AdaDHP: Fine-Grained Fine-Tuning via Dual Hadamard Product and Adaptive Parameter Selection

Han Liu[1], **Changya Li**[1], **Xiaotong Zhang**[1*], **Feng Zhang**[2],
Fenglong Ma[3], **Wei Wang**[4], **Hong Yu**[1]
[1]Dalian University of Technology, Dalian, China, [2]Peking University, Beijing, China
[3]The Pennsylvania State University, Pennsylvania, USA
[4]Shenzhen MSU-BIT University, Shenzhen, China
{liu.han.dut, lichangya.dut, zfeng.maria}@gmail.com, zxt.dut@hotmail.com,
fenglong@psu.edu, ehomewang@ieee.org, hongyu@dlut.edu.cn

Abstract

With the continuously expanding parameters, efficiently adapting large language models to downstream tasks is crucial in resource-limited conditions. Many parameter-efficient fine-tuning methods have emerged to address this challenge. However, they lack flexibility, like LoRA requires manually selecting trainable parameters and rank size, $(IA)^3$ can only scale the activations along columns, yielding inferior results due to less precise fine-tuning. To address these issues, we propose a novel method named AdaDHP with fewer parameters and finer granularity, which can adaptively select important parameters for each task. Specifically, we introduce two trainable vectors for each parameter and fine-tune the parameters through Hadamard product along both rows and columns. This significantly reduces the number of trainable parameters, with our parameter count capped at the lower limit of LoRA. Moreover, we design an adaptive parameter selection strategy to select important parameters for downstream tasks dynamically. This allows our method to flexibly remove unimportant parameters for downstream tasks. Finally, we demonstrate the superiority of our method on the T5-base model across 17 NLU tasks and on complex mathematical tasks with the Llama series models.

1 Introduction

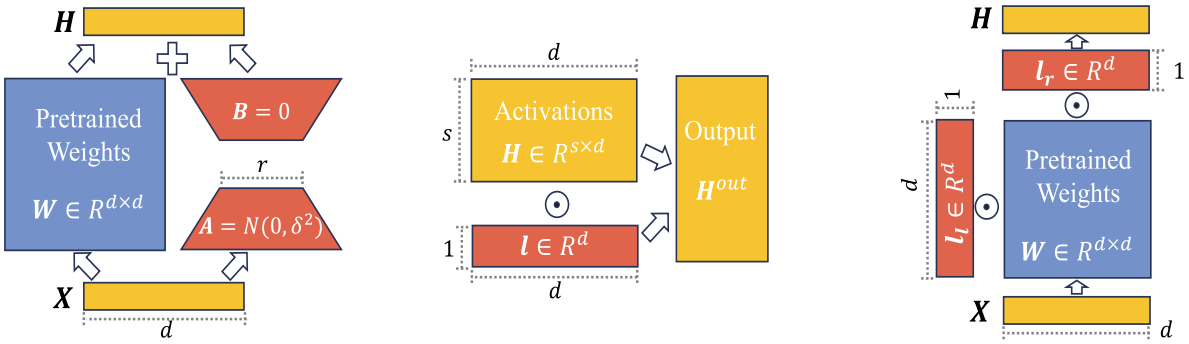
With the advent of large language models (LLMs) (Brown et al., 2020; Touvron et al., 2023a,b), there has been a migration of various natural language processing tasks towards the pre-training and fine-tuning paradigm. The straightforward approach involves appending task-specific modules and subsequently fine-tuning all parameters for downstream tasks. However, fine-tuning LLMs with escalating number of parameters poses a challenge under resource-constrained environments. Moreover, fine-tuning each downstream task requires the retention of all weights, leading to significant resource wastage. Consequently, parameter-efficient fine-tuning (PEFT) methods have emerged as a requisite solution to address the constraints of GPU and storage resources.

The PEFT methods achieve high efficiency by fine-tuning a small subset of parameters or introducing additional trainable parameters. Adapter (Houlsby et al., 2019) is an early-stage method that appends a bottleneck layer following multi-head attention and feed-forward blocks, updating only these added parameters and keeping the original backbone fixed. While yielding promising results, it is afflicted by inference latency. Another paradigm, prompt tuning (Liu et al., 2021; Li and Liang, 2021; Lester et al., 2021), employs trainable prompts to each task and only fine-tunes these prompts. The placement of these prompts varies, resulting in diverse approaches. However, the additional prompts extend the context length, requiring the storage of larger intermediate activations during training. Consequently, these methods may demand larger GPU resources compared to full fine-tuning.

Currently, LoRA series methods (Hu et al., 2022; Zhang et al., 2023b) have become prevalent for fine-tuning LLMs. These techniques introduce trainable low-rank decomposed matrices, denoted as $\mathbf{A}$ and $\mathbf{B}$, for the parameters $\mathbf{W}$, as shown in Figure 1(a). Due to the low rank of $\mathbf{A}$ and $\mathbf{B}$, the number of trainable parameters can be significantly decreased. Moreover, LoRA allows merging the incremental matrix $\Delta\mathbf{W}$ with $\mathbf{W}$ after training, addressing the issue of inference latency. However, the original LoRA method requires the manual selection of trainable parameters and rank size, thereby limiting its flexibility. Another noteworthy approach, $(IA)^3$ (Liu et al., 2022), as shown in Figure 1(b), updates activations directly through element-wise multiplication with a vector $\mathbf{l}$ during the forward

*Corresponding author.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 9492–9504



(a) LoRA (b) $(IA)^3$ (c) AdaDHP

Figure 1: Illustration of LoRA, $(IA)^3$, and our method AdaDHP. Blue for pretrained weights, yellow for inputs or activations, and red for additional trainable parameters. LoRA adds trainable low-rank matrices to weights, $(IA)^3$ applies trainable scaling to activations, while AdaDHP introduces two trainable vectors for each parameter, enabling row and column dot products.

propagation process. $(IA)^3$ with minimal trainable parameters achieves remarkable results. Nonetheless, it applies the same coefficient to each column of the activations and is equivalent to scale the weight matrices along column, significantly constraining the granularity of fine-tuning. Additionally, it offers lower flexibility as it fine-tunes the same activations for all tasks.

To handle the aforementioned issues simultaneously, we introduce a novel PEFT method named **AdaDHP**, which achieves fine-grained and flexible tuning by **D**ual **H**adamard **P**roduct and **Ada**ptive parameter selection. As shown in Figure 1(c), we introduce two trainable vectors for each parameter, enabling row and column dot products, respectively. Unlike $(IA)^3$ that only applies weight transformation on the row, we allow for different transformations on both the row and column of weight matrices. Additionally, we maintain the parameter size with the LoRA rank of 1. Secondly, we introduce a parameter importance metric and a budget allocation mechanism to adaptively select crucial parameters for downstream tasks. By pruning unimportant or redundant parameters based on their importance scores, we can enhance parameter efficiency and the flexibility of parameter selection. Finally, we validate our method on 17 NLP tasks and complex mathematical tasks, affirming its current state-of-the-art performance.

2 Related Work

The current PEFT methods can be primarily categorized into three types: partial fine-tuning, additive fine-tuning and hybrid fine-tuning methods.

2.1 Partial Fine-Tuning

Partial fine-tuning methods only update those critical parameters in pre-trained model for downstream task. For instance, the BitFit (Zaken et al., 2022) method only updates the biases and the task-specific layer, keeping major parameters unchanged. Additionally, pretrained weight masking methods (Zhao et al., 2020) use criteria such as thresholds and Fisher information (Sung et al., 2021) to evaluate the importance of parameters, which allow for only updating important weights.

2.2 Additive Fine-Tuning

Additive fine-tuning methods integrate trainable modules into pre-trained model while keeping the backbone frozen. Adapter-based approaches are early attempts. HAdapter (Houlsby et al., 2019) integrates trainable adapter modules into each block of the Transformer, and solely fine-tunes these additive modules for downstream tasks. Other adapter-based methods (He et al., 2022; Rücklé et al., 2021) optimize the integration and efficiency in distinct ways. Prompt-based approaches add specific trainable prompt tokens for each task, which are added at different locations in different methods. For instance, Prompt-tuning (Lester et al., 2021) inserts learnable prompt tokens into the model input, while Prefix-tuning (Li and Liang, 2021) adds soft prompts before the hidden states of the multi-head attention layer. However, the adapter-based methods incur inference latency, while prompt-based methods are memory-intensive.

Another widely recognized approach, LoRA (Hu et al., 2022), reduces trainable parameters by decomposing $\mathbf{W}$ into two low-rank matrices. But

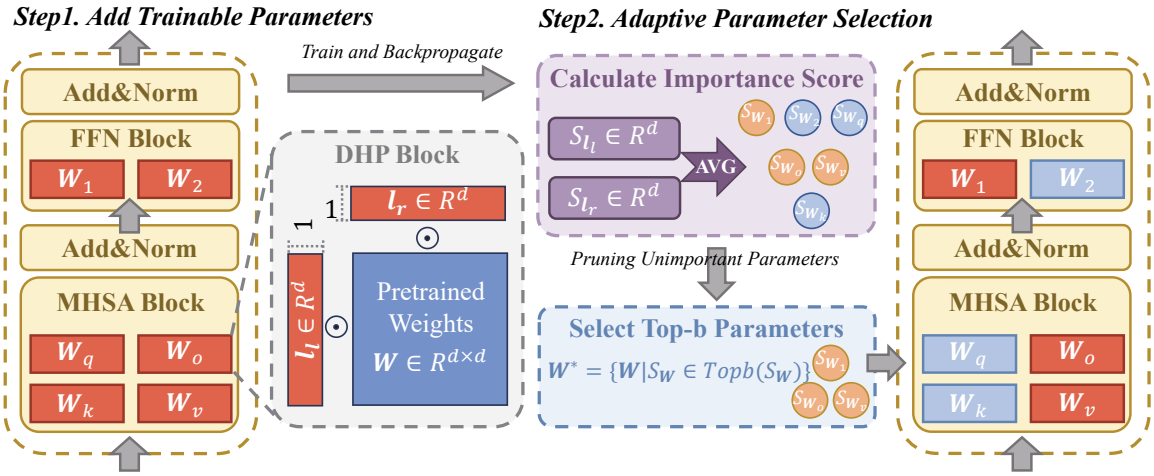


Figure 2: The framework of the AdaDHP method. MHSA indicates the mult-head self-attention block and FFN indicates the feed-forward block.

parameter allocation is not flexible becasue of the same rank of all paramters, some methods such as AdaLoRA (Zhang et al., 2023b) and IncreLoRA (Zhang et al., 2023a) are proposed. These methods not only optimize parameter efficiency but also enhance the flexibility and effectiveness. There are other additive fine-tuning methods, like ladder side-tuning (LST) (Sung et al., 2022) adds a side network to the model, which receives intermediate activations from the pretrained network via shortcut connections. And it only updates the parameters in the side network, so that backpropagation only passes through the side network without the backbone network, which greatly reduces the required GPU memory for training. And $(IA)^3$ (Liu et al., 2022) introduces three learned vectors, $\mathbf{l}_k$, $\mathbf{l}_v$, and $\mathbf{l}_{ff}$ to rescale the key, value and feedforward intermediate activations, significantly boosting parameter efficiency.

2.3 Hybrid Fine-Tuning

Hybrid fine-tuning approaches aim to combine various PEFT techniques leveraging the strengths of each to compensate for their weaknesses. These works can be classified into Manual Combination and Automatic Combination. An example of Manual Combination methods is the MAM Adapter (He et al., 2022), which combines scaled parallel adapter and prefix-tuning. Automatic combinations, such as AutoPEFT (Zhou et al., 2023), integrate various PEFT methods like sequential adapter, parallel adapter and prefix-tuning automatically through structure search. This approach usually requires more time and cost due to optimization searches in the model or structure.

3 The Proposed Method

3.1 Preliminary

Two particular PEFT strategies currently exist: one involves modifying the weights $\mathbf{W}$ mentioned earlier, while the other involves modifying intermediate activations. The most representative methods are LoRA and $(IA)^3$.

LoRA. LoRA assumes delta weight is low-rank, then decomposes it into the product of two low-rank matrices, which is expressed as:

$$\mathbf{W}' = \mathbf{W} + \Delta\mathbf{W} = \mathbf{W} + \mathbf{BA}, \tag{1}$$

where $\mathbf{W}, \mathbf{W}', \Delta\mathbf{W} \in \mathbb{R}^{d\times d}$, $\mathbf{B} \in \mathbb{R}^{d\times r}$, $\mathbf{A} \in \mathbb{R}^{r\times d}$. And the rank $r \ll d$. Thus, the quantity of parameters required for fine-tuning $\mathbf{W}$ reduces from d^2 to $2dr$, achieving efficient fine-tuning.

$(IA)^3$. To achieve mixed-task batches, $(IA)^3$ directly modifies activations. For the multi-head self-attention (MHSA) block, it performs element-wise multiplication on $\mathbf{K}$ and $\mathbf{V}$ with vectors $\mathbf{l}_k$ and $\mathbf{l}_v$, respectively, which can be represented as:

$$\text{softmax}(\frac{\mathbf{Q}(\mathbf{l}_k^T \odot \mathbf{K})^T}{\sqrt{d}})(\mathbf{l}_v^T \odot \mathbf{V}), \tag{2}$$

where $\mathbf{l}_k, \mathbf{l}_v \in \mathbb{R}^d$. For the feed-forward (FFN) block, it only performs on intermediate activations, which can be denoted as:

$$(\mathbf{l}_{ff}^T \odot \text{ReLU}(\mathbf{X}\mathbf{W}_1 + b_1))\mathbf{W}_2 + b_2, \tag{3}$$

where $\mathbf{l}_{ff} \in \mathbb{R}^{d_{in}}$. The total number of trainable parameters in each transformer layer is only $2d + d_{in}$, and $(IA)^3$ also is highly parameter-efficient.

3.2 Dual Hadamard Product

Considering that LoRA requires to manually select rank size, whereas $(IA)^3$ can only scale the weight matrices along column, these limit the flexibility of fine-tuning. Hence, we propose a vector dot production strategy to fine-tune the model parameters in a more fine-grained and flexible way. The framework of our method is illustrated in Figure 2. In particular, we fine-tune $\mathbf{W}$ by performing the element-wise dot product with both a row vector and a column vector, which can also be understood as scaling $\mathbf{W}$ along both row and column:

$$\mathbf{W}' = \mathbf{l}_l \odot \mathbf{W} \odot \mathbf{l}_r^T, \tag{4}$$

where $\mathbf{l}_l, \mathbf{l}_r \in \mathbb{R}^d$.

Compared to LoRA, our method can achieve better results even with rank of 1. This implies a more efficient approach in terms of parameter usage while still maintaining effectiveness in fine-tuning. It can be observed that $(IA)^3$ scales the weight values of each column equally, whereas our method allows for different transformations on both rows and columns of the weight matrices, thus obtaining more precise fine-tuning and achieving better outcomes.

To enhance the capacity of our approach for training complex tasks, we introduce a scaling mode of DHP during the experiments. We introduce multiple pair of $\mathbf{l}_l$ and $\mathbf{l}_r$ vectors for performing Hadamard product with $\mathbf{W}$ respectively in Eq. (4), followed by averaging or summing the output. Specifically, the trainable parameters can be formally defined as $\mathbf{l} = \{\mathbf{l}_l^1, \mathbf{l}_r^1, \mathbf{l}_l^2, \mathbf{l}_r^2, ..., \mathbf{l}_l^k, \mathbf{l}_r^k\}$, with k being adjustable based on the specific task. For each pair of parameters, we perform the same operation $\mathbf{W}^i = \mathbf{l}_l^i \odot \mathbf{W} \odot \mathbf{l}_r^{iT}$. The final $\mathbf{W}'$ is determined by the average or the sum of $\mathbf{W}^i$.

3.3 Adaptive Parameter Selection

By observing the empirical results, LoRA proposes that only fine-tuning $\mathbf{W}_q$ and $\mathbf{W}_v$ can achieve relatively satisfied results, while $(IA)^3$ only operates on $\mathbf{K}$, $\mathbf{V}$ and the intermediate activations in FFN. However, these parameters are manually selected for specific tasks and may not fit all tasks. We aim to adaptively select which parameters to be fine-tuned for downstream tasks. Currently, there are methods like AdaLoRA (Zhang et al., 2023b) that utilize importance scores to allocate different ranks to various parameters, thereby facilitating adaptive distribution. Our method, however, focuses on identifying which specific parameters should be fine-tuned. Aiming at this issue, the adaptive parameter selection strategy is divided into two main modules: **Importance Score Calculation** and **Dynamic Allocation**.

Importance Score Calculation. Since we need to remove some parameters that do not require fine-tuning, we first calculate the importance of parameters. There are many importance calculation strategies in pruning methods. Parameter magnitude (Han et al., 2015; Paganini and Forde, 2020) defines the importance of a weight based on its magnitude, but this alone cannot accurately represent the impact of a weight to the loss. Sensitivity of parameters (Ding et al., 2019; Molchanov et al., 2019; Sanh et al., 2020) is another importance metric which considers both the weight magnitude and the change rate of the loss with respect to this weight:

$$I(\theta) = |\theta \cdot \nabla_\theta \mathcal{L}|, \tag{5}$$

where $\mathcal{L}$ and θ represent the loss function and the trainable parameters respectively.

According to Zhang et al. (2022), Eq. (5) exhibits significant variations because of the use of mini-batch processing and the complex training process, such as dropout. To address this issue, Zhang et al. (2022) propose to smooth $I(\theta)$ using an exponential moving average as follows:

$$\overline{I}^{(t)} = \begin{cases} I^{(0)} & t = 0, \\ \beta_1 \overline{I}^{(t-1)} + (1-\beta_1) I^{(t)} & t \in [1, T), \end{cases} \tag{6}$$

where $\beta_1 \in (0, 1)$, t and T denote the t-th step and the total trainable steps. $\overline{I}^{(t)}$ represents the sensitivity in the t-th steps. Besides sensitivity smoothing, they also directly consider the uncertainty of importance estimation to reduce variability,

$$U^{(t)} = |I^{(t)} - \overline{I}^{(t)}|, \tag{7}$$

and they further introduce exponential weighted averaging.

$$\overline{U}^{(t)} = \beta_2 \overline{U}^{(t-1)} + (1-\beta_2) U^{(t)}, \tag{8}$$

where $\beta_2 \in (0, 1)$. $\overline{U}^{(t)}$ represents the uncertainty of sensitivity values at t step.

In summary, the importance score for a parameter is defined as follows:

$$S^{(t)} = \overline{I}^{(t)} \odot \overline{U}^{(t)}, \tag{9}$$

Algorithm 1 AdaDHP

Input: dataset $\mathcal{D}$; total steps T; start steps t_i; final steps t_f; hyperparameters β_1, β_2.
Add trainable parameters $\mathbf{l}_l$, $\mathbf{l}_r$ for each $\{\mathbf{W}\}$;
for $t = 1, \ldots, T$ **do**
  Sample a mini-batch from $\mathcal{D}$;
  Compute the gradient $\nabla\mathcal{L}(\mathbf{l}_l, \mathbf{l}_r)$;
  Compute the sensitivity $I^{(t)}, \overline{I}^{(t)}$ by Eqs. (5), (6) for every parameter $\{\mathbf{l}_l, \mathbf{l}_r\}$;
  Compute the uncertainty $U^{(t)}, \overline{U}^{(t)}$ by Eqs. (7), (8) for every parameter $\{\mathbf{l}_l, \mathbf{l}_r\}$;
  Compute the importance score $S^{(t)}$ by Eq. (9) for every parameter $\{\mathbf{l}_l, \mathbf{l}_r\}$;
  Average the importance score of $\mathbf{l}_l$ and $\mathbf{l}_r$ as final importance score of $\mathbf{W}$;
  Compute the budget $b^{(t)}$ by Eq. (10);
  Retain the top $b^{(t)}$ important parameters;
end for
Output: Fine-tuned parameters $\{\mathbf{l}_l^{(T)}, \mathbf{l}_r^{(T)}\}$.

where $\odot$ is Hadamard product. This takes into account the sensitivity and uncertainty of the parameter, resulting in a more stable importance score.

Dynamic Allocation. Using the above importance score calculation strategy, we can obtain the importance of each parameter at each step. For our approach, we only introduce two additional parameters $\mathbf{l}_l$ and $\mathbf{l}_r$ for each $\mathbf{W}$, so we treat $\mathbf{l}_l$ and $\mathbf{l}_r$ as a combined entity, using the average of all values from these two parameters to represent their importance. This measurement guides whether to remove or keep these two parameters, thus determining whether to fine-tune $\mathbf{W}$.

One intuitive observation is that during the initial training phase when stability is low, the importance of parameters is also less stable. Therefore, it is necessary to have a warm-up period at the beginning. Moreover, in the early stages, more parameters should be pruned, and as training progresses and stability increases, fewer parameters should be pruned to maintain stable training. Following Zhang et al. (2023b), the quantity of parameters required to retain in the t-th step is formulated as:

$$b^{(t)} = \begin{cases} b^{(0)} & t \in [0, t_i) \\ b^{(T)} + \alpha(\frac{t_f - t}{t_f - t_i})^3 & t \in [t_i, t_f) \\ b^{(T)} & t \in [t_f, T] \end{cases}, \quad (10)$$

where $b^{(0)}$ and $b^{(T)}$ are initial and final parameter quantity, the removed trainable parameter quantity is $\alpha = b^{(0)} - b^{(T)}$, t_i and t_f are start and end dynamic allocation steps, respectively, and T is the total number of steps to update the model parameters. Any monotonic exponential function can achieve this functionality.

3.4 The Overall Algorithm

We summarize our method AdaDHP in Algorithm 1. First, we add a pair of trainable parameters $\mathbf{l}_l$, $\mathbf{l}_r$ for each parameter. In each step, we first perform backpropagation to compute the gradient of every trainable parameter. Then, we calculate the sensitivity and uncertainty of each parameter according to Eqs. (5), (6), (7), and (8). Next, we calculate the importance of each parameter based on Eq. (9). We treat the $\mathbf{l}_l$ and $\mathbf{l}_r$ for each parameter $\mathbf{W}$ as a whole and compute the average importance score. Finally, after using Eq. (10) to calculate the budget $b^{(t)}$ in the t-th step, we retain the top $b^{(t)}$ important parameters and set the values in other parameters to 1. After T steps, we obtain the fine-tuned models.

4 Experiments

4.1 Datasets

For T5-base model, following previous work (Asai et al., 2022), we perform experiments on 17 NLP tasks: (1) GLUE benchmark (Wang et al., 2018) includes CoLA, SST-2, MRPC, QQP, STS-B, MNLI, QNLI and RTE. (2) SuperGLUE benchmark (Wang et al., 2019) includes MultiRC, BoolQ, WiC, WSC and CB. (3) Other datasets include WinoGrande (Sakaguchi et al., 2020), Yelp-2 (Zhang et al., 2015), SciTail (Khot et al., 2018) and PAWS-Wiki (Zhang et al., 2019). Detailed dataset statistics can be found in Table 6 of Appendix.

For all the datasets, we follow (Sung et al., 2022) and (Asai et al., 2022) to split them into training, validation and test sets. Specifically, for MNLI, we use mismatched validation set as validation set and matched validation set as test set. For the datasets with relatively small amounts of samples (RTE, MPRC, STS-B, CoLA, SciTail, PAWS-Wiki and all the datasets of SuperGLUE benchmark), we split the validation set into two equal-sized subsets as validation and test sets. For the remaining datasets (QQP, QNLI, SST-2, WinoGrande and Yelp-2), we split 1k samples from the training set as the new validation set and use the original validation set as test set. We limit the maximum size of training set to 100k for Yelp-2.

Method	# Params	CoLA Mc	STS-B Pc	MRPC Acc	RTE Acc	QNLI Acc	SST-2 Acc	QQP Acc	MNLI Acc	All Avg.
Full Fine-tuning	220M	61.8	89.7	90.2	71.9	93.0	94.6	91.6	86.8	84.9
Adapter	1.9M	64.0	90.7	85.3	71.9	93.2	93.8	90.2	**86.5**	84.5
AdapterDrop	1.1M	62.7	**91.4**	86.3	71.2	93.2	93.6	90.2	86.3	84.4
BitFit	280K	58.2	90.9	86.8	67.6	93.0	94.2	90.1	85.3	83.3
LoRA	3.8M	63.3	91.0	88.2	75.5	93.2	94.3	90.4	86.3	85.3
LST	3.8M	58.1	90.5	87.9	71.9	93.3	94.1	90.4	85.6	84.0
PT	76.8K	10.6	89.5	68.1	54.7	92.8	90.9	89.7	81.3	72.2
$(IA)^3$	129K	60.3$\pm$0.7	91.2$\pm$0.2	**91.0**$\pm$0.2	73.6$\pm$0.9	93.2$\pm$0.1	94.0$\pm$0.2	90.2$\pm$0.2	85.8$\pm$0.0	84.9
AdaLoRA	811K	62.9$\pm$0.5	91.3$\pm$0.0	88.6$\pm$0.2	75.3$\pm$0.9	**93.4**$\pm$0.1	94.2$\pm$0.2	90.2$\pm$0.0	86.0$\pm$0.0	85.2
AsymmetryLoRA	442K	53.4$\pm$2.5	90.2$\pm$0.3	89.5$\pm$0.6	71.2$\pm$3.1	92.7$\pm$0.1	93.5$\pm$0.2	89.9$\pm$0.0	84.4$\pm$0.1	83.1
FourierFT	72K	58.6$\pm$1.7	90.5$\pm$0.4	86.9$\pm$1.4	71.9$\pm$1.2	92.9$\pm$0.1	94.4$\pm$0.2	90.3$\pm$0.1	85.6$\pm$0.0	83.9
qGOFT	810K	56.1$\pm$2.1	91.3$\pm$0.2	88.7$\pm$1.0	71.2$\pm$1.0	93.3$\pm$0.1	**94.5**$\pm$0.2	90.5$\pm$0.1	86.1$\pm$0.0	84.0
DoRA	1.8M	61.6$\pm$2.1	89.9$\pm$0.2	87.4$\pm$1.6	74.1$\pm$1.2	92.9$\pm$0.1	94.2$\pm$0.2	**91.1**$\pm$0.1	84.5$\pm$0.0	84.5
AdaDHP	203K	**64.3**$\pm$2.6	**91.4**$\pm$0.1	90.5$\pm$0.8	**77.7**$\pm$1.0	**93.4**$\pm$0.0	**94.5**$\pm$0.2	90.6$\pm$0.0	**86.5**$\pm$0.0	**86.1**

Table 1: Test results on GLUE benchmark, with the corresponding size of trainable parameters. All the results are based on the T5-base model. We use Matthew's Correlation and Pearson Correlation as the metrics for CoLA and STS-B, respectively, and accuracy for other tasks.

To verify the scalability of our model on larger models and complex tasks, we train Llama-7B and Llama2-7B models on Math10K dataset (Hu et al., 2023) and test on six different math reasoning datasets: MultiArith (Roy and Roth, 2016), GSM8K (Cobbe et al., 2021), AddSub (Hosseini et al., 2014), AQuA (Ling et al., 2017), SingleEq (Koncel-Kedziorski et al., 2015) and SVAMP (Patel et al., 2021).

4.2 Baselines

We compare our method with the following strong baselines: (1) **Full Fine-tuning** fine-tunes all parameters of the backbone for downstream tasks. (2) **Adapter** (Houlsby et al., 2019) inserts bottleneck architecture after the attention and FFN modules. (3) **AdapterDrop** (Rücklé et al., 2021) removes adapters from lower transformer layers, enabling more efficiency during training and inference. (4) **BitFit** (Zaken et al., 2022) merely fine-tunes the biases in the pre-trained model. (5) **LoRA** (Hu et al., 2022) decomposes the parameters into two low-rank matrices and only fine-tunes them. (6) **Ladder Side-Tuning** (LST) (Sung et al., 2022) employs a side network that receives intermediate activations through shortcut connections, and only fine-tunes the side network. (7) **Prompt Tuning** (PT) (Lester et al., 2021) inserts learnable prompt tokens into the model input. (8) **$(IA)^3$** (Liu et al., 2022) fine-tunes intermediate activations, which introduces rescaling vectors for the keys and values in the attention architecture and for the intermediate activation in the FFN architecture. (9) **AdaLoRA** (Zhang et al., 2023b) dynamically assigns different ranks to each parameter based on the importance metrics. (10) **AsymmetryLoRA** (Zhu et al., 2024) explores the different role of **A** and **B** in LoRA, and proposes many initialization ways to get considerable results. (11) **FourierFT** (Gao et al., 2024) treat $\Delta\mathbf{W}$ as matrix in the spatial domain, learns only its spectral coefficients, and use IDFT to recover $\Delta\mathbf{W}$. (12) **qGOFT** (Ma et al., 2024) designs given-based orthogonal finetuning method with fewer parameters. (13) **DoRA** (Liu et al., 2024) divides the pretrained weights to magnitude and direction components, and use LoRA for directional updates.

4.3 Implementation Details

We conduct experiment on the T5-base (Raffel et al., 2020), Llama-7B (Touvron et al., 2023a) and Llama2-7B (Touvron et al., 2023b) models to evaluate the effectiveness of our method. We select hyperparameters based on the validation set for our method, $(IA)^3$ and AdaLoRA, and report averaged results over 3 different runs. All the results of other baselines are sourced from (Asai et al., 2022) and (Sung et al., 2022). For the T5-base model on the GLUE benchmark, we set batch size as 100 and maximum sequence length as 128, while setting batch size as 16 and maximum sequence length as 256 for the other datasets. The other hyperparameters are provided in Table 5 of Appendix.

Method	# Params	SuperGLUE						Other Datasets				
		BoolQ	WiC	WSC	CB	MultiRC	Avg.	WG	Yelp	SciTail	PAWS	Avg.
Full Fine-tuning	220M	81.1	70.2	59.6	85.7	72.8	73.9	61.9	96.7	95.8	94.1	87.1
Adapter	1.9M	82.5	67.1	67.3	85.7	**75.9**	75.7	**59.2**	96.9	94.5	94.3	86.2
BitFit	280K	79.6	70.0	59.6	78.6	74.5	72.5	57.2	94.7	94.7	92.0	84.7
LoRA	3.8M	81.3	68.3	67.3	89.3	72.6	75.8	58.2	97.1	94.7	94.0	86.0
PT	76.8K	61.7	48.9	51.9	67.9	58.7	57.8	49.6	95.1	87.9	55.8	72.1
$(IA)^3$	129K	82.2	69.1	66.7	90.5	73.2	76.3	57.7	97.1	95.2	93.9	86.0
AdaLoRA	811K	82.1	68.7	66.7	89.3	73.3	76.0	58.5	97.2	96.0	94.3	86.5
AsymmetryLoRA	442K	80.0	65.7	66.7	81.0	72.4	73.2	58.3	96.6	92.2	91.0	84.5
FourierFT	72K	80.1	66.2	63.5	83.3	72.2	73.1	57.1	96.8	93.3	92.4	84.9
qGOFT	810K	81.8	66.1	67.3	86.9	73.2	75.1	58.5	96.7	94.3	94.1	85.9
DoRA	1.8M	81.6	69.5	67.3	88.1	73.3	76.0	59.0	96.9	95.8	94.2	86.5
AdaDHP	203K	**82.6**	**70.2**	**67.9**	**91.7**	73.4	**77.1**	59.1	**97.3**	**96.2**	**94.6**	**86.8**

Table 2: Test results on SuperGLUE benchmark and other datasets. All the results are based on the T5-base model. We use F1 for MultiRC, and accuracy for other tasks as evaluation metrics.

Model	Method	Variable	MultiArith	GSM8K	AddSub	AQuA	SingleEq	SVAMP	Avg.
Llama-7B	**LoRA**	$r = 4$	95.2	34.7	78.5	16.1	77.8	45.4	57.9
		$r = 8$	96.2	35.6	80.5	15.7	82.3	49.6	60.0
		$r = 16$	95.5	36.2	82.8	13.8	84.4	50.9	60.6
		$r = 32$	95.0	37.5	83.3	18.9	84.4	52.1	61.9
	AdaDHP	$k = 1$	94.3	30.4	79.5	19.1	77.2	41.1	56.9
		$k = 4$	97.5	36.2	83.5	16.9	81.5	46.1	60.3
		$k = 8$	94.0	37.5	84.1	16.1	84.3	49.4	60.9
		$k = 16$	95.0	36.9	84.1	18.1	86.2	50.9	61.9
Llama2-7B	**LoRA**	$r = 4$	95.1	39.3	83.0	19.7	82.1	47.6	61.1
		$r = 8$	94.3	39.7	82.5	20.9	85.0	47.4	61.6
		$r = 16$	94.5	41.5	84.1	19.3	85.8	50.9	62.7
		$r = 32$	94.5	39.3	83.5	19.3	86.2	47.5	61.7
	AdaDHP	$k = 1$	96.2	37.1	83.8	18.1	83.5	44.0	60.5
		$k = 4$	94.7	35.6	83.5	18.9	84.3	47.1	60.7
		$k = 8$	92.0	42.0	84.6	19.7	86.0	52.5	62.8
		$k = 16$	94.2	39.7	85.1	20.1	83.7	49.7	62.1

Table 3: Test results of six math datasets on Llama-7B and Llama2-7B models. We use accuracy for all tasks as evaluation metrics.

4.4 Experimental Results

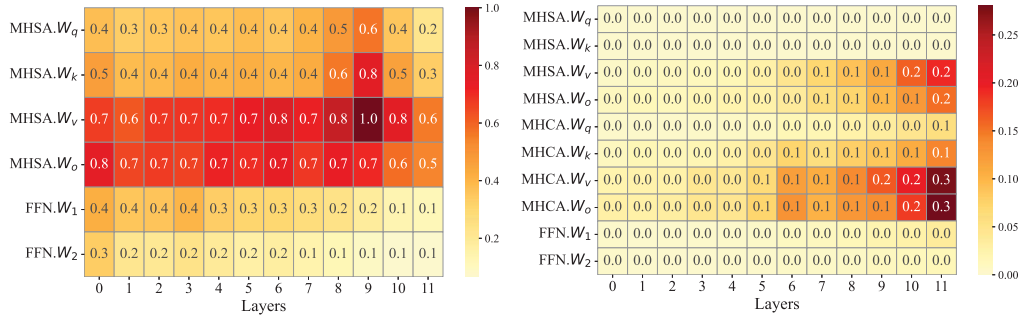
4.4.1 Results on the T5-base Model

Table 1 presents our results on the GLUE benchmark using the T5-base model. Compared with other PEFT methods, our method AdaDHP outperforms other PEFT methods on 6 out of 8 datasets and surpasses Full Fine-tuning on 5 datasets. Additionally, the average performance of AdaDHP exceeds that of both Full Fine-tuning and other baselines. Despite fine-tuning only 0.05% of the parameters compared to our strong baseline LoRA, our method delivers superior results across all datasets. These findings demonstrate that AdaDHP achieves superior performance while maintaining higher parameter efficiency compared to other methods.

Table 2 shows our results on the SuperGLUE benchmark and other datasets based on the T5-base model. On the SuperGLUE benchmark, our method yields the best performance across 4 out of 5 datasets compared to all other baselines. Additionally, our method outperforms Full Fine-tuning across all the datasets. Regarding the other four datasets, our method outperforms Full Fine-tuning and the other baselines on three datasets. These

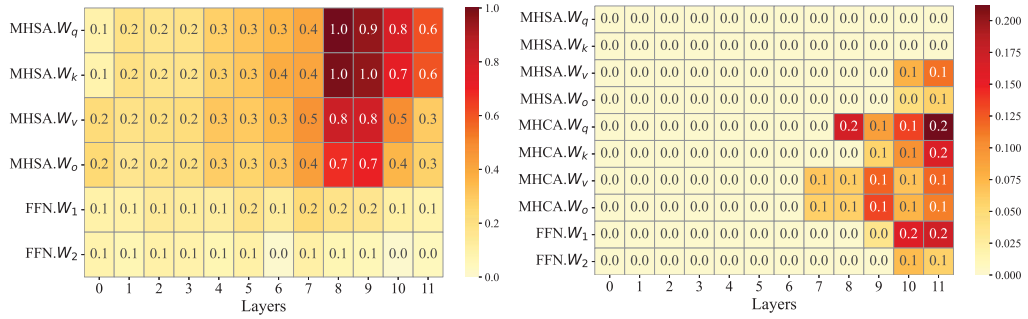
Method	CoLA	STS-B	MRPC	RTE	QNLI	SST-2	QQP	MNLI
AdaDHP	64.3±2.6	91.4±0.1	90.5±0.8	77.7±1.0	93.4±0.0	94.5±0.2	90.6±0.0	86.5±0.0
-Adaptive	62.3±0.4	91.4±0.2	88.2±1.8	73.6±0.3	93.3±0.1	94.2±0.3	90.4±0.2	86.2±0.1
-Adaptive$_{column}$	61.9±1.3	90.9±0.1	87.9±1.0	72.2±0.3	93.3±0.0	94.0±0.1	90.3±0.0	85.7±0.1
-Adaptive$_{row}$	60.6±1.9	91.1±0.0	88.7±1.1	75.3±0.9	93.3±0.1	92.4±0.5	90.4±0.2	86.2±0.2
AdaDHP$_{magnitude}$	61.6±0.5	91.0±0.1	89.4±1.0	73.9±1.8	93.2±0.0	91.2±0.2	90.3±0.0	86.0±0.1
AdaDHP$_{sensitivity}$	61.6±2.1	91.3±0.1	89.5±2.0	74.6±0.9	93.2±0.1	90.5±0.5	90.5±0.1	86.5±0.1

Table 4: Ablation study on the GLUE benchmark using the T5-base model. -Adaptive indicates the removal of the adaptive parameter selection module. -Adaptive$_{column}$ and -Adaptive$_{row}$ refer to removing the adaptive module while scaling only the column parameters or the row parameters. AdaDHP$_{magnitude}$ and AdaDHP$_{sensitivity}$ replace the importance selection strategy with magnitude and sensitivity methods, respectively.



(a) Encoder Layers of Full Fine-tuning

(b) Decoder Layers of Full Fine-tuning



(c) Encoder Layers of AdaDHP

(d) Decoder Layers of AdaDHP

Figure 3: The importance scores of T5-base model parameters. MHSA, MHCA, and FFN represent multi-head self-attention, multi-head cross-attention, and the feed-forward module.

observations once again demonstrate the high parameter efficiency of our method.

4.4.2 Results on Llama Series Models

Table 3 presents our results using the Llama-7B and Llama2-7B models. Among them, $k = 1$ is the basic mode of our method AdaDHP, and we also explore the impact of scaling with larger values of k. From the results, we can observe that our method at $k = 1$ shows comparable performance to LoRA at $r = 4$ on both two models. And when k increases, our results consistently improve and are also superior to the results of all baselines under the same parameter settings ($r = k$). This trend highlights the scalability and adaptability of our method, and demonstrates the effectiveness of our method on complex tasks and large models.

4.4.3 Ablation Study

Table 4 shows the results of our method when the adaptive parameter selection module is abandoned, denoted as "-Adaptive". It can be observed that the results of "-Adaptive" are much worse than AdaDHP, especially there is a significant decrease in performance for CoLA, MRPC, RTE, WiC, WSC, and CB. The reason is that these datasets are relatively small, and using a larger number of parameters can lead to overfitting. Therefore,

the performance is improved when removing the unimportant or redundant parameters by this module. Compared with -Adaptive, -Adaptive$_{column}$ and Adaptive$_{row}$ which scale the parameters along either rows or columns generally performs much worse. It can be seen that scaling only the rows or only the columns leads to a significant performance decrease in both cases, demonstrating the importance of fine-grained fine-tuning via dual hadamard product.

Additionally, we conduct experiments using commonly applied importance scoring methods, such as the magnitude and the sensitivity. The results are shown in Table 4. The results indicate that our method outperforms these two variants. Our importance calculation method employs an exponential moving average, which provides a more stable and reliable importance score. Therefore, the retained parameters are more impactful, leading to better overall performance.

4.4.4 Visualization

To validate that the trainable parameters selected by LoRA and (IA)^3 are not flexible due to manual selection, we take the RTE dataset as an example, and use the importance score calculation module to compute the importance for each parameter of the T5-base model. Figure 3 shows the normalized parameter importance scores of Full Fine-tuning and AdaDHP, the parameters with the scores larger than 0 are the important parameters required to be fine-tuned. It can be seen that the selected parameters of AdaDHP are generally consistent with Full Fine-tuning, i.e., all the parameters in the encoder and the cross-attention parameters of the last few layers in the decoder are important. Whereas LoRA proposes to only fine-tune $\mathbf{W}_q$ and $\mathbf{W}_v$, while (IA)^3 is equivalent to fine-tuning $\mathbf{W}_k$, $\mathbf{W}_v$ and $\mathbf{W}_2$, leaving many important parameters unfine-tuned. This observation demonstrates that AdaDHP can precisely select and fine-tune important parameters.

5 Conclusion

In this paper, we propose an innovative parameter-efficient fine-tuning method called AdaDHP for fine-tuning large language models. To increase the granularity of fine-tuning parameters and significantly reduce the number of training parameters, we perform Hadamard Product on each parameter along both rows and columns. To adaptively select parameters for downstream tasks, we use an importance score calculation strategy and an dynamic allocation mechanism to select important parameters. We conduct extensive experiments on 17 NLP datasets with T5-base model and complex mathematical tasks with Llama series models, validating the superiority of our method.

Limitations

Our approach currently prioritizes achieving a good balance between parameter count and performance, but it does not yet address excessive memory usage. Additionally, some hyperparameters still require fine-tuning. The limitations of this study are as follows: (1) In future work, we plan to explore quantization techniques for the base model, similar to the approach in QLoRA (Dettmers et al., 2023), to reduce memory usage and enable more parameter-efficient fine-tuning. (2) We will also focus on developing additional strategies to optimize budget allocation in the Adaptive Parameter Selection module, while reducing the number of hyperparameters in this module to simplify the tuning process and improve efficiency.

Acknowledgments

This work was supported by National Natural Science Foundation of China (No. 62206038, 62106035), Liaoning Binhai Laboratory Project (No. LBLF-2023-01), Chunhui Project Foundation of the Education Department of China (No. HZKY20220419), and Xiaomi Young Talents Program.

References

Akari Asai, Mohammadreza Salehi, Matthew E. Peters, and Hannaneh Hajishirzi. 2022. ATTEMPT: parameter-efficient multi-task tuning via attentional mixtures of soft prompts. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6655–6672.

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 *Conference on Neural Information Processing Systems (NeurIPS)*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv*, abs/2110.14168.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Xiaohan Ding, Guiguang Ding, Xiangxin Zhou, Yuchen Guo, Jungong Han, and Ji Liu. 2019. Global sparse momentum SGD for pruning very deep neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, pages 6379–6391.

Ziqi Gao, Qichao Wang, Aochuan Chen, Zijing Liu, Bingzhe Wu, Liang Chen, and Jia Li. 2024. Parameter-efficient fine-tuning with discrete fourier transform. In *International Conference on Machine Learning (ICML)*.

Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both weights and connections for efficient neural network. In *Conference on Neural Information Processing Systems (NeurIPS)*, pages 1135–1143.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations (ICLR)*.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–533.

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

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. 2023. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5254–5276.

Tushar Khot, Ashish Sabharwal, and Peter Clark. 2018. Scitail: A textual entailment dataset from science question answering. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 5189–5197.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3045–3059.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4582–4597.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 158–167.

Haokun Liu, Derek Tam, Mohammed Muqeeth, 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 *Conference on Neural Information Processing Systems (NeurIPS)*.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: Weight-decomposed low-rank adaptation. In *International Conference on Machine Learning (ICML)*.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021. GPT understands, too. *arXiv*, abs/2103.10385.

Xinyu Ma, Xu Chu, Zhibang Yang, Yang Lin, Xin Gao, and Junfeng Zhao. 2024. Parameter efficient quasi-orthogonal fine-tuning via givens rotation. In *International Conference on Machine Learning (ICML)*.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. Importance estimation for neural network pruning. In *Computer Vision and Pattern Recognition (CVPR)*, pages 11264–11272.

Michela Paganini and Jessica Zosa Forde. 2020. On iterative neural network pruning, reinitialization, and the similarity of masks. *arXiv*, abs/2001.05050.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 2080–2094.

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.

Subhro Roy and Dan Roth. 2016. Solving general arithmetic word problems. *arXiv*, abs/1608.01413.

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 *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7930–7946.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Winogrande: An adversarial winograd schema challenge at scale. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 8732–8740.

Victor Sanh, Thomas Wolf, and Alexander M. Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. 2022. LST: ladder side-tuning for parameter and memory efficient transfer learning. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Yi-Lin Sung, Varun Nair, and Colin Raffel. 2021. Training neural networks with fixed sparse masks. In *Conference on Neural Information Processing Systems (NeurIPS)*, pages 24193–24205.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *arXiv*, abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv*, abs/2307.09288.

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 *Conference on Neural Information Processing Systems (NeurIPS)*, pages 3261–3275.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 353–355.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1–9.

Feiyu Zhang, Liangzhi Li, Junhao Chen, Zhouqiang Jiang, Bowen Wang, and Yiming Qian. 2023a. Increlora: Incremental parameter allocation method for parameter-efficient fine-tuning. *arXiv*, abs/2308.12043.

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

Qingru Zhang, Simiao Zuo, Chen Liang, Alexander Bukharin, Pengcheng He, Weizhu Chen, and Tuo Zhao. 2022. PLATON: pruning large transformer models with upper confidence bound of weight importance. In *International Conference on Machine Learning (ICML)*, pages 26809–26823.

Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Conference on Neural Information Processing Systems (NeurIPS)*, pages 649–657.

Yuan Zhang, Jason Baldridge, and Luheng He. 2019. PAWS: paraphrase adversaries from word scrambling. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 1298–1308. Association for Computational Linguistics.

Mengjie Zhao, Tao Lin, Fei Mi, Martin Jaggi, and Hinrich Schütze. 2020. Masking as an efficient alternative to finetuning for pretrained language models. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2226–2241.

Han Zhou, Xingchen Wan, Ivan Vulic, and Anna Korhonen. 2023. Autopeft: Automatic configuration search for parameter-efficient fine-tuning. *arXiv*, abs/2301.12132.

Jiacheng Zhu, Kristjan H. Greenewald, Kimia Nadjahi, Haitz Sáez de Ocáriz Borde, Rickard Brüel Gabrielsson, Leshem Choshen, Marzyeh Ghassemi, Mikhail Yurochkin, and Justin Solomon. 2024. Asymmetry in low-rank adapters of foundation models. In *International Conference on Machine Learning (ICML)*.

A Datasets

Table 6 summarizes the relevant information of the training, validation and test sets for the 17 NLP tasks we used. As mentioned in the main paper, for datasets with relatively small samples, such as RTE, MPRC, STS-B, CoLA, SciTail, PAWS-Wiki, and all the datasets from the SuperGLUE benchmark, we divide the validation set into two equal-sized subsets, one serving as the new validation set and the other as the test set. Conversely, for the remaining datasets including QQP, QNLI, SST-2, WinoGrande, and Yelp-2, we extract 1,000 samples from the original training set to constitute a new validation set, while retaining the original validation set as the test set. Additionally, for Yelp-2, we cap the training set at 100,000 samples.

B Hyperparameter Investigation

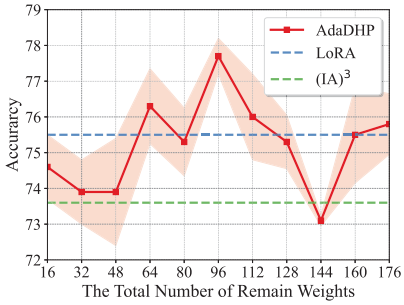


Figure 4: Results on RTE while retaining different parameter quantities.

Besides the number of epochs and learning rate selected based on the validation set, the hyperparameters of our method include β_1, β_2, $b^{(0)}$, $b^{(T)}$, t_i and t_f. We follow (Zhang et al., 2022) to set β_1 and β_2. $b^{(0)}$ is the number of all the trainable parameters. Hence, we only need to investigate the remaining three hyperparameters. For $b^{(T)}$, the accuracy of validation set for our method on the RTE dataset under different budget scenarios is shown in Figure 4. It can be seen that our method outperforms the strong baselines LoRA and $(IA)^3$ when $b^{(T)}$ reaches 64, and achieves the best performance when $b^{(T)} = 96$. And the number of all the trainable parameters is only 135K when $b^{(T)} = 64$, which is equivalent to $(IA)^3$, indicating that our method can select the most important parameters and achieve better results. According to the results in Figure 4, we set $b^{(T)} = 96$ when fine-tuning the T5-base model. After manual tuning on the CoLA dataset, we obtain the values for t_i and t_f. For other datasets, we scale these values based on the ratio with the total number of updates T, selecting corresponding values for t_i and t_f accordingly. Detailed statistics of the hyperparameters are presented in Table 5. Depending on the size of dataset, the replication time ranges from 8 minutes (for RTE) to 5 hours (for MNLI).

Dataset	Epochs	T	t_i	t_f	LR
GLUE Benchmark					
CoLA	20	1,720	100	800	$7e-3$
RTE	20	500	50	250	$2e-2$
MRPC	20	740	80	400	$2e-3$
STS-B	20	1,160	150	600	$9e-4$
SST-2	10	6,640	400	3,600	$2e-2$
QNLI	10	10,380	600	5,500	$1e-2$
QQP	10	36,290	2,000	19,000	$9e-4$
MNLI	10	39,280	2,000	21,000	$4e-2$
SuperGLUE Benchmark					
CB	30	480	30	280	$8e-4$
WSC	30	1,050	50	550	$4e-3$
WiC	20	6,800	400	3,600	$1e-3$
BoolQ	20	11,800	800	6,300	$3e-3$
MultiRC	10	17,030	1,000	9,000	$1e-3$
Other Datasets					
SciTail	10	14,750	800	8,000	$3e-3$
WinoGrande (WG)	10	24,630	1,400	13,000	$1e-3$
PAWS	10	30,880	1,800	16,500	$3e-3$
YelpPolarity (Yelp)	10	62,500	3,600	33,500	$9e-4$

Table 5: The hyperparameters of all datasets.

C Additional Experimental Results

C.1 Results on the Llama2 Model

We evaluate our method on Llama2-7B, and due to resource constraints, we just perform it on the RTE and SST-2 dataset. We load the original model parameters with 16 bit, set the batch size to 16, the maximum sequence length to 128, and the total training steps to 4000. The comparison of our method with $(IA)^3$ and AdaLoRA is shown in Table 7. It can be observed that our method can also achieve good results with larger models.

C.2 The Runtime of Different Methods

We conduct experiments on T5-base model, and the table below displays the number of training steps per second and the number of samples per second for several methods. LoRA, AdaLoRA and AdaDHP train all the weights, while $(IA)^3$ only trains the key, value, and the intermediate activations of FFN, as stated in their original papers. As shown in Table 8, the results show that

Dataset	Train	Valid	Test	Task	Metric
GLUE Benchmark					
CoLA	8,551	521	522	Acceptability	Matthew's Correlation
SST-2	66,349	1,000	872	Sentiment	Accuracy
MRPC	3,668	204	204	Paraphrase	Accuracy
QQP	362,846	1,000	40,431	Paraphrase	Accuracy
STS-B	5,749	750	750	Sentence Similarity	Pearson Correlation
RTE	2,490	138	139	Natural Language Inference	Accuracy
MNLI	392,702	9,832	9,815	Natural Language Inference	Accuracy
QNLI	362,846	1,000	40,431	Natural Language Inference	Accuracy
SuperGLUE Benchmark					
MultiRC	27,243	2,424	2,424	Question Answering	F1
BoolQ	9,427	1,635	1,635	Question Answering	Accuracy
WSC	554	52	52	Common Sense Reasoning	Accuracy
WiC	5,428	319	319	Word Sense Disambiguation	Accuracy
CB	250	28	28	Natural Language Inference	Accuracy
Other Datasets					
WinoGrande (WG)	39,398	1,000	1,267	Common Sense Reasoning	Accuracy
YelpPolarity (Yelp)	100,000	1,000	38,000	Sentiment	Accuracy
SciTail	23,596	652	652	Natural Language Inference	Accuracy
PAWS	4,9401	8,000	8,000	Sentence Similarity	Accuracy

Table 6: Stastics of all datasets.

Model	Params	Train Memory	RTE	SST-2
$(IA)^3$	0.009%	30G	64.3 ± 1.2	95.2 ± 0.2
AdaLoRA	0.074%	33G	68.9 ± 7.1	96.7 ± 0.2
AdaDHP	0.018%	32G	**70.0** ± 1.2	**97.0** ± 0.4

Table 7: Test results of RTE and SST-2 on Llama2-7B model.

Method	LoRA ($r = 1$)	LoRA ($r = 32$)	$(IA)^3$	AdaLoRA	AdaDHP
Train steps per second	1.562	1.446	**1.775**	1.225	1.471
Train samples per second	155.544	144.07	**176.766**	122.026	146.488

Table 8: Runtime comparison among LoRA, AdaLoRA, $(IA)^3$ and AdaDHP.

Method	CoLA	STS-B	MRPC	RTE	QNLI	SST-2	QQP	MNLI
LoRA ($r = 1$)	55.6± 1.1	88.6± 0.4	86.9± 0.2	67.4± 3.4	93.1± 0.1	93.6± 0.2	90.0± 0.2	85.9± 0.1
DHP	62.3± 0.4	91.4± 0.2	88.2± 1.8	73.6± 0.3	93.3± 0.1	94.2± 0.3	90.4± 0.2	86.2± 0.1

Table 9: Comparison of LoRA with $rank = 1$ and our method DHP.

although AdaDHP includes adaptive parameter selection module. it trians faster than both LoRA ($r = 32$) and AdaLoRA.

D The Difference of DHP with LoRA

Our method can be formulted as $\mathbf{W} \odot (\mathbf{l}_l \mathbf{l}_r^T)$, but it is not identical to LoRA (i.e. $\mathbf{W} + \mathbf{BA}$) with $rank = 1$. Mathematically, Hadamard product and addition are distinct operations, and their computations during backpropagation are different, so it is hard to make $\mathbf{W} \odot (\mathbf{l}_l \mathbf{l}_r^T)$ equal to $\mathbf{W} + \mathbf{BA}$ during optimization, especially in the case where the global optimal solution is difficult to be obtained. Empirically, we conduct experiments fot LoRA with $rank = 1$ and our method that only includes Dual Hadamard Product (DHP) by removing Adaptive Parameter Selection module. As shown in Table 9, our methods shows a clear advantage, which confirms the differences between the two methods.