Unlocking the Global Synergies in Low-Rank Adapters

Zixi Zhang [1] **Cheng Zhang** [2] **Xitong Gao** [3] **Robert D. Mullins** [1] **George A. Constantinides** [2] **Yiren Zhao** [2]

Abstract

Low-rank Adaption (LoRA) has been the de-facto parameter-efficient fine-tuning technique for large language models. We present *HeteroLoRA*, a lightweight 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 budget. For example, on MRPC, we see an improvement of 1.6% in accuracy with similar training parameter budget. We have open-sourced our algorithm.

1. Introduction

Recently, large language models (LLMs) have shown impressive performance in a range of natural language processing tasks (Qiu et al., 2020). However, fine-tuning pre-trained language models (PLMs) is computationally and memory-intensive. 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 $W' = W + AB$, incurring no additional latency. LoRA achieves performance levels similar to full fine-tuning while drastically reducing memory usage. We identify the following limitations of LoRA.

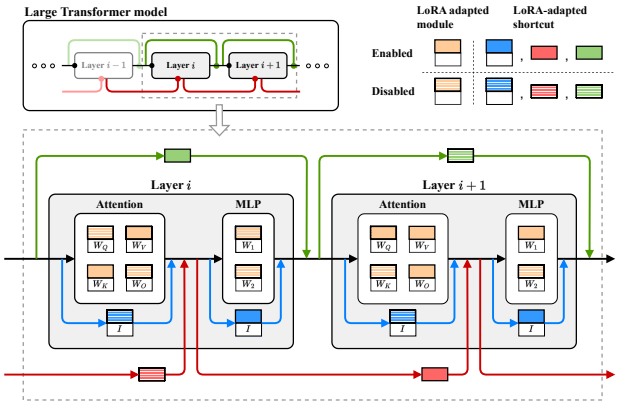


Figure 1. An illustration of the HeteroLoRA search space in a Transformer model. Given a fixed number of trainable parameters, HeteroLoRA finds an efficient heterogeneous LoRA configuration for a model on a specific task. Each of the standard LoRA modules and LoRA-adapted shortcuts can be enabled or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

[1]Department of Computer Science, University of Cambridge, Cambridge, United Kingdom [2]Department of Electrical and Eletronic Engineering, Imperial College London, London, United Kingdom [3]Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Guangdong, China. Correspondence to: Zixi Zhang <zz458@cam.ac.uk>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

Table 1. 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)	$S_{\texttt{synflow}}(M) = \sum_{\theta \in A} s_{\texttt{synflow}}(\theta) + \sum_{\phi \in B} s_{\texttt{synflow}}(\phi)$

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.
- 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 MRPC, we see an improvement of 1.6% in accuracy with similar model size budgets. Our project is publicly available at `https://github.com/ZixiBenZhang/lora_global_synergies`.

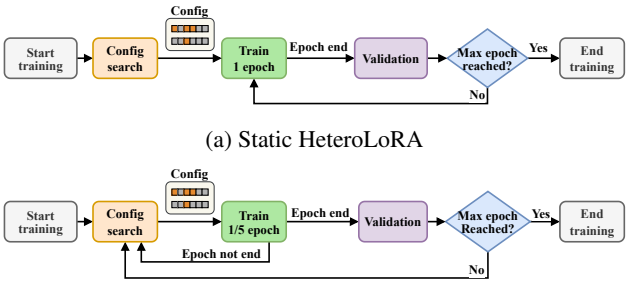
2. HeteroLoRA

We adopt zero-cost proxies to estimate the importance of LoRA modules in Section 2.1, and discuss two ways to integrate the HeteroLoRA search into the existing PET pipeline in Section 2.2. In Section 2.3, we introduce LoRA-adapted shortcuts to enable exploration of global synergies.

2.1. Saliency Estimation using Zero-Cost 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 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 1. 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.



(a) Static HeteroLoRA

(b) Dynamic HeteroLoRA

Figure 2. 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.

Note that the saliency proxies are applied to the whole LoRA module 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 measurements for deciding the "on/off" of the LoRA modules.

2.2. Static or Dynamic?

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 usually applied on a handful of training samples at the start that only introduces minimal search cost, taking around 10% of one epoch training. We then maintain the same rank allocation throughout training, as depicted in Figure 2a. 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.

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 2b. Dynamic HeteroLoRA offers an opportunity to inspect the importance of each LoRA module through the

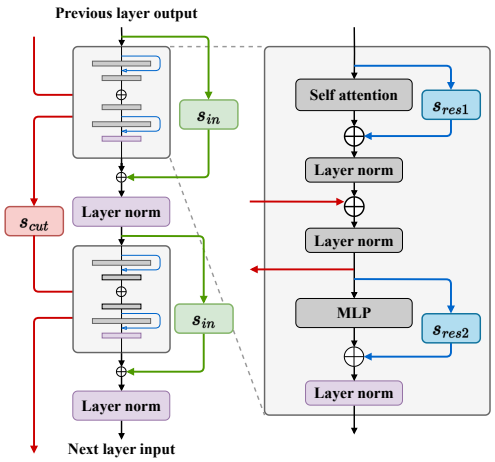


Figure 3. LoRA-adapted shortcut architecture on two Transformer layers with post-layer-normalisation. Blue blocks are the residual shortcuts s_{res1} and s_{res2}. Green blocks are the s_{in} same-layer style "cross-layer" shortcuts. Red blocks are the s_{cut} "cut-layer" style cross-layer shortcuts.

frequency it has been enabled.

2.3. 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 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.

As shown in Figure 3, 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 3). 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 3. 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.

3. Experiments

We outline our basic experiment setup in Section 3.1. In Section 3.2, we identify the most promising saliency proxy and verify that HeteroLoRA achieves a better performance than the standard homogeneous LoRA. In Section 3.3, we demonstrate that LoRA-adapted shortcuts enable an additional performance gain. In Section 3.4 we use HeteroLoRA to allocate the rank in a search space that includes both standard LoRA modules and LoRA-adapted shortcuts, highlighting the efficacy of HeteroLoRA.

3.1. Experiment Setup

We perform the experiments on OPT-350M (Zhang et al., 2022) over GLUE subsets MRPC, RTE, and SST-2 (Wang et al., 2019). For each experiment, the median and standard deviation of the performance are calculated over five 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. Detailed rank and training hyperparameters are summarised in Appendix B.

3.2. Determining Proxy and Training Strategy of HeteroLoRA

Table 2 shows the comparison of the four proxies under static and dynamic HeteroLoRA. 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 HeteroLoRA 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.

3.3. Verifying Performance Gain of Shortcuts

We conduct controlled experiments in various configurations to validate the effectiveness of shortcut connections. Table 3 presents two shortcut setups under 2.3M and 9.4M trainable parameters:

- L-only: Only the standard LoRA is applied to the model. This group serves as the baseline.

Table 2. 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 3. 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

- $r_S = r_L$: Both standard LoRA and LoRA-style 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 to the model, 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 included in Appendix B. We observe that the shortcut-adapted architecture generally outperforms the LoRA-only architecture, meanwhile with a more prominent advantage as the budget becomes larger. This observation indicates that the linear projections on the shortcuts have larger "intrinsic ranks" than the LoRA update matrices. When performance has "saturated" in LoRA modules, shortcuts foster further performance improvement by developing global synergies across layers.

3.4. Dynamic HeteroLoRA with Extended Search Space

Finally, we integrate the components discussed above. We combine Dynamic HeteroLoRA and GRAD-NORM, as explored in Section 3.2, thereby embracing the expanded search space as inferred from the findings in Section 3.3.

Table 4 compares the LoRA-adapted model with/without Dynamic LoRA under the same trainable parameter budget:

Table 4. Dynamic HeteroLoRA combined with LoRA-adapted shortcuts. The S & L baselines have all modules enabled. The DH & 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 than the homogeneous baseline in a challenging search space including both LoRA modules and shortcuts.

Rank budget	Setup	MRPC	RTE	SST-2	Avg.
r=2 for S & L	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**
r=8 for S & L	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**

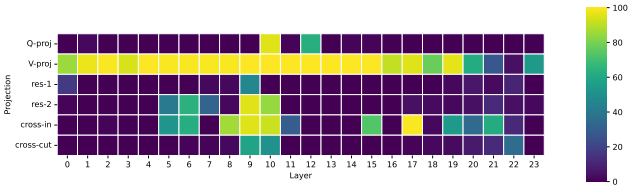


Figure 4. Frequency of linear projections in every model layer being enabled in the dynamic HeteroLoRA trained on MRPC for LoRA and shortcut modules. A noticeable preference for value projections over query projections indicates that the value update generally contributes more to the fine-tuned performance.

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

As illustrated in Table 4, we observe that dynamic HeteroLoRA further improves model performance over S & L, indicating the HeteroLoRA finds a more optimal rank allocation. Figure 4 displays the frequency of each LoRA or shortcut module being enabled over the 20 training epochs on MRPC. 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. More results are available in Appendix E.

4. Conclusions

We propose dynamic HeteroLoRA, a framework that automatically determines 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 demon-

strate 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.

References

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

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp, 2019.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models, 2021.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

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

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach, 2019.

XiPeng Qiu, TianXiang Sun, YiGe Xu, YunFan Shao, Ning Dai, and XuanJing Huang. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, 63(10):1872–1897, September 2020. ISSN 1869-1900. doi: 10.1007/s11431-020-1647-3. URL http://dx.doi.org/10.1007/s11431-020-1647-3.

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

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

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. Opt: Open pre-trained transformer language models, 2022.

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 5 and 6, we find that applying LoRA to W_Q and W_V is most helpful for improving model performance.

Table 5. 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 6. 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

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., 2019) 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 3.3, 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 7 display the training hyperparameters. The learning rates were determined through hyperparameter searches from 1e-5 to 5e-3; other hyperparameters were decided according to previous works (Liu et al., 2019; 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.

Table 7. The hyperparameters for training on datasets from the GLUE benchmark.

Method	Dataset	MNLI	MRPC	QQP	RTE	SST-2
	Optimiser	AdamW				
OPT-350M LoRA	Batch size	8				
	# Epochs	10	20	10	20	10
	Learning rate	1e-4	2e-4	2e-4	3e-4	1e-4
	Weight decay	0.01				
	Max Seq. Len.	512				
	LoRA config.	$r_Q = r_V = 8$				
	LoRA α	16				

Table 8. The hyperparameters for experiments on shortcut-adapted models.

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		

- `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 in Section 3.2 and Section 3.4, HeteroLoRA search was conducted 5 times per training epoch. Further ablation experiment results are shown in Appendix E.

The training hyperparameters in Section 3.2 followed Table 7 except the learning rates, which were searched across 5e-5 to 1e-3 respectively for each proxy and training strategy.

The learning rates for the shortcut-adapted models were searched from 5e-5 to 5e-4. Moreover, the scaling factor α of shortcut modules was searched across 1 to 16. The best hyperparameters and configurations, as in Table 8, were used for the two series of shortcut-adapted models in Section 3.3 and the dynamic HeteroLoRA training in Section 3.4 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. 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 3).

LayerNorm Inserted After Shortcut Layer normalisation 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 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.

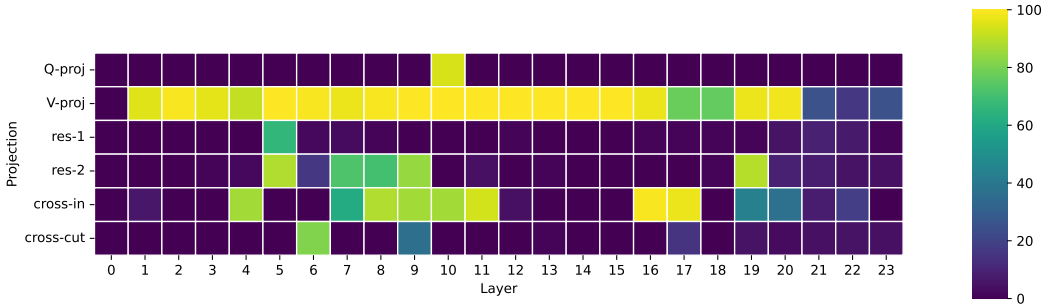
E. Additional Experimental Results

Dynamic HeteroLoRA is experimented on RTE and SST-2 with the same setup as in Section 3.4. Figure 5, Figure 6 and Figure 7 demonstrate the frequency of each LoRA or shortcut module being enabled over 20, 20 and 10 training epochs on MRPC, RTE and SST-2, respectively.

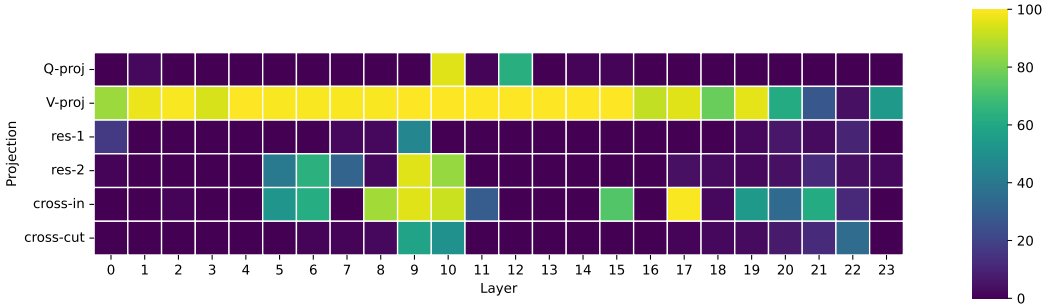
Intuitively, frequent LoRA configuration searches give more chances to explore the configuration search space, while a long search interval allows the chosen configuration to be fully trained. Table 9 shows the performance on MRPC and RTE of dynamic HeteroLoRA with various configuration search frequencies. As we can see, no particular performance pattern across the search frequency can be easily observed.

Table 9. 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

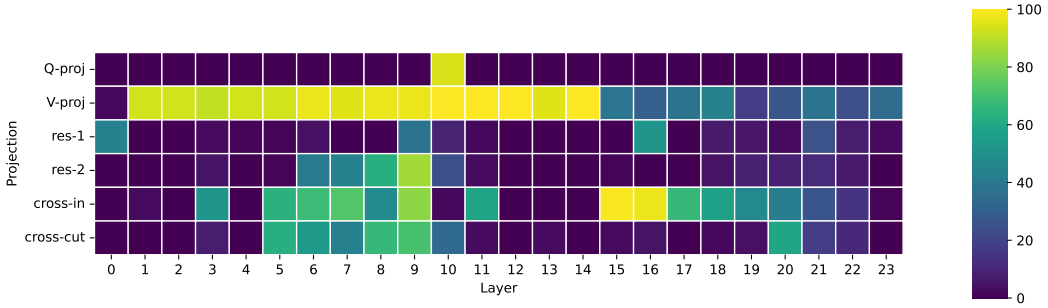


(a) Dynamic HeteroLoRA with modules at $r = 8$

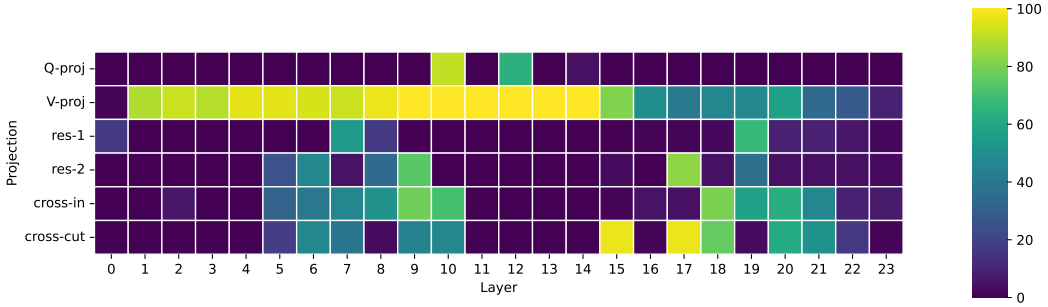


(b) Dynamic HeteroLoRA with modules at $r = 32$

Figure 5. 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.

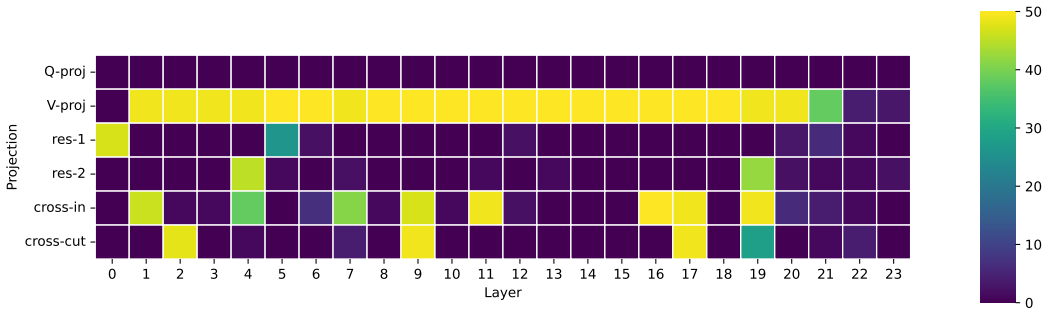


(a) Dynamic HeteroLoRA with modules at $r = 8$

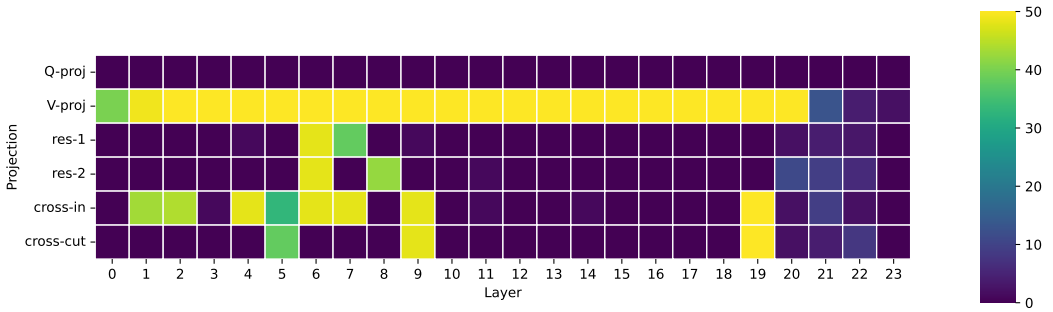


(b) Dynamic HeteroLoRA with modules at $r = 32$

Figure 6. 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 7. 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.