRA or shortcut module being enabled over 20 and 10 training epochs on RTE and SST-2, respectively.

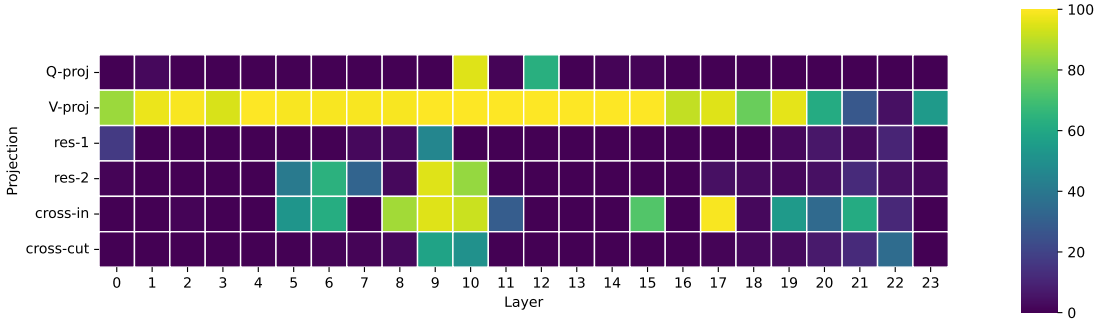
Intuitively, frequent LoRA configuration searches give more chances to explore the configuration search space, while a long search interval allows the chosen configuration to be fully trained. Table 13 shows the performance on MRPC and RTE of dynamic HeteroLoRA with various configuration search frequencies. As we can see, no particular performance pattern across the search frequency can be easily observed.

Table 13: Dynamic HeteroLoRA with different HeteroLoRA search frequencies per training epoch.

LoRA Ranking Method	Combined Allocation				Separated Allocation			
Search Freq (per epoch)	10	5	2	1	10	5	2	1
MRPC (acc)	84.6	84.3	84.1	84.3	84.6	83.7	84.6	85.1
RTE (acc)	72.3	72.9	72.3	72.7	74.5	72.8	70.1	66.9

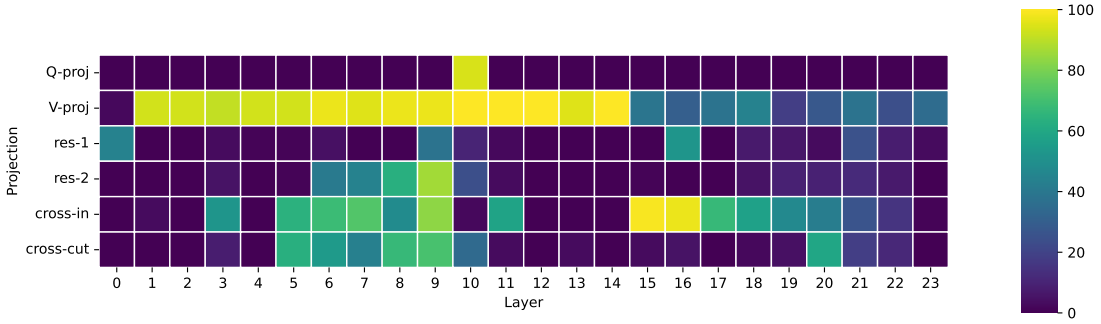


(a) Dynamic HeteroLoRA with modules at $r = 8$

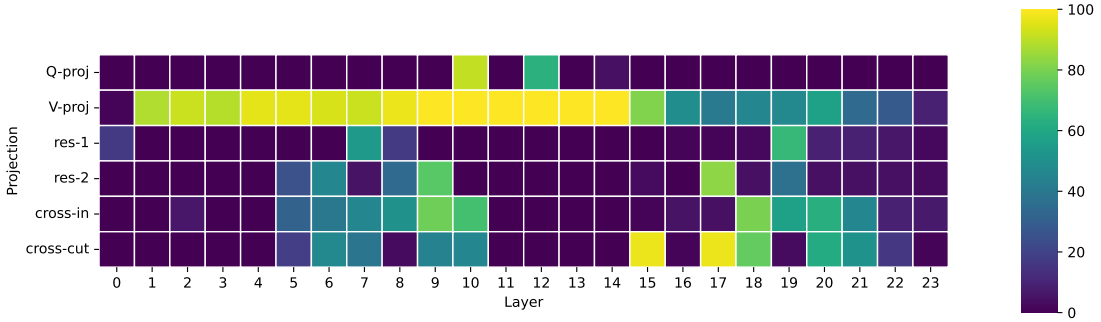


(b) Dynamic HeteroLoRA with modules at $r = 32$

Figure 6: Frequency of linear projections in every model layer being enabled in the dynamic HeteroLoRA training on MRPC with (a) $r = 8$ and (b) $r = 32$ for LoRA and shortcut modules.

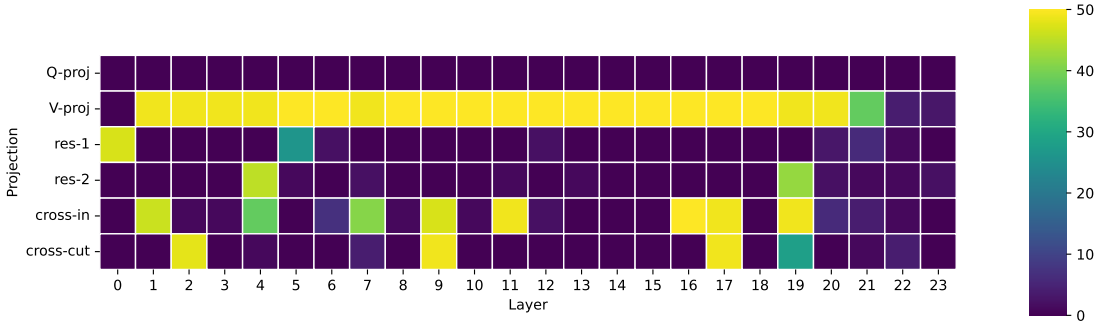


(a) Dynamic HeteroLoRA with modules at $r = 8$

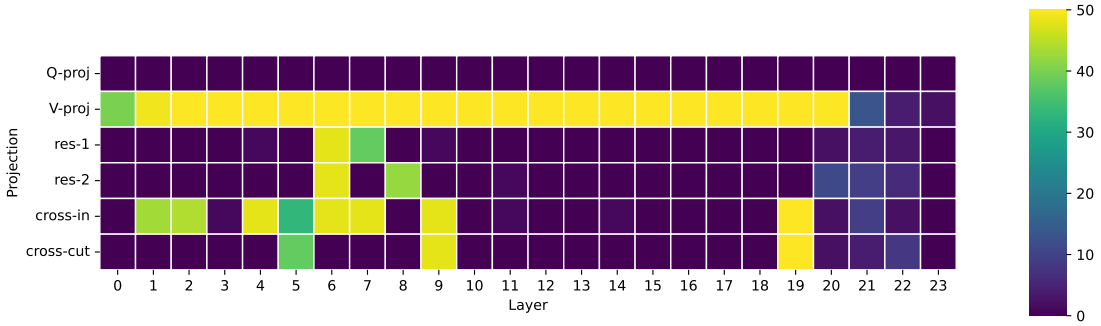


(b) Dynamic HeteroLoRA with modules at $r = 32$

Figure 7: Frequency of linear projections in every model layer being enabled in the dynamic HeteroLoRA training on RTE with (a) $r = 8$ and (b) $r = 32$ for LoRA and shortcut modules.



(a) Dynamic HeteroLoRA with modules at $r = 8$



(b) Dynamic HeteroLoRA with modules at $r = 32$

Figure 8: Frequency of linear projections in every model layer being enabled in the dynamic HeteroLoRA training on SST-2 with (a) $r = 8$ and (b) $r = 32$ for LoRA and shortcut modules.