Parameter-Efficient Fine-Tuning Methods for Pretrained Language Models: A Critical Review and Assessment

Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, Fu Lee Wang

***Abstract*—With the continuous growth in the number of parameters of transformer-based pretrained language models (PLMs), particularly the emergence of large language models (LLMs) with billions of parameters, many natural language processing (NLP) tasks have demonstrated remarkable success. However, the enormous size and computational demands of these models pose significant challenges for adapting them to specific downstream tasks, especially in environments with limited computational resources. Parameter Efficient Fine-Tuning (PEFT) offers an effective solution by reducing the number of fine-tuning parameters and memory usage while achieving comparable performance to full fine-tuning. The demands for fine-tuning PLMs, especially LLMs, have led to a surge in the development of PEFT methods, as depicted in Fig. 1. In this paper, we present a comprehensive and systematic review of PEFT methods for PLMs. We summarize these PEFT methods, discuss their applications, and outline future directions. Furthermore, we conduct experiments using several representative PEFT methods to better understand their effectiveness in parameter efficiency and memory efficiency. By offering insights into the latest advancements and practical applications, this survey serves as an invaluable resource for researchers and practitioners seeking to navigate the challenges and opportunities presented by PEFT in the context of PLMs.**

***Index Terms*—Parameter-efficient, fine-tuning, pretrained language model, large language model, memory usage.**

I. Introduction

TRANSFORMER-BASED PLMs [1], [2], [3], [4] have demonstrated remarkable performance across a wide range of NLP tasks. To fully harness the potential of PLMs, fine-tuning is employed to adapt the PLMs to task-specific data to enhance performance on downstream tasks. However, traditional fine-tuning involves updating all the pretrained parameters of PLMs, which is time-consuming and computationally expensive. As the size of PLMs continues to increase, from models like BERT [1] with 110 million parameters to T5 [4] with 770 million parameters, computational resource requirements become a significant challenge. The advent of LLMs [5], [6], [7], exemplified by Falcon [8] with a staggering 180 billion parameters, further exacerbates the computational demands. To perform task-specific full fine-tuning with Falcon-180B, a minimum of 5120GB of computational resources may be required[1]. The enormous computational resource requirements are prohibitive for anyone but the superpower players to utilize LLMs for task-specific fine-tuning.

To address this challenge, a prominent method known as PEFT [9] has emerged as a viable solution to compensate for the tremendous computational cost of full parameter fine-tuning. PEFT involves employing various deep learning techniques [9], [10], [11] to reduce the number of trainable parameters while still maintaining comparable performance to the full fine-tuning. In addition, PEFT updates only a small number of additional parameters or updates a subset of the pretrained parameters, preserving the knowledge captured by the PLM while adapting it to the target task and reducing the risk of catastrophic forgetting. Furthermore, since the size of the fine-tuned dataset is typically much smaller than the pretrained dataset, performing full fine-tuning to update all the pretrained parameters may lead to overfitting, which is circumvented by the PEFT through selectively or not updating pretrained parameters.

Recently, there has been a significant surge in interest regarding PEFT methods, as demonstrated by the growing number of studies depicted in Fig. 1. This also leads to a few surveys on PEFT approaches for the PLMs. However, the existing surveys have certain limitations. Ding et al. [12] conducted a comprehensive study on PEFT methods, but this survey did not cover much of the latest work in the field and only four PEFT methods were quantitatively experimented with. Lialin et al. [13] delved into the ideas and operational implementations of PEFT methods in detail but do not perform relevant experiments. In this work, we address these gaps comprehensively. We meticulously categorize the PEFT methods, providing detailed explanations of the ideas and specific implementations of each method. We compare the similarities and differences among various types of PEFT methods, facilitating a better understanding of the evolving landscape of PEFT. Moreover, we conduct extensive fine-tuning experiments with 11 representative PEFT methods.

In this paper, we aim to provide a comprehensive and systematic study of PEFT methods for PLMs in NLP. We undertake an in-depth exploration of these PEFT methods and

This work was supported by a research grant entitled "Medical Text Feature Representations based on Pre-trained Language Models" (871238) and Faculty Research Grant (DB24A4) at Lingnan University, Hong Kong. *(Corresponding author: Haoran Xie.)*

Lingling Xu and Fu Lee Wang are with the Hong Kong Metropolitan University, Hong Kong (email: xxiao199409@gmail.com; pwang@hkmu.edu.hk).

Haoran Xie and Si-Zhao Joe Qin are with Lingnan University, Hong Kong (email: hrxie@ln.edu.hk; joeqin@ln.edu.hk).

Xiaohui Tao is with University of Southern Queensland, Queensland, Australia (email: xtao@usq.edu.au).

[1]https://huggingface.co/blog/falcon-180b#hardware-requirements

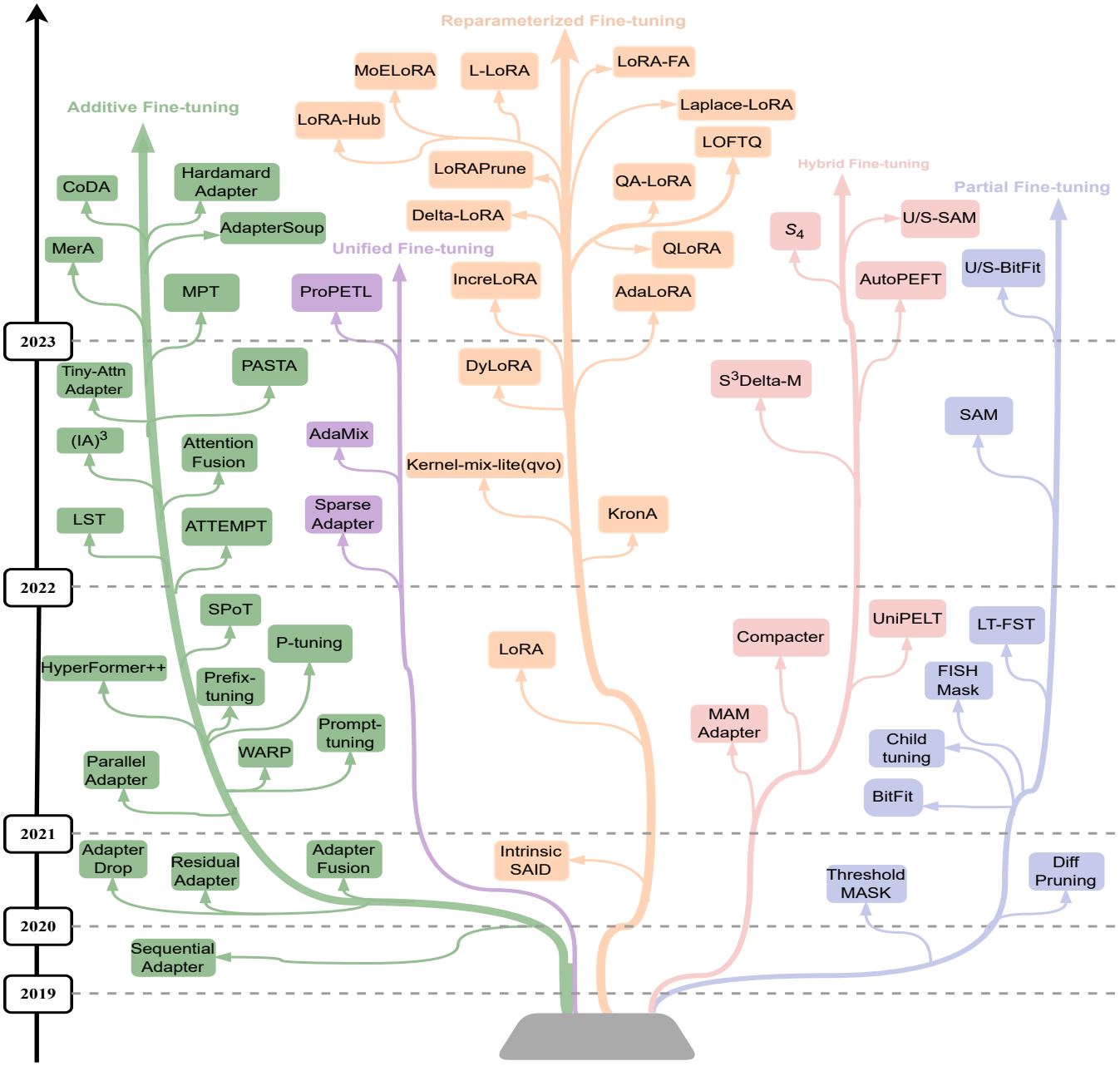


Fig. 1: The evolutionary development of PEFT methods in recent years. Models on the same branch have some common features. The vertical position of the models shows the timeline of their release dates. Notably, the year of the paper's initial publication is shown as the reference. For instance, if a paper is published in ACL 2022 but listed on arXiv in 2021, the year 2021 will be considered as the reference date.

present a comprehensive taxonomy scheme in Section III. By categorizing PEFT methods into additive fine-tuning, partial fine-tuning, reparameterized fine-tuning, hybrid fine-tuning, and unified fine-tuning, we establish a structured framework for understanding these PEFT approaches, as depicted in Fig. 2. In Section IV, we conduct quantitative investigations and analyses to assess the performance, parameters efficiency, and memory usage of these PEFT approaches. Our quantitative studies primarily focus on natural language understanding (NLU), machine translation (MT), and natural language generation (NLG) tasks. Additionally, we extensively explore the applications of PEFT in multi-task learning, cross-lingual transfer, and backdoor attack and defense, underscoring its effectiveness. Furthermore, our research also unveils potential directions for future investigations in this rapidly evolving field. To summarize, the main contributions of this survey can be outlined as follows:

- We present a comprehensive analysis and review of PEFT methods for transformer-based PLMs.
- We identify the key techniques and approaches employed in PEFT methods, and classify them into additive, partial, reparameterized, hybrid, and unified fine-tuning methods.

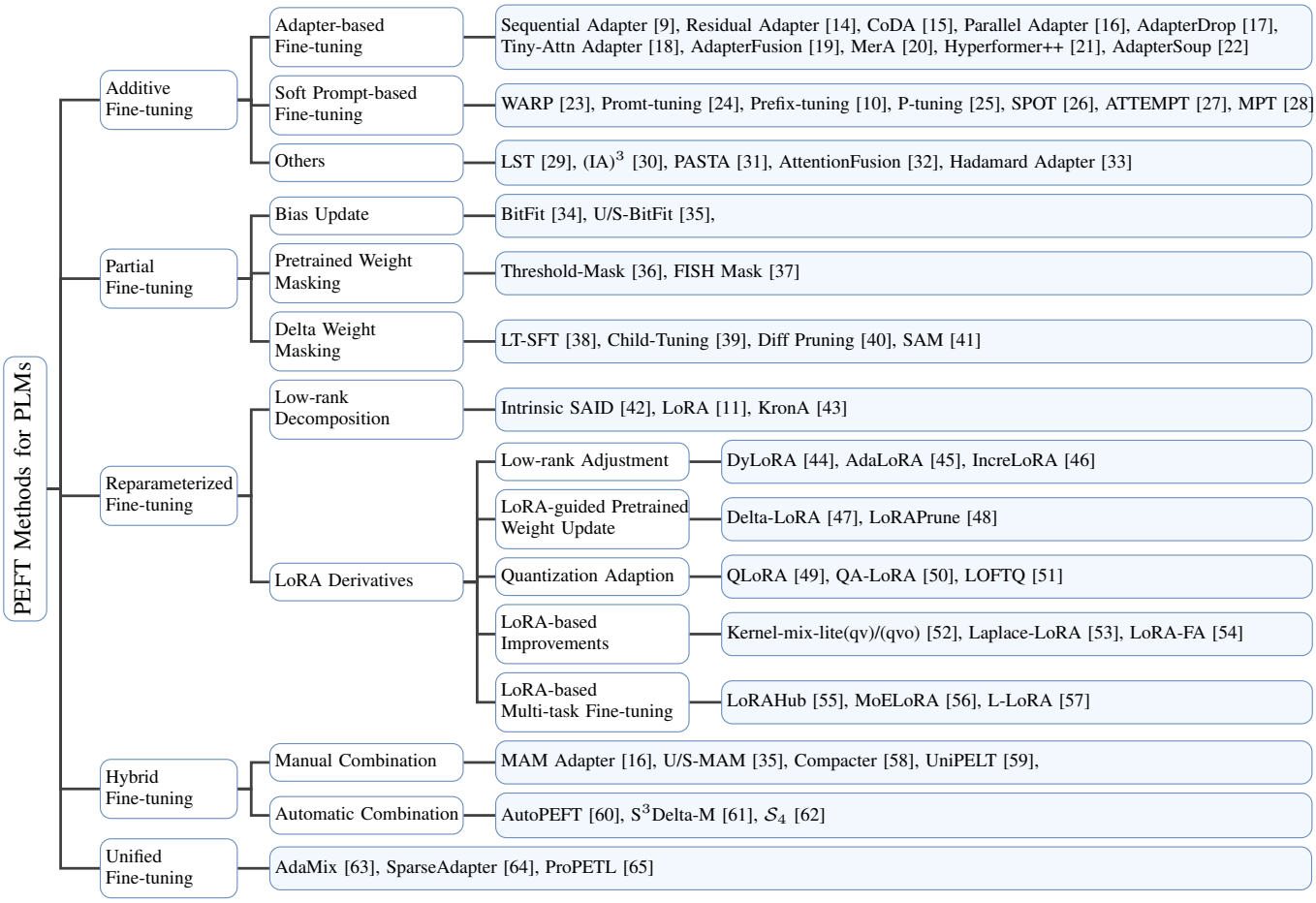


Fig. 2: Taxonomy of Parameter-Efficient Fine-Tuning Methods for Pretrained Language Models.

- We conduct extensive experiments to evaluate the effectiveness of several representative PEFT methods, specifically examining their impact on parameter efficiency and memory usage.

II. Preliminaries

A. Transformer

Transformer [66] has emerged as a foundational architecture for numerous PLMs, it adopts an encoder-decoder architecture, comprised of a stack of encoder and decoder layers, each equipped with the self-attention mechanism. Both the encoder and decoder in the Transformer architecture consist of a multi-head self-attention layer and a feed-forward network (FFN) layer, interconnected by a residual connection [67] followed by layer normalization [68]. Residual connection allows the model to effectively propagate information from one layer to the subsequent layer without losing valuable information. Layer normalization further stabilizes the training process by normalizing the inputs of each layer.

Multi-head self-attention layer employs the self-attention function with h heads in parallel. For an input sequence $X \in \mathbb{R}^{n\times d}$ with the sentence length n and hidden dimension size of d. The query ($\mathbf{Q}$), key ($\mathbf{K}$), and value ($\mathbf{V}$) vectors are the transformation of input sequence X,

$$\mathbf{K} = XW_k + b_k,\ \mathbf{Q} = XW_q + b_q,\ \mathbf{V} = XW_v + b_v, \quad (1)$$

where $Q, K, V \in \mathbb{R}^{n\times d}$, b_k, b_q and b_v are typically learnable parameter vectors that help model to better capture specific information in the input vector X and adjust the value of the query vector Q to better match the key vector K, thereby improving performance of the self-attention mechanism. The self-attention output of input X is computed as:

$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}})\mathbf{V}, \quad (2)$$

then multi-head self-attention can be described as follows:

$$\text{MHA}(Q, K, V) = \text{Concat}(\text{head}_1, \cdots, \text{head}_h)W^O, \quad (3)$$

$$\text{head}_i = \text{Attn}(QW_Q^i, KW_K^i, VW_V^i). \quad (4)$$

While the FFN consists of two linear transformations with a non-linear ReLU activation function in between:

$$\text{FFN}(X) = \text{ReLU}(XW_1 + b_1)W_2 + b_2, \quad (5)$$

where W_1, b_1, W_2 and b_2 are the weight matrices of two linear transformations. Most PEFT methods primarily focus on the self-attention layer and FFN layer, allowing models like encoder-based RoBERTa [2], encoder-decoder-based T5 [4], and decoder-based LLaMA [7] to leverage relevant techniques for parameters reduction.

B. Full Fine-tuning of PLMs

Full fine-tuning of transformer-based PLMs involves training the entire model, including all layers and parameters, on a specific downstream task using task-specific data. Initially, PLMs are trained on large-scale datasets with unsupervised learning objectives like language modeling or masked language modeling, to learn general language representations [1], [2], [4], [7]. However, these PLMs may not perform optimally when applied to specific tasks like sentiment analysis, question answering, or translation due to a lack of appropriate domain knowledge [69], [70], [71]. Full fine-tuning provides an effective solution to address this limitation.

During full fine-tuning, the PLM is initialized with pretrained weights and subsequently trained on task-specific data using techniques like backpropagation and gradient descent [72], [73]. All model parameters, including pretrained weights, are updated to minimize a task-specific loss that quantifies the disparity between predicted outputs and ground truth. In this way, full fine-tuning enables the model to learn task-specific patterns and nuances from the labeled data, facilitating predictions or outputs tailored to the target tasks [74]. Notably, full fine-tuning necessitates substantial computational resources and labeled data, as the model is trained from scratch for the specific target task. Moreover, as PLMs grow in size and with the advent of LLMs containing billions of parameters, full fine-tuning places even greater demands on computational resources. In contrast, PEFT methods aim to alleviate these requirements by selectively updating or modifying specific parts of the PLMs while still achieving performance comparable to full fine-tuning [34], [39]. Furthermore, full fine-tuning may give rise to overfitting when the task-specific dataset is small or when the PLMs are already well-suited to the target task [19], [75].

III. Parameter-Efficient Fine-Tuning Methods

A. Additive Fine-tuning

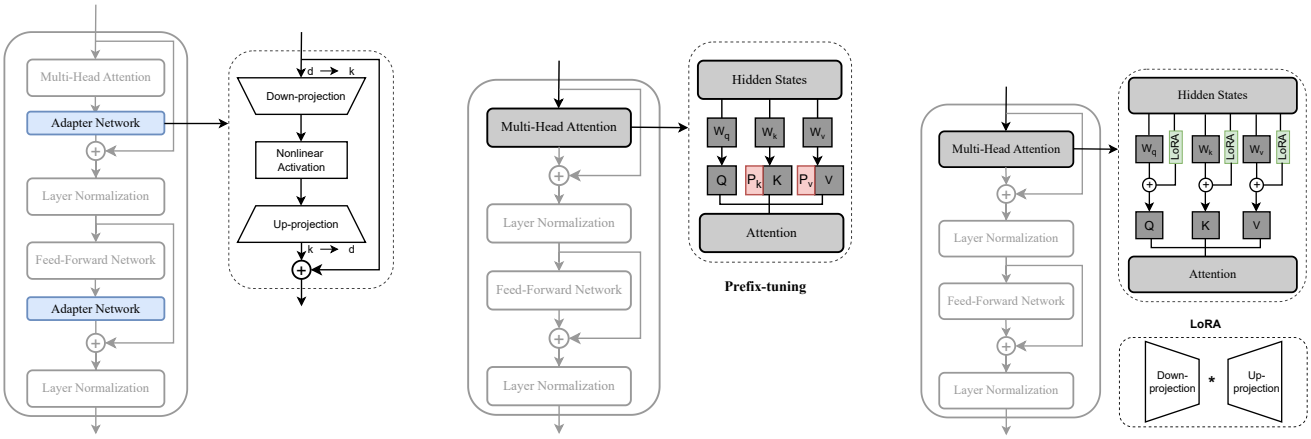
Additive fine-tuning approaches involve introducing new extra trainable parameters for task-specific fine-tuning. We classify additive fine-tuning into three groups: **Adapter-based Fine-tuning** [9], [14], [15], [16], [17], [18], [19], [20], [21], [22], [76], in which the adapter module is incorporated into the transformer, allowing for fine-tuning without modifying the pretrained parameters, **Soft Prompt-based Fine-tuning** [10], [23], [24], [25], [26], [27], [28], where soft prompts or prefix vectors are appended to the input embeddings or hidden states during fine-tuning, and **Others** [29], [30], [31], [32], [33], in which various methods that introduce supplementary parameters for model fine-tuning are fall into this category.

1) Adapters-based Fine-tuning: The idea of Adapter is first introduced in multi-domain image classification [77], allowing for the efficient transfer of knowledge across multiple visual domains. **Sequential Adapter**[9] extends and applies it to NLP tasks by inserting the adapter (trainable modules) into the transformer block and fine-tuning the parameters of adapters to make the PLMs adapt to the downstream tasks. Specifically, adapter networks are inserted after the self-attention layer and feed-forward layer of the Transformer sequentially. Each adapter are low-rank module that consists of a down-projection, a non-linear activation function, and an up-projection as well as a residual connection. For the input X, the output of a sequential adapter with the ReLU non-linear activation function can be defined with Equation 6. During fine-tuning, only the parameters of adapter network W_{up} and W_{down} need to be updated to make the PLMs adapt to the specific downstream tasks. The specific architecture of the sequential adapter is presented in Fig. 3.

$$X = (\text{ReLU}(XW_{down}))W_{up} + X, \tag{6}$$
$$W_{down} \in \mathbb{R}^{d\times k}, W_{up} \in \mathbb{R}^{k\times d}.$$

Inspired by sequential adapter, many adapter-based PEFT methods have been proposed. **Residual Adapter** [14] further improves efficiency by inserting the adapter module only after the feed-forward and layer normalization. **Parallel Adapter** [16], [76] inserts the adapter network in parallel with both the attention layer and the feed-forward layer, allowing for more efficient integration of the adapter module into the transformer. **AdapterDrop** [17] removes adapters in each layer of the transformer that are not important to the given task to improve inference efficiency. While **CoDA** (Condition Adapter) [15] employs parallel adapter for task-specific parameter fine-tuning and remains most pretrained parameters fixed. However, unlike prior methods in which all input tokens are processed with pretrained transformer, CoDA utilizes a router function to select k important input tokens for conditional computation. In this way, CoDA not only enhances parameter efficiency but also inference efficiency. **Tiny-Attn Adapter** (Tiny-Attention Adapter) [18] introduces a dot-product attention module between the down- and up-projections, which can also be seen as a multi-head attention module with its per-head dimensionality to be extremely small. Moreover, the Tiny-Attn Adapter regards its multiple attention heads as a mixture of experts and averages their weights to further reduce inference costs. Akin to the sequential adapter, the Tiny-Attn Adapter is also injected right after the multi-head attention layer.

AdapterFusion [19] integrates multiple task-specific adapters into a single adapter module, allowing for effective knowledge transfer across related tasks without modifying the original pretrained model. AdapterFusion provides a practical and efficient approach to task composition, enabling the transferability of pretrained models across multiple tasks while minimizing the computational costs associated with fine-tuning the entire model. However, AdapterFusion requires additional trainable parameters in the composition layers, increasing computational costs. **MerA** (Merging Pretrained Adapters) [20] adopts summation and averaging strategies to merge the parameters of pretrained adapters without introducing extra trainable parameters. It employs the optimal transport method [78], [79] to align the parameters of adapters based on weights and activations, which gives better performance with fewer trainable parameters compared to AdapterFusion. **Hyperformer++** [21] utilizes the shared hypernetwork [80] to learn task-specific and layer-specific adapter parameters that condition on task and layer id embeddings. By sharing



(a) Sequential Adapter (b) Prefix-tuning (c) LoRA

Fig. 3: The detailed architecture of (a) **Sequential Adapter**, (b) **Prefix-tuning**, and (c) **LoRA**.

knowledge across tasks via hypernetworks while enabling the model to adapt to each task through task-specific adapters, significantly reducing the number of trainable parameters. **AdapterSoup** [22] is developed to address cross-domain task adaptation, which first trains multiple adapters based on various domains and then employs domain clustering [81] to select the most appropriate top-k adapters for the new domain. Fine-tuning parameters for the new domain in AdapterSoup are determined by calculating the weighted average of the selected k adapters. Apart from cross-domain task adaptation, AdapterSoup can also be used to strengthen in-domain results via weight averaging of adapters trained on the same domain but with different hyperparameters.

2) Soft Prompt-based Fine-tuning: Soft prompt fine-tuning is a class of methods in which trainable continuous vectors, known as soft prompts, are inserted into the input or hidden state of the model. Unlike manually designed hard prompts, soft prompts are generated by searching for prompts in a discrete token space based on task-specific training data. Soft prompts exhibit more flexibility and adaptability during fine-tuning, as these prompts can be optimized and adjusted based on the specific task and training data.

WARP (Word-level Adversarial ReProgramming) [23] inserts special prompt tokens $[P_1], [P_2], \cdots, [P_l]$ and [Mask] token before or after the sentences relying on the prompt template. The training objective is to minimize the cross-entropy loss between the output of MLM and the verbalizer tokens $[V_1], [V_2], \cdots, [V_c]$ for classes $\{1, 2, \cdots, c\}$. Only the parameters of $[P_1], [P_2], \cdots, [P_l]$ and $[V_1], [V_2], \cdots, [V_c]$ are trainable, resulting in a significant reduction in the number of fine-tuning parameters. **Prompt-tuning** [24] incorporates additional l learnable prompt tokens, $P = [P_1], [P_2], \cdots, [P_l]$, into the model input $X \in \mathbb{R}^{n \times d}$ and then concatenates them to generate the final input $\hat{X}$, the new input can be expressed with Equation 7. During fine-tuning, only the prompt parameters of P are updated through gradient descent, while pretrained parameters remain frozen. Thus, the parameter cost of prompt-tuning is determined by multiplying the prompt length by the token embedding dimension, and extending the prompt length beyond a single token is critical for achieving good performance.

$$\hat{X} = \text{Concat}(P, X) = [P, X] \in \mathbb{R}^{(l+n) \times d}. \tag{7}$$

Prefix-tuning [10] proposes to prepend soft prompts $P = [P_1], [P_2], \cdots, [P_l]$ (l denotes the length of the prefix) to the hidden states of the multi-head attention layer, differing from prompt-tuning that adds soft prompts to the input. To ensure stable training, a FFN is introduced to parameterize the soft prompts, as direct optimization of the soft prompts can lead to instability. Two sets of prefix vectors $\hat{P}_k$ and $\hat{P}_v$ are concatenated to the original key (K) and value (V) vectors of the attention layer. The self-attention mechanism with prefix-tuning can be represented by Equation 8. During training, only $\hat{P}_k$, $\hat{P}_v$, and the parameters of FFN are optimized, while all other parameters of PLMs remain frozen. The structure of prefix-tuning is illustrated in Fig. 3. After training, the FFN is discarded, and only P_k and P_v are used for inference. **P-tuning** [25] also considers inserting the soft prompts $[P_1], \cdots, [P_i], [P_{i+1}], \cdots, [P_l]$ into the model input. Nonetheless, P-tuning differs by concatenating these prompts to form a template and maps it to obtain $\{h_1, \cdots, h_i, e(x), h_{i+1}, \cdots, h_l, e(x)\}$, in which e represents pretrained embedding layer. The training goal is to optimize the continuous prompts $\{h_1, \cdots, h_l\}$. As the weights of PLMs are fixed and only a few parameters need to be fine-tuned, the template can be effectively learned in few-shot learning scenarios. P-tuning employs a bidirectional long short-term memory network (LSTM) with a ReLU-activated multilayer perceptron (MLP) to initialize the embedding of soft prompts through MLP(LSTM($h_1, \cdots, h_i$): LSTM($h_i, \cdots, h_l$)).

$$\text{head} = \text{Attn}(XW_q, [\hat{P}_k, XW_k], [\hat{P}_v, XW_v]), \tag{8}$$

$$\hat{P}_k = \text{FFN}(P_k), \hat{P}_v = \text{FFN}(P_v). \tag{9}$$

SPOT (Soft Prompt Transfer) [26] is a multitask prompt method that builds upon the prompt-tuning, in which "prompt pertaining" is introduced between PLMs and prompt-tuning of target tasks. There are two variants of SPOT: *generic*

SPOT and *targeted* SPOT. *Generic* SPOT first learns a generic prompt on one or more source tasks and then employs the learned prompt to initialize target prompts for specific target tasks. *Targeted* SPOT learns separate prompts for various source tasks, creating a source prompt library. Subsequently, the optimal source prompt, which exhibits higher similarity to the target task embedding, is retrieved and used to initialize the target prompt for the target task. **ATTEMPT** (ATTEntional Mixtures of Prompt Tuning) [27] begins by pretraining transferable soft prompts (source prompts) on large-scale source tasks that possess valuable knowledge applicable to other tasks. The new target prompt is initialized specifically for a given target task. ATTEMPT employs a shared and lightweight network that is trained simultaneously to learn an attention-weighted combination of source prompts and target prompt. This enables modular multi-task learning, as pretrained soft prompts can be flexibly combined, reused, or removed to leverage knowledge from different tasks. **MPT** (multitask prompt tuning) [28] utilizes multitask data to decompose and distill knowledge from the source prompts to learn a single shared prompt. MPT then learns a multiplicative low-rank matrix to update the shared prompt, efficiently adapting it to each downstream target task. Specifically, MPT assumes that the soft prompts for the k-th source task, denoted as $\hat{P}_k$, can be decomposed into the shared prompts across all source tasks P^* and a task-specific low-rank matrix W_k. The decomposition is given by $\hat{P}_k = P^* \odot W_k = P^* \odot (u_k \otimes v_k^T)$, where $\odot$ denotes the Hadamard product, $\otimes$ denotes the Kronecker product, and u_k and v_k are task-specific vectors for the task k.

3) Others: Apart from adapters family and soft prompts fine-tuning methods, there are some other approaches that also incorporate extra trainable parameters during fine-tuning. They involve adding a ladder side network operating alongside the transformer, introducing an additional diff vector to rescale the attention, incorporating an extra vector to the special token representations, using the late fusion technique to integrate additional attention weight, or combing an extra joint importance weight for each token representation.

LST (Ladder Side-Tuning) [29] trains a ladder side network in conjunction with the pretrained network and takes intermediate activations as input via shortcut connections, known as ladders, from pretrained network. Since all training parameters are stored in the ladder side network, back-propagation is achieved through side networks and ladder connections rather than pretrained networks, reducing the number of fine-tuned parameters. In addition, LST further boosts parameter efficiency by utilizing structural pruning [82] to retrieve a smaller pruned network to initialize the side network and dropping certain layers of the side network. **(IA)**^{3} (Infused Adapter by Inhibiting and Amplifying Inner Activations) [30] leverages learned vectors to scale activations, leading to improved performance while introducing a relatively small number of new parameters. (IA)^3 introduces three learned vectors, l_k, l_v, and l_{ff}, to rescale the key (K) and value (V) vectors in the attention networks and hidden activations in the position-wise FFN. As a result, the attention output and hidden activation output can be rescaled using the following expressions:

$$\text{Attn}(Q, K, V) = (\frac{Q(l_k \odot K^T)}{\sqrt{d_k}})(l_v \odot V), \tag{10}$$

$$\text{FFN}(X) = (l_{ff} \odot \gamma(XW_1))W_2, \tag{11}$$

in which $\odot$ represents element-wise multiplication, W_1 and W_2 are the weight matrices of FFN, and γ is activation function. (IA)^3 only optimizes three learned vectors l_k, l_v, and l_{ff} for each transformer block, resulting in great parameter efficiency. Notably, (IA)^3 incurs minimal overhead because l_k and l_v can be seamlessly integrated into the corresponding linear layers, with the only additional overhead arising from l_{ff}. **PASTA** (PArameter-efficient tuning with Special Token Adaptation) [31] improves parameter efficiency by modifying the special token representations (e.g., [SEP] and [CLS] in BERT) with an extra special trainable vector before the self-attention layer at each transformer layer. Assuming that the inputs to the transformer layer are denoted as $H = \{h_i\}_{i=1}^N$, PASTA modifies the inputs as follows:

$$H_{mod} = \{h_i + m_i\}_{i=1}^N, \tag{12}$$

$$m_i = e(v_p),\ i \text{ is the } p\text{-th special token; otherwise, } m_i = 0,$$

m_i is the special token adaptation. PASTA enables a remarkable reduction in trainable parameters by training only the trainable vector $e(v_p)$ to update the representations of special tokens. The reasons for using [CLS] and [SEP] as special tokens in PASTA are that the [CLS] representation provides a global representation of the input text and that the attention scores in PLMs are primarily allocated to the [CLS] or [SEP] tokens across attention heads [83], [84]. **AttentionFusion** [32] introduces the late fusion technique, which involves combining features or representations from diverse tasks or layers to generate a final joint representation, to adjust the the importance of each token representation. For a given task t, let the attention query vector be denoted by Q^t and the representation of token i at layer j be V_i^j, then the representation of token i for task t, $\hat{V}_i^j$, is expressed as:

$$\hat{V}_i^j = \sum_j \alpha_i^j(t) V_i^j, \quad \alpha_i^j(t) = \frac{\exp(Q^t V_i^j)}{\sum_k \exp(Q^t V_i^j)}, \tag{13}$$

where $\alpha_i^j(t)$ represents the attention weight of token i at layer j for task t. The number of extra parameters that need to be updated in AttentionFusion is determined by the size of the query vector Q^t, which is the same as the hidden dimension of the pretrained encoder. By employing the attention weight as extra trainable parameters, AttentionFusion adjusts the importance of each token representation dynamically. **Hadamard Adapter** [33] is an additive fine-tuning method that introduces a weight vector and a bias vector in each transformer with the same dimensions as the output of the multi-head attention module for fine-tuning. A weight vector and a bias vector are injected right after the multi-head attention layer to perform element-wise multiplication (Hadamard product) with the multi-head attention outputs. Notably, the number of Hadamard adapter is the same as that of the transformer layers in the PLM. During fine-tuning, only the parameters in

TABLE I: The weight update in pretrained weight masking and delta weight masking. $\odot$ denotes the Hadamard product.

Method	Weight Update	Mask Criterion	Mask Matrix
Threshold-Mask	$\hat{W} = M \odot W$	Threshold	$M = \mathbb{I}_{s_{i,j}>\tau}$
FISH Mask	$\hat{W} = M \odot W$	Fisher information	$M = \mathbb{I}_{top-k(f_{i,j})}$
LT-SFT	$\hat{W} = W + M \odot \nabla_W \mathcal{L}(W)$	Absolute difference of parameters	$M = \mathbb{I}_{top-k(\lvert W_1 - W_0 \rvert)}$
Child-Tuning$_F$	$\hat{W} = W - M \odot \eta \nabla_W \mathcal{L}(W)$	Bernoulli distribution	$M = \{0,1\}^n$
Child-Tuning$_D$	$\hat{W} = W - M \odot \eta \nabla_W \mathcal{L}(W)$	Fisher information	$M = \mathbb{I}_{top-k(f_{i,j})}$
Diff Pruning	$\hat{W} = W + M \odot \Delta W$	Fixed sparsity	$M = \{0,1\}^n$
SAM	$\hat{W} = W + M\Delta W$	Analytical solution	$M_{i,j} = 0, \forall i \neq j; M_{i,i} \in \{0,1\}$

the Hadamard adapter, layer normalization, and classifier are updated.

B. Partial Fine-tuning

Partial fine-tuning methods aim to reduce the number of fine-tuned parameters by selecting a subset of pre-trained parameters that are critical to downstream tasks while discarding unimportant ones. We categorize partial fine-tuning methods into three groups: **Bias Update** [34], [35], in which only the bias term in the attention layer, feed-forward layer and layer normalization of the transformer is updated, **Pretrained Weight Masking** [36], [37], where the pretrained weights are masked using various pruning criterion, and **Delta Weight Masking** [38], [39], [40], [41], in which delta weights are masked via pruning techniques and optimization approximation. A detailed analysis of pretrained weight and delta weight masking is provided in Table I.

1) Bias Update: **Bit-Fit** (Bias-term Fine-tuning) [34] achieves parameter efficiency by only updating the bias terms and the task-specific classification layer while keeping the majority of parameters in the transformer-based PLMs frozen. The bias parameters are involved in the attention layer, where they are involved in calculating query, key, value, and combining multiple attention heads, as well as in the fee-forward and layer normalization layers. Further, **U/S-BitFit** [35] combines NAS algorithm [85] and pruning technique to automatically determine which parameters of the network need to be fine-tuned based on BitFit. **U-BitFit** (Unstructured BitFit) decides which PEFT parameters to prune based on the first-order approximation of the change in training loss resulting from pruning the PEFT parameter W, i.e., $-W \cdot \nabla_W \mathcal{L}(W)$. While **S-BitFit** (Structured BitFit) sums the criterion over the overall bias update Δb (b is the bias term).

2) Pretrained Weight Masking: Pretrained weight masking employs pruning criteria like threshold and Fisher information to measure the importance of pretrained weight to construct a binary mask matrix for weight masking. **Threshold-Mask** [36] utilizes the threshold to construct a binary mask matrix to select pretrained weights W of the attention and FFN layers through element-wise multiplication, expressed as $\hat{W} = W \odot M$ ($\odot$ denotes the Hadamard product). To begin, a random uniformly distributed real-valued matrix S, which shares the same dimensions as matrices W and M, is created. Subsequently, if an element in S surpasses a predetermined global threshold τ, the corresponding position in the binary mask matrix is assigned a value of 1; otherwise, it is assigned 0. **FISH Mask** (Fisher-Induced Sparse uncHanging) [37] uses the Fisher information of pretrained weight to measure their importance and construct a sparse binary mask. FISH Mask selects the top-k parameters with the largest Fisher information to construct the sparse binary mask, where the positions corresponding to the top-k parameters are set to be 1 and the rest are set to 0. Note that k is preset based on the desired mask sparsity level of the mask, and the resulting sparse binary mask can be reused across many subsequent iterations.

3) Delta Weight Masking: Delta weight masking also employs various pruning techniques and criteria to construct a binary mask matrix to reduce trainable parameters. However, Delta weight pruning typically involves an update at each iteration. **LT-SFT** (Lottery Ticket Sparse Fine-Tuning) [38] is a novel PEFT method inspired by the Lottery Ticket Hypothesis[2] [86]. LT-SFT first fine-tunes the PLM on target data using pretrained parameters W_0 to obtain the fully fine-tuned parameters W_1, and then identifies the top-k pretrained parameters with the greatest absolute differences ($|W_1 - W_0|$). The top-k parameters are selected for further fine-tuning using binary mask M, in which the positions corresponding to the selected k parameters are set to 1 and the remaining positions to 0. LT-SFT then resets the model parameters to their original pretrained weights W_0 but fine-tunes only the selected k parameters while keeping the remaining parameters frozen, and can be expressed as $\delta = M \odot \Delta W (\Delta W = \nabla_W \mathcal{L}(W)$. By iteratively repeating this process, the method gradually fine-tunes only a small fraction of the model's parameters. **Child-Tuning** [39] calls the network formed by the parameters to be updated a child network and masks out the gradients of non-child networks to improve parameter efficiency. The parameter updates in Child-Tuning is expressed as $\delta = M \odot \Delta W$ ($\Delta W = \eta \nabla_W \mathcal{L}(W)$, η denotes learning rate). Child-Tuning provides two variants: Child-Tuning$_F$ (F stands for Task-Free) and Child-Tuning$_D$ (D stands for Task-Driven). Child-Tuning$_F$ generates the binary mask matrix M using Bernoulli distribution with a probability denoted as p_F. Increasing the value of p_F updates a larger number of parameters, and Child-Tuning$_F$ is equivalent to full fine-tuning when $\mathrm{p}_F = 1$. In contrast, Child-Tuning$_D$ uses Fisher information estimation to identify a subset of parameters (i.e., child network) that are highly correlated with a specific downstream task. The binary

[2]Lottery Ticket Hypothesis states that each neural model contains a sub-network (a "winning ticket") that can match or even outperform the performance of the original model when trained in isolation.

mask matrix in Child-Tuning$_D$ is constructed by setting the position of the child network to be 1 and the non-child network to be 0.

Diff Pruning [40] introduces a sparse task-specific "diff" vector δ during fine-tuning while remaining the pretrained model parameters fixed. To make the diff vector δ sparse, Diff Pruning introduces a learnable binary mask M on the Delta weight and decomposes $\delta = M \odot \Delta W$. The binary mask M is learnable and is used as a regularizer during fine-tuning. It acts as a differentiable approximation to the L0-norm of diff vector δ. This approach is well-suited for multi-task deployment in edge (mobile) applications with limited storage. Significantly, Diff pruning incurs higher memory consumption compared to traditional fine-tuning, which may become problematic as model sizes continue to grow. **SAM** (Second-order Approximation Method) [41] also employs the sparse mask matrix to update the delta weight. However, SAM directly optimizes the approximation function to obtain an analytical solution for the mask matrix, which is then used to update the pretrained weight. Concretely, SAM [41] views the PEFT methods as p-sparse fine-tuned model by representing fine-tuned parameter as $W = W_0 + M\Delta W$. M is a mask matrix, and the optimization problem is

$$\min_{\Delta W, M} \mathcal{L}(W_0 + M\Delta W), \quad s.t. \quad \|M\|_0 = \lfloor mp \rfloor, M_{i,j} = 0, \forall i \neq j; \text{and } M_{i,j} \in \{0, 1\}.$$

SAM approximates the loss function using its second-order Taylor expansion as:

$$\begin{aligned}\mathcal{L}(W_0 + M\Delta W) \approx & \mathcal{L}(W_0) + \Delta\mathcal{L}(W_0)^T \Delta\mathcal{L}(W_0)^T M\Delta W \\ & + \frac{1}{2}(M\Delta W)^T H M\Delta W, \end{aligned} \tag{14}$$

in which H is the Hessian matrix. In practice, SAM first obtains the gradient $\nabla\mathcal{L}(W_0)_i$ for the i-th parameter W_i, then calculates $\left|\nabla\mathcal{L}(W_0)_i^2\right|$, and selects the top $\lfloor mp \rfloor$ delta weight for optimization.

C. Reparameterized Fine-tuning

Reparameterized fine-tuning methods utilize low-rank transformation to reduce the number of trainable parameters while allowing operating with high-dimensional matrices (e.g., pre-trained weights). We categorize reparameterized fine-tuning methods into two groups: **Low-rank Decomposition** [11], [42], [43], in which various low-rank decomposition techniques are used to reparameterize the updated matrix, and **LoRA derivatives** [44], [45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55], [56], [57], where a series of PEFT methods are developed based on LoRA. Specific details of ΔW parameters reparameterization of various approaches can be seen in Table II.

1) Low-rank Decomposition: This involves finding a lower-rank matrix that captures the essential information of the original matrix while reducing computational complexity and memory usage by reparameterizing the updated delta weight. Reparameterization covers transforming the delta weight matrix into a low-rank representation using methods such as Fastfood transformation, low-rank down-up projection, or Kronecker product projection.

Intrinsic SAID (Structure-Aware Intrinsic Dimension) [42] leverages the concept of intrinsic dimensionality to reduce the number of parameters during fine-tuning. The intrinsic dimensionality refers to the minimum dimensionality required to solve a high-dimensional optimization problem. In the context of PLMs, measuring the intrinsic dimensionality helps estimate the minimum number of parameters needed to adapt to new tasks. Instead of optimizing the empirical loss in the original parameterization, Intrinsic SAID fine-tunes the model by reparametrization the model in a lower-dimensional space, i.e., $\Delta W = F(W^r)$, in which W^r is the parameter to be optimized and $F : \mathbb{R}^r \to \mathbb{R}^d$ is a Fastfood transform[3] [87] that projects parameters from low-dimensional r to high-dimensional d. However, Intrinsic SAID is not practical for fine-tuning larger networks due to the $\mathcal{O}(d)$ memory complexity of the Fastfood transform and the need to update all of the model's parameters.

Inspired by Intrinsic SAID, **LoRA** (Low-Rank Adaptation) [11] introduces two trainable low-rank matrices for weight update. In LoRA, a down-projection matrix and an up-projection matrix are utilized in parallel with the query (Q), key (K), and value (V) matrices in the attention layer of the transformer, shown in Fig. 3. For a pretrained weight matrix $W \in \mathbb{R}^{d\times k}$, LoRA updates W using low-rank decomposition $\Delta W = W_{down} W_{up}$. During training, the weights of PLM are frozen, and only the low-rank matrices of LoRA, i.e., $W_{down} \in \mathbb{R}^{d\times r}$ and $W_{up} \in \mathbb{R}^{r\times k}$ are fine-tuned ($r \ll \{d, k\}$). During inference, the LoRA weights are merged with the original weight matrix of the PLMs without increasing the inference time. Practically, a scaling factor ($s = 1/r$) is added to the LoRA module. **KronA** (Kronecker Adapter) [43] is structurally similar to LoRA but replaces the low-rank decomposition in LoRA with Kronecker product decomposition, $\Delta W = W_{down} \otimes W_{up}$. Kronecker product decomposition maintains the rank of the input matrix (i.e., $\text{rank}(A \otimes B) = \text{rank}(A) \times \text{rank}(B)$), ensuring that important information is preserved during the adaptation process. Moreover, Kronecker product can speed up computation and reduce the number of required floating-point operations (FLOPS) by avoiding the explicit reconstruction of the Kronecker product matrix. KronA has two variants: KronA$_B$ and KronA$^B_{\text{res}}$. KronA$_B$ inserts the KronA module in parallel to the FFN layer, while KronA$^B_{\text{res}}$ inserts the KronA module alongside the FFN layer and incorporates a learnable residual connection.

2) LoRA Derivatives: LoRA derivatives refer to a series of PEFT methods that are improved based on LoRA, including **Low-Rank Adjustment** [44], [45], [46], where different methods are developed to adjust the rank of LoRA dynamically, **LoRA-guided Pretrained Weight Update** [47], [48], in which LoRA is used to guide the update of pretrained weight, **Quantization Adaption** [49], [50], [51], in which various quantization techniques are proposed to improve the high precision fine-tuning and inference of LoRA, **LoRA-**

[3]Fastfood transform is a computationally efficient dimensionality expansion method.

TABLE II: Delta weight reparameterization of various reparameterized fine-tuning methods.

Method	ΔW Reparameterization	Notes
Intrinsic SAID	$\Delta W = F(W^r)$	$F : \mathbb{R}^r \to \mathbb{R}^d$, $W^r \in \mathbb{R}^r$ is parameters to be optimized, and $r \ll d$.
LoRA	$\Delta W = W_{dowm} W_{up}$	$W_{down} \in \mathbb{R}^{k\times r}$, $W_{up} \in \mathbb{R}^{r\times d}$, and $r \ll \{k, d\}$.
KronA	$\Delta W = W_{down} \otimes W_{up}$	$\text{rank}(W_{down} \otimes W_{up}) = \text{rank}(W_{down}) \times \text{rank}(W_{up})$.
DyLoRA	$\Delta W = W_{down\downarrow b} W_{up\downarrow b}$	$W_{down\downarrow b} = W_{down}[: b, :]$, $W_{up\downarrow b} = W_{up}[:, : b]$, $b \in \{r_{min}, \cdots, r_{max}\}$.
AdaLoRA	$\Delta W = P\Lambda Q$	$PP^T = P^T P = I = QQ^T = Q^T Q$, $\Lambda = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r)$.
IncreLoRA	$\Delta W = W_{down}\Lambda W_{up}$	$\Lambda = [\lambda_1, \lambda_2, \cdots, \lambda_r]$ with λ_i could be an arbitrary constant.
DeltaLoRA	$\Delta W = W_{down} W_{up}$	$W^{(t+1)} \leftarrow W^{(t)} + (W_{down}^{(t+1)} W_{up}^{(t+1)} - W_{down}^{(t)} W_{up}^{(t)})$.
LoRAPrune	$\Delta W = W_{down} W_{up} \odot M$	$\delta = (W + W_{down} W_{up}) \odot M$, $M \in \{0, 1\}^{1\times G}$, G is group number.
QLoRA	$\Delta W = W_{down}^{BF16} W_{up}^{BF16}$	$Y^{BF16} = X^{BF16} \text{doubleDequant}(c_1^{FP32}, c_2^{FP8}, W^{NF4}) + X^{BF16} W_{down}^{BF16} W_{down}^{BF16}$.
QA-LoRA	$\Delta W = W_{down} W_{up}$	$W_{down} \in \mathbb{R}^{k\times r}$, $W_{up} \in \mathbb{R}^{r\times L}$, L is the quantization group number of W.
LOFTQ	$\Delta W = \text{SVD}(W - Q_t)$	$Q_t = q_N(W - W_{down}^{t-1} W_{up}^{t-1})$, q_N is N-bit quantization function.
Kernel-mix	$\Delta W^h = (B_{LoRA}, B^h) \begin{pmatrix} A_{LoRA}^h \\ A^h \end{pmatrix}$	B_{LoRA} is shared across all heads, B^h, A^h provide rank-r update in each head.
LoRA-FA	$\Delta W = W_{down} W_{up} = QRW_{up}$	W_{down} is frozen, and only update W_{up}.

based Improvements [52], [53], [54], in which several novel technique are incorporated into LoRA for improvements, and **LoRA-based Multi-task Fine-tuning** [55], [56], [57], where multiple LoRA modules are combined for cross-task transfer to fine-tune model on a novel task.

Low-rank Adjustment. DyLoRA (Dynamic LoRA) [44] is introduced to overcome two limitations of LoRA: (a) LoRA's rank is fixed and prevents any changes after training (b) determining the optimal rank for LoRA requires exhaustive search and considerable effort. DyLoRA trains LoRA modules for a range of ranks instead of a single rank, allowing for adaptability. DyLoRA addresses these limitations during training by sorting the representations learned at various ranks. Specifically, DyLoRA operates within the range of ranks denoted as $r \in [r_{min}, r_{max}]$ for a series of iterations. In each iteration, DyLoRA randomly selects a specific rank b from $\{r_{min}, \cdots, r_{max}\}$. It then truncates the down-projection matrix as $W_{down\downarrow b} = W_{down}[: b, :]$ and the up-projection matrix as $W_{up\downarrow b} = W_{up}[:, : b]$ and only update truncated parameter matrices $W_{down\downarrow b}$ and $W_{up\downarrow b}$. The parameter updates in each iteration of DyLoRA could be expressed as $\Delta W = W_{down\downarrow b} W_{up\downarrow b}$. By allowing dynamic low-rank adaptation and search-free low-rank adaptation, DyLoRA reduces the computational cost and training time required to identify the optimal rank for a particular task. **AdaLoRA** (Adaptive Low-Rank Adaptation) [45] extends LoRA by dynamically adjusting the rank of matrices to control the allocation budget. In AdaLoRA, the incremental update ΔW is reparameterized using singular value decomposition (SVD) and then truncates the smallest singular values, i.e., $\Delta W = P\Lambda Q$. Both P and Q are orthogonal matrices, and Λ is a diagonal matrix containing the singular values $\{\sigma_1, \sigma_2, \ldots, \sigma_r\}$. Here, r represents the rank of the matrix Λ. During training, P and Q are initialized with Gaussian distribution with a regularizer to ensure the orthogonality, while Λ is initialized with zero and iteratively pruned to adjust the rank. AdaLoRA employs the sensitivity-base importance scoring [88], [89] with a new metric to prune the singular values of unimportant updates to update the Λ. By doing this, AdaLoRA effectively improves parameter efficiency and allocation budgets. **IncreLoRA** [46] dynamically incorporates trainable parameters into LoRA by increasing their ranks, guided by importance scores assigned to each module during training. The allocation process assigns lower ranks, possibly 0 to indicate no parameter updates, to less important modules, while allocating higher ranks to more important modules. The parameter updates in IncreLoRA can be expressed as $\Delta W = W_{down}\Lambda W_{up}$, in which $\Lambda = [\lambda_1, \lambda_2, \cdots, \lambda_r]$ is a diagonal matrix with λ_i could be any arbitrary constant, r is the rank of the each LoRA module. Besides, an upper bound on the rank is set for each module to control the parameter growth. Additionally, IncreLoRA introduces a unique pretraining technique called "advance learning", which ensures that the newly added parameters in each module begin with favorable initial states. In this way, it prevents insufficient training of subsequently added parameters, allowing for effective utilization of the incremental parameter allocation. Unlike LoRA, which operates on the query (Q), key (K), and value (V) projection modules of the attention layer, the parameter updates are applied to all linear layers in IncreLoRA.

LoRA-guided Pretrained Weight Update. Delta-LoRA [47] updates the pretrained weight W as well as two low-rank matrices W_{down} and W_{up}, while using the same memory as the original LoRA. The two low-rank matrices W_{down} and W_{up} are automatically updated as usual. The pretrained weight, however, leverages the mathematical property that $\nabla_W \mathcal{L}(W, W_{down}, W_{up}) = \nabla_{W_{down}W_{up}} \mathcal{L}(W, W_{down}, W_{up})$ (it is achieved by removing the dropout layer in the original LoRA module) for parameters update. Specifically, W is updated with the delta of the product of two low-rank matrices in consecutive iterations, i.e., $W \leftarrow W + \Delta W_{down} W_{up} = W + (W_{down}(t+1)W_{up}(t+1) - W_{down}(t)W_{up}(t))$. **LoRAPrune** [48] introduces a LoRA-guided pruning criterion, which utilizes the weights and gradients of LoRA instead of the gradients of pretrained weights for importance estimation to prune parameters of LoRA and pretrained weights. To address the substantial memory overhead associated with unstructured pruning and dependency-aware structured pruning,

LoRAPrune devises a structured iterative pruning procedure that selectively eliminates redundant channels and heads. LoRA-guided pruning criterion involves using low-rank matrices W_{down} and W_{up}, along with their corresponding gradients $\nabla_{W_{down}}$ and $\nabla_{W_{up}}$, to calculate the importance score[4]. This score determines which weights are deemed unimportant and subsequently removed. Notably, LoRAPrune not only prunes structured weights, such as heads and channels, from the pretrained weights, but also prunes the corresponding weights in the LoRA, i.e., $\delta = (W + W_{down}W_{up}) \odot M$, $M \in \{0,1\}^{1\times G}$, G is the group number. Binary mask M is set to 0 when the corresponding group is unimportant, and 1 when it is important. Therefore, after pruning and fine-tuning, the LoRA weights can seamlessly merge with the pretrained weights, ensuring that no additional computations are necessary during inference.

Quantization Adaption. QLoRA [49], a quantized variant of LoRA, effectively addresses the limited computational resource of LoRA for fine-tuning LLMs by quantizing the transformer model to 4-bit NormalFloat (NF4) precision with double quantization processing, and using a paged optimizer to deal with memory spikes. NF4 is a new data type that is theoretically optimal for normally distributed weights. Although QLoRA quantizes pretrained weight W from FP16 into NF4 so that LLMs can be fine-tuned with fewer GPUs, the auxiliary weight of LoRA matrix $W_{down}W_{up}$ makes the final weight return to FP16 again after fine-tuning. To this end, **QA-LoRA** (Quantization-Aware Low-rank Adaption) [50] employs group-wise quantization with low-rank adaptation to the pretrained weight W, in which each column of W is partitioned into L groups for quantization. In this way, QA-LoRA ensures that pretrained weights W and auxiliary weights are integrated into a quantized form after fine-tuning, resulting in a faster and more accurate computation during inference. While **LOFTQ** (LoRA-Fine-Tuning-aware Quantization) [51] applies an N-bit quantized weight Q and low-rank approximation $W_{down} \in \mathbb{R}^{d_1\times r}$, $W_{up} \in \mathbb{R}^{d_2\times r}$ to approximate the original high-precision pretrained weight $W \in \mathbb{R}^{d_1\times d_2}$ as the initialization of LoRA fine-tuning. Such an initialization alleviates the quantization discrepancy in QLoRA and significantly improves the generalization in downstream tasks.

LoRA-based Improvements. Kernel-wise Adapter [52] treats the different attention heads in the transformer as independent kernel estimators and utilizes the kernel structure in self-attention to guide the assignment of tunable parameters. LoRA is used as the underlying model to combine kernel-wise adaptation for its flexibility in parameter assignment for different weight matrices. Kernel-wise adapter has two variants: Kernel-mix-lite (qv) and Kernel-mix-lite (qvo). Kernel-mix-lite (qv) provides a lightweight solution for scenarios with limited parameter budgets, while Kernel-mix (qvo) is suitable for scenarios with intermediate parameter budgets. The suffix (qv) means that the method will adjust W_q and W_v, while the suffix (qvo) means that the method will modify W_q, W_v, and W_o. **Laplace-LoRA** [53] incorporates Bayesian inference into the LoRA parameters to address the issue of overconfidence and improve calibration. A key challenge lies in obtaining the posterior distribution for Bayesian inference, which is resolved by using Laplace approximation [90]. Laplace-LoRA can be viewed as an approximation of the posterior distribution over LoRA parameters using Laplace approximation. Hence, Laplace-LoRA maintains existing pretraining and fine-tuning procedures while reducing the dimensionality of Bayesian inference. **LoRA-FA** (LoRA with Frozen-A) [54] is proposed to reduce the expensive activation memory of LoRA without introducing any computational overhead. LoRA-FA keeps the pretrained weight W and down-projection matrix W_{down} frozen and only updates the up-projection matrix W_{up}. W_{down} is decomposed into Q and R via QR decomposition, and $\Delta W = W_{down}W_{up} = QRW_{up} = Q\hat{W}_{up} = \sum_{i=1}^{r} Q_{:,i}\hat{W}_{up,i,:}$, in which $\{Q_{:,i}\}_{i=1}^{r}$ are orthogonal unit vectors (r is the rank of W_{down}). Thus, ΔW is a combination of r orthogonal vectors, limiting the change of weight residing in a low-rank space. Consequently, there is no need to store full-rank input activations simultaneously, alleviating the memory burden associated with activation storage.

LoRA-based Multi-task Fine-tuning. LoRAHub [55] leverages a composition of multiple trained LoRA modules for cross-task transfer to fine-tune the model on new tasks. Specifically, LoRAHub trains task-specific LoRA modules in a variety of tasks to obtain a synthesized module, $\hat{m} = (w_1W_{down}^1 + \cdots + w_NW_{down}^N)(w_1W_{up}^1 + \cdots + w_NW_{up}^N)$, which is then amalgamated with the LLMs to adapt the new task. Thus, the objective of LoRAHub is to find the best weight set $\{w_1, w_2, \cdots, w_N\}$, which is achieved by the gradient-free combinatorial optimization approach Shiwa [91]. **MOELoRA** [56] combines LoRA with mixture-of-experts (MoE) for multi-task fine-tuning, in which each expert is a LoRA module for learning task-specific knowledge. Additionally, MOELoRA devises a task-motivated gate function to produce distinct fine-tuned parameters for various tasks. **L-LoRA** (Linearized LoRA) [57] is a linearized PEFT method to improve the multi-task fusion capability of fine-tuned task-specific models with low computation costs. L-LoRA constructs a linear function using a fir-order Taylor expansion, as illustrated in Equation 15. In L-LoRA, only the linearized LoRA modules are fine-tuned in the tangent space, incurring fewer trainable parameters compared to LoRA. For the multi-task fusion methods, simple average, task arithmetic [92], [93], ties-merging [94], and LoRAhub [55] are employed for multi-task fusion.

$$f_{\theta_0}(x;\phi(t)) \approx f_{\theta_0}^{\text{lin}}(x;\phi(t)) = f_{\theta_0}(x;\phi(0)) + \nabla_\phi f_{\theta_0}(x;\phi(0))^T(\phi(t)-\phi(0)). \quad (15)$$

D. Hybrid Fine-Tuning

Hybrid fine-tuning approaches aim to combine various PEFT approaches, such as adapter, prefix-tuning, and LoRA, to leverage the strengths of each method and mitigate their weaknesses. By integrating different features of PEFT methods, hybrid fine-tuning achieves improved overall performance compared to individual PEFT methods. These works are classified into two approaches: **Mannual Combination**

[4]Importance score I is calculate via $I = \nabla_W \odot W, \nabla_W \approx W_{down} \cdot \nabla_{W_{up}} + \nabla_{W_{down}} \cdot W_{up} - \nabla_{W_{down}} \cdot \nabla_{W_{up}}$.

[16], [35], [58], [59], in which multiple PEFT methods are combined manually by sophisticated design, and **Automatic Combination** [60], [61], [62], where various PEFT methods are incorporated automatically via structure search.

1) Manual Combination: Manual combination mainly involves integrating the structure or features of one PEFT method into another PEFT method to enhance performance while achieving parameter efficiency. **MAM Adapter** (Mix-And-Match Adapter) [16] is the combination of scaled parallel adapter and prefix-tuning, which employs prefix-tuning with smaller bottleneck dimensions at the attention layer and allocates more parameter budget to modify the representation of FFN using the scaling parallel adapter. Scaled parallel adapter denotes the parallel adapter with a scaling factor to adjust the adapter output. Concretely, the output of the MAM adapter h can be expressed with $h = \text{LN}(X + \text{scale} * \text{FFN}(\text{LN}(\text{Attn}([P_k, X]) + [P_v, X])))$ for the input X. Further, **U-MAM** (Unstructured MAM) and **S-MAM** (structured MAM) [35] are proposed by combining NAS algorithm [85] and pruning technique to automatically determine which parameters of the network need to be fine-tuned based on MAM adapter. NAS algorithm takes the maximum number of parameters required for PEFT architectures as input and applies the pruning operation to reduce trainable parameters. The criteria for deciding which PEFT parameters to prune are based on the first-order approximation of the change in training loss resulting from pruning the PEFT parameter W, i.e., $-W \cdot \nabla_W \mathcal{L}(W)$. U-MAM directly employs this criterion to prune the parameters in MAM, while S-MAM sums the criterion over each column of W_{down}.

Compacter [58] is developed based on adapters, low-rank optimization, and a parameterized hypercomplex multiplication (PHM) layer [95]. It follows a similar structure to adapters, consisting of a down-projection, a nonlinear activation function, and an up-projection. However, Compacter replaces the down-projection and up-projection in the adapters with the low-rank parameterized hypercomplex multiplication (LPHM) layer, which is an extension of PHM that incorporates low-rank optimization. Structurally, PHM layer resembles a fully connected layer, but with the learned W represented as a sum of Kronecker products, i.e., $W = \sum_{i=1}^{n} A_i \otimes B_i$. Notably, when the weights of down-projection and up-projection are calculated as in that of the PHM layer, A_i is a shared parameter across all adapter layers, while B_i represents adapter-specific parameters. This kind of adapter is called **PHM Adapter**. Similarly, Compacter obtains the weight matrix in each LPHM layer utilizing the sum of Kronecker products, but Compacter reparameterizes B_i as the product of two independent ranks with one weight, and the weight matrix in Compacter is calculated as follows:

$$W = \sum_{i=1}^{n} A_i \otimes B_i = \sum_{i=1}^{n} A_i \otimes (s_i t_i^{\mathrm{T}}). \tag{16}$$

$W \in \mathbb{R}^{k \times d}, A_i \in \mathbb{R}^{n \times n}, B_i \in \mathbb{R}^{\frac{k}{n} \times \frac{d}{n}}; s_i \in \mathbb{R}^{\frac{k}{n} \times r}, t_i \in \mathbb{R}^{r \times \frac{d}{n}}$.

Compacter++ is a variant of Compacter that inserts a Compacter layer after the FFN layer of each transformer module and requires fewer parameters to be updated than Compacter.

UniPELT [59] incorporates sequential adapter, prefix-tuning, and LoRA via a gating mechanism. In UniPELT, adapters are added after the feed-forward layer, prefix-tuning is employed to the key (K) and value (V) vectors of the multi-head attention layer, and LoRA is used in attention matrices of W_q and W_v of the transformer. Each PEFT module is equipped with a gating mechanism composed of a linear function with the dimension of the output being 1, a sigmoid function, and a mean function. The gating mechanism controls the activation of each submodule, dynamically assigning higher weights to submodules that make positive contributions to a given task. The trainable parameters encompass low-rank LoRA matrices W_{down} and W_{up}, prefix-tuning parameters P_k and P_v, adapter parameters, and weights for the gating function. Consequently, UniPELT requires more parameters and inference time than adapter, prefix-tuning, and LoRA, but achieves better performance compared with the performance of the best individual PEFT method.

2) Automatic Combination: Automatic combination explores how to configure PEFT methods like adapters, prefix-tuning, BitFit, and LoRA to different layers of the transformers automatically using various structure search and optimization approaches. However, it typically requires more time and cost due to the need to perform optimization searches in the model or structure. **AutoPEFT** [60] integrates sequential adapter, parallel adapter, and prefix-tuning into the transformer block. The serial adapter receives the hidden state from the FFN output as input, while the parallel adapter takes the hidden state before the FFN layer as its input. In addition, the prefix-tuning module concatenates two prefix vectors, P_k and P_v, with the original key and value vectors, respectively, enabling multi-head attention to adapt to specific target tasks. Motivated by the success of NAS algorithm, AutoPEFT proposes to use the Bayesian optimization approach to automatically search for an appropriate neural architecture network that selectively activates certain layers to incorporate these PEFT modules. Bayesian optimization is not only sample-efficient and zeroth-order but also well-suited for multi-objective setups, enabling cost-efficient optimization and facilitating the trade-off between performance and cost. Moreover, it is more parallelizable during search, which can decrease memory usage.

S^{3}Delta-M (Search for Sparse Structure of Delta Tuning Mix) [61] is a mixture of LoRA, Compacter (low-rank adapter), BitFit, and LNFit[5]. Different from the simple incorporation of PEFT techniques, S^{3}Delta-M is developed by conducting a differentiable delta tuning structure search. It explicitly controls sparsity and searches for an optimal combination of these techniques in a unified search space. In S^{3}Delta-M, each PEFT module (LoRA, Compacter, BitFit, and LNFit) is inserted into the corresponding layers of the PLM to ensure the best performance is achieved. The specific combination and placement of these modules are determined through the structure search process, which is guided by explicit sparsity control. $\mathcal{S}_4$ [62] is a combination of Sequential Adapter, Prefix-tuning, BitFit, and LoRA. Unlike previous

[5]LNFit is trained only on the variance vectors in the layer normalization module of the PLMs, inspired by [86] which trains only on the batch normalization module in convolutional neural networks.

methods that utilize the same PEFT module uniformly across all layers of the transformer, $\mathcal{S}_4$ is designed by searching for various layer groupings, trainable parameter allocations, tunable groups, and PEFT module assignments. In $\mathcal{S}_4$, the layers of the PLMs are divided into four groups, G_1, G_2, G_3, G_4, in a "spindle" pattern. This means that more layers are allocated to the middle groups (G_2 and G_3) while fewer layers are assigned to the top and bottom groups (G_1 and G_4). However, all trainable parameters are allocated uniformly, i.e., the number of trainable parameters in each layer remains the same across all groups. Different groups are equipped with different combinations of sequential adapter, prefix-tuning, BitFit, and LoRA. Extensive experimental results demonstrate that better performance is achieved when each group is equipped with the following combinations of PEFT methods. A denotes sequential adapter, P denotes prefix-tuning, B denotes BitFit, and L denotes LoRA.

$$G_1 : (A, L); \quad G_2 : (A, P);$$
$$G_3 : (A, P, B); \quad G_4 : (P, B, L).$$

E. Unified Fine-tuning

Unified fine-tuning presents a unified framework for fine-tuning, which streamlines the incorporation of diverse fine-tuning methods into a cohesive architecture, ensuring consistency and efficiency across the adaptation and optimization of models. Unlike hybrid fine-tuning methods, unified fine-tuning methods typically utilize a single PEFT method rather than a combination of various PEFT methods.

AdaMix [63] leverages a mixture of adaptation module approaches to obtain a unified framework for fine-tuning. Motivated by sparsely-activated MoE [96], AdaMix treats each adaptation module as an individual expert and employs stochastic routing to randomly select a down-projection matrix and an up-projection matrix for weight updates. Such stochastic routing allows the adaption module to learn multiple views for the given task, but it also poses a challenge in deciding which adaption module to use during inference. To this end, Adamix utilizes consistency regularization and adaption module merging (i.e., average weights of all down- and up-projection matrices) to select the trained adaption module and obtain the same computational cost as that of a single module. Notably, adaption modules in Adamix could be adapters like sequential adapter [9] or low-rank decomposition matrices like LoRA [11].

SparseAdapter [64] utilizes network pruning technique to construct a unified framework in which various PEFT methods, including adapters family and LoRA [9], [11], [16], can be further pruned to improve parameter efficiency. SparseAdapter sets a target sparsity, denoted as s, and assigns a score, denoted as z, to all parameters of adapters and LoRA. Parameters with scores below the threshold z_s (corresponding to the s-th lowest percentile of z) are considered redundant and removed. The score z can be computed using pruning methods, such as random pruning, magnitude pruning [97], Erdos-Renyi [98], SNIP [99], or GraSP [100], based on the adapter weight W, with SNIP-based SparseAdapter yielding the best results. Furthermore, SparseAdapter exhibits improved performance compared to full fine-tuning when utilizing the "Large-Sparse" setting, which involves larger bottleneck dimensions and higher sparsity ratios. Notably, the network pruning technique proposed in SparseAdapter is a plug-in method that can be applied to any adapter variants, such as LoRA [11], MAM Adapter [16], and AdapterFusion [19]. The optimized parameters in SparseAdapter can be represented as $\hat{W} = W \odot M$, in which M is a binary mask matrix with $M = \mathbb{I}_{\{z \geq z_s\}}$ and $z = \text{score}(W)$.

ProPETL [65] introduces a single prototype network (e.g., adapter, prefix-tuning, and LoRA) across layers and tasks and constructs different sub-networks for each layer using various binary masks. Inspired by ALBERT [101], ProPETL leverages parameter sharing within the prototype network modules in each layer of the transformer, enhancing parameter efficiency and reducing storage requirements. In ProPETL, binary masks $M \in \{0, 1\}^n$ are introduced in each layer of the transformer, in which n is the number of parameters in a single PEFT module. Each mask corresponds to a specific sub-network of the shared prototype network. By doing so, though each layer shares the parameters of the same prototype network, each layer has a different sub-network to capture meaningful semantic representations. The final objective of the task adaptation for the PLMs can be expressed as follows:

$$\max_{\theta_{pro}, m_1, m_2, \cdots, m_L} \sum_{i=0}^{N} \log P(Y_i | X_i; \theta_{lm}, \theta_{sub}), \tag{17}$$
$$\theta_{sub} = [\theta_{pro} \odot m_1, \theta_{pro} \odot m_2, \cdots, \theta_{pro} \odot m_L].$$

Here, θ_{lm} represents the frozen pretrained parameters of the PLMs, m_i $(i = 1, 2, \cdots, L)$ is binary mask matrix, and θ_{sub} denotes the parameters to be optimized.

IV. Experiments

A. Experimental Settings

1) PLMs and Datasets: We use the encoder-only models RoBERTa-base (125M) and RoBERTa-large (355M) [2] to evaluate on the GLUE benchmark [100], encoder-decoder models T5-base (220M) and T5-large (770M) [4] to evaluate on the WMT16 En-Ro dataset[6], and decoder-only models LLaMA-7B and LLaMA-13B [7] fine-tuned with the Alpaca dataset [102] to evaluate on the MMLU benchmark [103]. All these PLMs with different model types and model scales are based on the encoder, decoder, or encoder-decoder of the **Transformer** architecture. The datasets we use for experiments cover a wide range of tasks, from NLU to MT and NLG. The GLUE benchmark covers a collection of NLU tasks, including single-sentence classification, and sentence-pair classification tasks. WMT16 En-Ro dataset consists of parallel data pairs, where each pair consists of an English sentence and its corresponding translation into Romanian. Alpaca [102] is an instruction dataset containing 52k samples. MMLU Benchmark [103] encompasses a comprehensive range of 57 disciplines spanning science, humanities, social sciences, and more. The level of difficulty of the benchmark ranges from

[6]https://huggingface.co/datasets/wmt16

TABLE III: Fine-tuning RoBERTa-base (RoB_B) and RoBERTa-large (RoB_L) models on the GLUE benchmark. Specifically, we report the Matthews correlation for COLA, accuracy/F1 score for MRPC and QQP, Pearson/Spearman correlation for STS-B, averaged matched accuracy for MNLI, and accuracy for other NLU tasks. Higher values indicate better performance across all metrics. We present the number of trainable parameters (# TPs) of each method, excluding Child-Tuning$_D$ due to its randomness during network pruning. We also bold the maximum values and underline the minimum values.

Model	PEFT Method	#TPs	CoLA	SST2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	Avg.
RoB_B	FT	124.6M	59.07	92.89	88.24/91.58	90.87/90.61	90.81/87.72	86.27	91.07	72.20	84.00/84.00
	Adapter^S	7.41M	63.32	94.31	**90.44/93.18**	91.25/90.94	90.81/86.55	87.33	92.06	73.56	85.39/85.16
	Prompt-tuning	0.61M	49.37	92.09	70.83/81.72	82.44/83.11	82.99/78.35	80.57	80.03	58.12	74.56/75.42
	Prefix-tuning	0.96M	59.31	93.81	87.25/91.03	88.48/88.32	87.75/84.09	85.21	90.77	54.51	80.89/80.88
	(IA)^3	0.66M	59.58	93.92	87.00/90.52	90.30/90.32	87.99/84.10	83.95	90.88	71.12	83.09/83.05
	BitFit	0.69M	61.32	94.72	89.22/92.41	90.34/90.27	88.12/84.11	84.64	91.09	**77.98**	84.68/84.57
	Child-Tuning$_D$	-	60.33	93.58	89.22/92.20	91.14/90.93	**90.98/88.04**	**87.40**	92.20	77.62	85.31/85.29
	LoRA	0.89M	62.09	94.04	87.50/90.68	90.66/90.83	88.83/85.21	86.54	92.02	72.92	84.33/84.29
	AdaLoRA	1.03M	59.82	93.92	87.99/91.33	90.83/90.73	88.58/84.98	86.26	91.43	70.04	83.61/83.56
	MAM Adapter	46.78M	61.42	**94.87**	89.31/92.21	90.74/90.42	88.31/83.20	86.63	90.19	72.62	84.26/83.95
	ProPELT$_{\text{Adapter}}$	1.87M	**66.33**	93.85	87.25/90.82	**91.33/91.04**	89.22/85.79	86.49	**92.56**	75.54	**85.32/85.30**
	ProPELT$_{\text{Prefix}}$	10.49M	61.79	94.30	88.73/91.98	90.30/90.19	88.54/85.05	86.22	91.51	63.31	83.08/83.04
	ProPELT$_{\text{LoRA}}$	1.77M	60.38	94.11	87.42/90.87	90.76/90.55	88.90/85.55	86.84	92.04	67.39	83.48/83.47
RoB_L	FT	355.3M	65.78	95.54	89.22/92.28	91.74/91.76	89.30/86.68	89.42	93.61	81.23	86.98/87.04
	Adapter^S	19.77M	67.03	96.37	89.94/92.54	**92.58/92.42**	**92.19/88.50**	91.00	94.31	85.25	88.58/88.43
	Prompt-tuning	1.07M	61.13	94.61	73.04/81.29	78.51/78.99	80.74/75.16	68.15	89.13	60.29	75.70/76.09
	Prefix-tuning	2.03M	59.01	95.76	88.24/91.37	90.92/91.07	88.88/85.45	89.30	93.32	74.01	84.93/84.91
	(IA)^3	1.22M	61.15	94.61	86.52/90.33	92.22/92.03	89.45/86.25	88.63	94.25	81.23	86.00/86.06
	BitFit	1.32M	**68.01**	96.10	**90.93/93.38**	91.93/91.77	89.48/86.43	89.98	94.47	87.73	88.57/88.47
	Child-Tuning$_D$	-	63.08	95.07	90.69/93.43	92.36/92.18	91.52/88.75	35.45	93.15	86.25	80.95/80.92
	LoRA	1.84M	64.47	**96.67**	87.50/91.19	91.66/91.44	90.15/86.91	90.76	95.00	79.78	87.00/87.03
	AdaLoRA	2.23M	65.85	94.95	89.46/92.34	92.05/91.80	89.60/86.30	90.36	94.62	77.98	86.86/86.78
	MAM Adapter	122.2M	67.39	95.81	90.12/92.77	92.44/92.18	90.87/86.65	90.62	94.31	86.62	88.52/88.29
	ProPELT$_{\text{Adapter}}$	5.40M	65.55	96.27	89.71/92.54	91.92/91.67	90.67/87.74	**91.37**	**95.20**	**88.89**	**88.70/88.65**
	ProPELT$_{\text{Prefix}}$	26.85M	62.24	96.17	90.04/92.92	90.70/90.49	89.30/86.30	90.33	94.73	79.71	86.65/86.61
	ProPELT$_{\text{LoRA}}$	4.19M	61.90	95.93	89.06/92.19	91.66/91.38	90.93/88.05	90.53	94.93	83.57	87.31/87.31

beginner to advanced levels of expertise, testing both world knowledge and problem-solving abilities.

2) PEFT Methods: Eleven representative PEFT methods: sequential adapter (Adapter^S) [9], prompt-tuning [24], prefix-tuning [10], (IA)^3 [30], BitFit [34], Child-Tuning [39], LoRA [11], AdaLoRA [45], QLoRA [49], MAM adapter [16], and ProPELT [65] are chosen. Since the GLUE benchmark consists of a series of NLU tasks, it serves as the preferred evaluation dataset used by most PLMs to validate the effectiveness of PEFT methods. Ten representative PEFT methods other than QLoRA are selected to fine-tune RoBERTa-base/large. For T5-base/large, we use (IA)^3 and LoRA for fine-tuning. As for LLaMA-7B/13B, (IA)^3, LoRA, and QLoRA are used for fine-tuning.

3) Implementation Details: Since "prompt-tuning, prefix-tuning, (IA)^3, LoRA, and AdaLoRA" have been integrated into the PEFT library[7]. Therefore, we directly utilize the PEFT library to invoke these PEFT methods for fine-tuning. For BitFit, Child-tuing$_D$, MAM adapter, QLoRA, and ProPELT, we experiment using their original code. Significantly, we experiment with sequential adapter using code from the MAM adapter. For RoBERTa-base/large, all PEFT methods are fine-tuned using a batch size of 32 and a sequence length of 128, except for (IA)^3 which is fine-tuned using batch size 8. We use the batch size 64 for T5-base and 32 for T5-large. For LLaMA-7B/13B, we use batch size 16 for fine-tuning. All experiments are implemented with A800 GPU.

[7]https://huggingface.co/docs/peft/index

B. Fine-tuning Performance and Parameter Efficiency

1) RoBERTa Base/Large on GLUE: Experimental results of full fine-tuning and 11 representative PEFT methods with RoBERTa-base/large on the GLUE benchmark are presented in Table III, the following findings are observed:

- All PEFT methods reduce the number of trainable parameters, and most PEFT methods achieve performance matching or even better than full fine-tuning on the GLUE benchmark. For RoBERTa-base, the average performance of prompt-tuning, prefix-tuning, IA^3, AdaLoRA, ProPELT$_{\text{prefix}}$ and ProPELT$_{\text{LoRA}}$ on GLUE all underperforms full finetuning, while that of sequential adapter, BitFit, Child-Tuning$_D$, LoRA, MAM adapter, and ProPELT$_{\text{Adapter}}$ outperforms full fine-tuning. For RoBERTa-large, the average performance of prompt-tuning, prefix-tuning, IA3, AdaLoRA, ProPELT$_{\text{prefix}}$ and Child-Tuning$_D$ on GLUE underperforms full fine-tuning, while that of sequential adapter, BitFit, LoRA, MAM adapter, ProPELT$_{\text{Adapter}}$ and ProPELT$_{\text{LoRA}}$ outperforms full fine-tuning.
- ProPELT$_{\text{adapter}}$, a unified fine-tuning method that employs the AdapterFusion as the backbone, uses about 1.50% of the trainable parameters to fine-tune RoBERT-base and RoBERTa-large, but achieves optimal average performance on the GLUE benchmark, outperforming RoBERT-base (FT) by about 1.30% and RoBERT-large (FT) by about 1.65%.
- MAM Adapter, a hybrid fine-tuning method that combines parallel adapters and prefix-tuning, achieves better performance than prefix-tuning, but also consumes a large

TABLE IV: Fine-tuning T5-base and T5-large models on the WMT16 En-Ro dataset and evaluating their performance using BLEU score. The higher BLEU score indicates a better quality of translation output.

Model	PEFT Method	# TPs	BLEU
T5-base	FT	222.9M	27.42
	$(IA)^3$	0.07M	27.58
	LoRA	0.88M	**27.78**
T5-large	FT	737.7M	**28.13**
	$(IA)^3$	0.19M	28.12
	LoRA	2.36M	28.12

amount of trainable parameters.

- Sequential adapter requires more trainable parameters than prompt-tuning, prefix-tuning, $(IA)^3$, BitFit, Child-Tuning$_D$, LoRA, and AdaLoRA, but achieves better performance than them on the GLUE benchmark.
- Prompt-tuning with the virtual marker length set to 20 achieves the smallest trainable parameter, but also the worst performance, with its average performance on the GLUE benchmark being about 10% lower than full fine-tuning.
- Child-Tuning$_D$ performs well when fine-tuning RoBERT-base on the GLUE benchmark and obtains better performance than full fine-tuning, but performs poorly when fine-tuning RoBERT-large on the MNLI dataset, which we guess it caused by the learning rate.

2) T5 Base/Large on WMT16 En-Ro Dataset: As depicted in Table IV, both $(IA)^3$ and LoRA significantly reduce the number of trainable parameters compared to full fine-tuning, while maintaining comparable performance. Specifically, $(IA)^3$ employs only 0.03% of trainable parameters and achieves a BLEU score [104] 0.16 higher than full fine-tuning for T5-base and 0.01 lower for T5-large. LoRA achieves a BLEU score 0.36 higher than full fine-tuning on T5-base using only 0.39% of trainable parameters, and 0.01 lower than full fine-tuning on T5-large using only 0.32% of trainable parameters.

3) LLaMA on MMLU: We first tracked the 5-shot MMLU dev accuracy of LLaMA-7B-Alpaca and LLaMA-13B-Alpaca with full fine-tuning and PEFT approaches LoRA, QLoRA, and $(IA)^3$, following the work in [49]. As depicted in Fig. 4, there are significant performance fluctuations in the 5-shot MMLU dev accuracy throughout model training, particularly in LoRA and QLoRA. Moreover, we discovered that full fine-tuning performance of LLaMA-7B-Alpaca on the MMLU benchmark is extremely sensitive to the learning rate, as shown in Table V. Subsequently, we select the checkpoint with the best performance on the dev set and perform 5-shot accuracy experiments on the test set of the MMLU benchmark.

As illustrated in Table VI, full fine-tuning of both LLaMA-7B and LLaMA-13B produces better 5-shot MMLU test accuracy compared to other PEFT methods. $(IA)^3$, LoRA, and QLoRA methods all greatly reduce the number of trainable parameters with $(IA)^3$ performs best. Although $(IA)^3$ only consumes 0.02% of full fine-tuning parameters, it performs 2-4% lower than full fine-tuning. LoRA and QLoRA require about 2% of full fine-tuning parameters, achieving 5-shot MMLU accuracy that is about 2% lower than full fine-tuning. In particular, QLoRA only uses half the number of trainable parameters of LoRA but achieves comparable performance. This reduction of parameters in QLoRA can be attributed to the incorporation of 4-bit NormalFloat quantization.

TABLE V: Full fine-tuning performance of LLaMA-7B-Alpaca on the test set of MMLU benchmark with different learning rates.

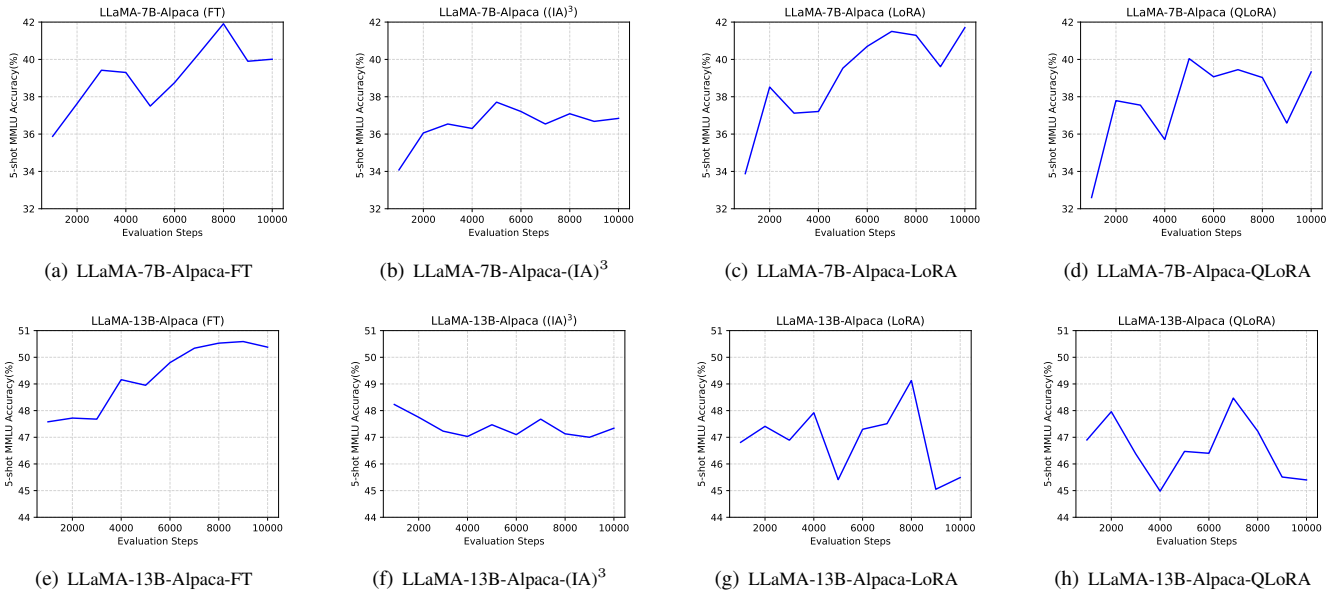
Learning rate	5-shot MMLU Accuracy
2e-4	25.71
5e-5	26.65
1e-6	**41.79**

C. Memory Efficiency

It has been demonstrated that PEFT methods effectively reduce the number of trainable parameters. However, it remains unclear whether they can also reduce GPU memory usage. To assess the impact of PEFT methods on GPU memory, we compare the GPU memory cost of full fine-tuning and PEFT methods across various models and benchmarks. The specific experimental settings can be seen in the section of implementation details. As presented in Table VII, the memory usage of full fine-tuning in RoBERTa, T5, and LLaMA is positively related to total model parameters. RoBERTa, specifically RoBERTa-base, consumes less memory, requiring only 5.38GB. In contrast, LLaMA demands significantly larger memory, notably LLaMA-13B, necessitating approximately 290GB for full fine-tuning.

In RoBERTa-base/large, prompt-tuning, prefix-tuning, $(IA)^3$, LoRA and AdaLoRA (implemented using the PEFT library), and BitFit significantly reduce the GPU memory footprint compared to full fine-tuning. Surprisingly, sequential adapter, MAM adapter, Child-Tuning$_D$, and ProPELT all use more memory than full fine-tuning. Both sequential adapter and MAM adapter exhibit higher memory consumption, around three times that of full fine-tuning, with the MAM adapter consuming even more memory. For T5-base/large models, $(IA)^3$ and LoRA all demonstrate effective memory reduction during fine-tuning, with LoRA outperforming $(IA)^3$. Notably, $(IA)^3$ consumes less GPU memory than LoRA in RoBERTa-base/large, which is caused by the smaller batch size during $(IA)^3$ fine-tuning ($(IA)^3$ sets the batch size to 8).

Likewise, $(IA)^3$, LoRA, and QLoRA all significantly reduce the GPU footprint compared to full finetuning in LLaMA-7B/13B. In addition, we discovered that the PEFT method is more effective in reducing memory usage when the number of model parameters is larger. For example, in LLaMA-7B-Alpaca, compared with full fine-tuning, IA^3, LoRA, and QLoRA reduce memory usage by 24.08%, 26.30%, and 66.66%, respectively; while in LLaMA-13B-Alpaca, compared with full fine-tuning, IA^3, LoRA, and QLoRA reduce memory usage by 33.55%, 39.46% and 76.86% of memory usage. Notably, QLoRA dramatically reduces GPU memory consumption, with QLoRA fine-tuning the LLaMA-7B requiring



(a) LLaMA-7B-Alpaca-FT (b) LLaMA-7B-Alpaca-(IA)^3 (c) LLaMA-7B-Alpaca-LoRA (d) LLaMA-7B-Alpaca-QLoRA

(e) LLaMA-13B-Alpaca-FT (f) LLaMA-13B-Alpaca-(IA)^3 (g) LLaMA-13B-Alpaca-LoRA (h) LLaMA-13B-Alpaca-QLoRA

Fig. 4: The 5-shot accuracy fluctuates on the MMLU dev set with the increase in evaluation steps when fine-tuning LLaMA-7B-Alpaca and LLaMA-7B-Alpaca using the IA^3, LoRA, and QLoRA methods.

TABLE VI: Comparison of the average 5-shot MMLU test accuracy of LLaMA-7B and LLaMA-13B models fine-tuned with Alpaca. The higher the MMLU accuracy, the better. We also report total model parameters (# APs) and the ratio of trainable parameters.

Model	PEFT Method	# TPs	# APs	% Params	5-shot MMLU Accuracy
LLaMA-7B-Alpaca	FT	6738.4M	6738.4M	100	**41.79**
	(IA)^3	1.58M	6740.0M	0.02	37.88
	LoRA	159.9M	6898.3M	2.32	40.67
	QLoRA	79.9M	3660.3M	2.18	39.96
LLaMA-13B-Alpaca	FT	13015.9M	13015.9M	100	**49.60**
	(IA)^3	2.48M	13018.3M	0.02	47.42
	LoRA	250.3M	13266.2M	1.88	47.49
	QLoRA	125.2M	6922.3M	1.81	47.29

only 1/3 of the memory required for full fine-tuning, and fine-tuning the LLaMA-13B requiring less than 1/4 of the memory required for full fine-tuning. This advancement opens up the possibility of fine-tuning LLMs for various downstream tasks in computational resource-constrained scenarios.

V. Applications

A. Multi-task Learning

Multi-task learning is a method that involves training a model on multiple related tasks and exploiting the information shared and transferred between them to improve the performance of each task. PEFT methods such as adapters, prompt-tuning, and LoRA utilize additional modules that can be plugged into PLMs and thus can be used for task-specific fine-tuning to improve generalization of multi-task learning. For instance, studies from [19], [21], [22], [75] leverage task-specific adapters to learn information stored in multiple tasks to achieve more robust transfer learning on new tasks. Several works [26], [27], [28] employ prompt-tuning for multi-task learning. They either utilize pretrained soft prompts from multiple source tasks to initialize the soft prompt of the target task, based on the similarity between the source and target tasks, or employ multi-task data to learn a single shared prompt and transfer it to the target task. Similar to the adapter, a composition of multiple task-specific LoRA modules is also leveraged to transfer knowledge to new tasks [55], [56]. L-LoRA [57] enhances the fusion capabilities of multi-task learning by preventing negative inference between task-specific representations. Additionally, [93] utilizes arithmetic operators, such as the addition and negation operators, to merge parameters of various PEFT methods trained on different tasks for multi-task learning.

B. Cross-Lingual Transfer

Cross-lingual transfer involves transferring knowledge or models from one language to another. Numerous works have employed PEFT methods, such as adapters, for cross-lingual transfer due to their unique modular design. Bapna and Firat [105] utilize sequential adapter [9] to fine-tune and restore the performance of a multilingual neural machine translation

TABLE VII: The peak GPU memory usage when fine-tuning RoBERT-base, RoBERTa-large, T5-base, T5-large, LLaMA-7B, and LLaMA-13B model using full fine-tuning and various PEFT methods.

Model & Method	**Memory (GB)**	**Model & Method**	**Memory (GB)**
RoBERTa-base (FT)	5.38	RoBERTa-large (FT)	11.96
RoBERTa-base (Adapter^S)	15.29	RoBERTa-large (Adapter^S)	37.17
RoBERTa-base (Prompt-tuning)	3.84	RoBERTa-large (Prompt-tuning)	7.98
RoBERTa-base (Prefix-tuning)	3.56	RoBERTa-large (Prefix-tuning)	7.58
RoBERTa-base ((IA)^3)	2.62	RoBERTa-large ((IA)^3)	4.83
RoBERTa-base (BitFit)	3.27	RoBERTa-large (BitFit)	7.50
RoBERTa-base (Child-Tuning$_D$)	6.02	RoBERTa-large (Child-Tuning$_D$)	13.67
RoBERTa-base (LoRA)	3.59	RoBERTa-large (LoRA)	7.50
RoBERTa-base (AdaLoRA)	3.57	RoBERTa-large (AdaLoRA)	7.43
RoBERTa-base (MAM Adapter)	**15.35**	RoBERTa-large (MAM Adapter)	**37.82**
RoBERTa-base (ProPELT$_{\text{Adapter}}$)	8.63	RoBERTa-large (ProPELT$_{\text{Adapter}}$)	19.82
RoBERTa-base (ProPELT$_{\text{Prefix}}$)	9.47	RoBERTa-large (ProPELT$_{\text{Prefix}}$)	22.85
RoBERTa-base (ProPELT$_{\text{LoRA}}$)	8.25	RoBERTa-large (ProPELT$_{\text{LoRA}}$)	19.52
T5-base (FT)	**25.17**	T5-large (FT)	**30.17**
T5-base ((IA)^3)	21.36	T5-large ((IA)^3)	25.71
T5-base (LoRA)	19.43	T5-large (LoRA)	23.77
LLaMA-7B-Alpaca (FT)	**169.36**	LLaMA-13B-Alpaca (FT)	**287.79**
LLaMA-7B-Alpaca ((IA)^3)	128.57	LLaMA-13B-Alpaca ((IA)^3)	191.24
LLaMA-7B-Alpaca (LoRA)	124.82	LLaMA-13B-Alpaca (LoRA)	174.24
LLaMA-7B-Alpaca (QLoRA)	56.46	LLaMA-13B-Alpaca (QLoRA)	66.60

model on high-resource languages. Artetxe et al. [106] employ sequential adapter [9] to transfer a pretrained monolingual model to an unseen language. MAD-X [75], [107] uses language-specific, task-specific, and invertible adapter to learn language-specific and task-specific transformations, as well as address vocabulary mismatches between multilingual and target languages in a modular manner, enabling the adaptation of pretrained multilingual models to target languages. MAD-G [108] generates language adapters from language representations based on typological features, allowing the sharing of linguistic knowledge across languages for cross-lingual transfer. LT-SFT [38] employs sparse fine-tuning to train the model on the source language and learn task-specific sparse difference vectors for cross-lingual transfer. While BAD-X [109] trains a bilingual language-pair adapter on both the source and target languages for zero-shot cross-lingual transfer.

C. Backdoor Attacks and Defense

Backdoor attacks pose a significant security threat, where a small portion of training samples are contaminated with malicious backdoor triggers. When trained on such poisoned datasets, the model behaves normally on benign samples but predicts attacker-selected labels on samples containing the predefined triggers. The susceptibility of PLMs to backdoor attacks poses a substantial risk to real-world applications [110]. Building on the vulnerability of pretrained weights, Gu et al. [111] employ the PEFT methods to construct backdoor attacks, in which backdoor attacks are directly injected into PEFT modules. However, Zhu et al. [112] discover that PEFT can serve as a backdoor defense solution by reducing the model capacity via optimizing only a small number of parameters. The findings from [113] also confirm that PEFT can slightly weaken the backdoor attacks and design a novel trojan attack for the PEFT paradigm.

VI. Further Directions

A. Lightweight Hybrid PEFT Methods

There exist many approaches [16], [35], [58], [59], [60], [61], [62] to combine multiple PEFT methods, aiming to leverage the distinctive advantages of each PEFT method and achieve enhanced performance. Nevertheless, the exploration has been limited to PEFT methods such as adapter, LoRA, prefix-tuning, and BitFit, leaving room for further exploitation by incorporating additional combinations of PEFT methods. Moreover, while drawing inspiration from the NAS algorithm, several PEFT methods [60], [61] have been investigated using diverse optimization techniques to explore optimal neural network architectures for configuring these PEFT methods. There remains potential for continued exploration in utilizing other optimization methods to automatically search for neural network architectures and configure specific combinations of PEFT modules at specific layers. Additionally, utilizing multiple PEFT methods typically results in increased parameter and memory usage, although it enhances performance. Hence, an intriguing research direction involves investigating how to leverage multiple PEFT methods to improve performance while minimizing the number of trainable parameters.

B. LoRA-derived PEFT Methods

Recently, a multitude of LoRA-based PEFT methods have emerged, as demonstrated in Fig. 1. These methods further enhance LoRA by incorporating adaptive rank adjustment, unstructured pruning techniques, weight quantization, and multi-task integration. This encourages future research to develop

more LoRA-derived PEFT approaches build upon LoRA. Particular emphasis should be placed on pruning technology and weight quantification. The application of pruning techniques can be extended not only to AdaLoRA [45] for rank adjustment but also to LoRAPrune [48] for pruning both pretrained and LoRA weights. Notably, pruning and weight quantization techniques effectively reduce the number of trainable parameters, compress model size, optimize storage and computational requirements of PLMs (especially LLMs), and enhance their utility and scalability across downstream tasks. These techniques can be further explored in conjunction with LoRA to unlock synergistic benefits.

C. Developing PEFT Library

Numerous PEFT methods have emerged, but employing them is not a straightforward endeavor. To address this challenge, the PEFT library[8] and AdapterHub[9] have been developed. These libraries integrate commonly used PEFT methods such as prefix-tuning, LoRA, and AdaLoRA. With just a few lines of code, users can directly invoke these PEFT methods, simplifying their usage. Moreover, both the PEFT and AdapterHub libraries offer a range of examples illustrating how to apply these PEFT methods to various PLMs and LLMs for fine-tuning downstream tasks. However, not all PEFT methods are currently integrated into these two libraries. Future efforts can be directed towards expanding the integration of additional methods, further boosting the application development of PEFT methods.

D. Explainability of PEFT Methods

Though numerous PEFT methods have been proposed, there is a lack of comprehensive studies exploring the reasons behind their ability to achieve comparable performance and reduce trainable parameters. Work from [41] unifies PEFT methods under the concept of sparse fine-tuned models and provides a theoretical analysis demonstrating that sparsity can serve as a regularization technique for the original model, effectively controlling the upper bound of stability. While [114] explores and analyzes the express power of LoRA for fully connected neural networks and transformer networks, showing the conditions under which there exist effective low-rank adapters for a given task. These studies shed light on the working mechanism and effectiveness of certain PEFT methods, but still lack generalization. Future research endeavors could focus on advancing theoretical studies to unravel the underlying working mechanisms of PEFT methods.

E. Exploring PEFT Methods in Computer Vision and Multimodal Learning

Though PEFT methods have been extensively studied in NLP, their application in computer vision and multimodal learning also shows great potential for further exploration. The sequential adapter in NLP, initially inspired by multi-domain image classification [77], [115], has paved the way for rapid advancements in PEFT methods for PLMs. Moreover, researchers have increasingly delved into various PEFT techniques for computer vision [116], [117], as well as language-image and image-audio multimodal learning [118], [119], building upon PEFT methods in NLP [9], [11], [58]. However, there is still significant room for further exploration and exploitation in these domains. In particular, PEFT methods hold the potential to facilitate cross-modality transfer in multimodal learning. By fine-tuning pretrained models using PEFT techniques, knowledge acquired from one modality can be effectively transferred to another, resulting in improved performance in multimodal tasks. Consequently, the application of PEFT methods in computer vision and multimodal learning holds tremendous promise as a future research direction.

[8]https://github.com/huggingface/peft/tree/main

[9]https://adapterhub.ml/

VII. Conclusions

This paper presents a comprehensive and structured study of PEFT methods for PLMs. By classifying the PEFT methods in NLP, we identify the main techniques and challenges associated with them. We employ several representative PEFT methods to fine-tune encoder-based RoBERTa, encoder-decoder-based T5, and decoder-based LLaMA on various downstream tasks. Experimental results reveal that most PEFT methods significantly improve parameter efficiency and achieve comparable or even better performance compared to full fine-tuning. Additionally, most PEFT methods lower the memory footprint, with QLoRA drastically reducing the computational memory requirement, and alleviating the memory challenge when fine-tuning LLMs. Furthermore, we introduce common applications of PEFT methods and outline future research directions. As the development of LLMs continues, there is a clear need to develop PEFT methods that can effectively reduce computational resource demands and memory usage during fine-tuning. This survey aims to provide a bird's-eye view of PEFT methods for PLMs and inspiring further research in this area.

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Hum. Lang. Technol.*, 2019, pp. 4171–4186.

[2] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," in *Proc. Int. Conf. Learn. Representations*, 2020.

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[4] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, no. 1, pp. 5485–5551, 2020.

[5] S. Zhang, M. Diab, and L. Zettlemoyer, "Democratizing access to large-scale language models with opt-175b," *Meta AI*, 2022.

[6] T. L. Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilić, D. Hesslow, R. Castagné, A. S. Luccioni, F. Yvon, M. Gallé *et al.*, "Bloom: A 176b-parameter open-access multilingual language model," *arXiv preprint arXiv:2211.05100*, 2022.

[7] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[8] E. Almazrouei, H. Alobeidli, A. Alshamsi, A. Cappelli, R. Cojocaru, M. Alhammadi, M. Daniele, D. Heslow, J. Launay, Q. Malartic *et al.*, "The falcon series of language models: Towards open frontier models," *Hugging Face repository*, 2023.

[9] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for nlp," in *Proc. Int. Conf. Mach. Learn.* PMLR, 2019, pp. 2790–2799.

[10] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," in *Proc. Annu. Meeting Assoc. Comput. Linguistics, Int. Joint Conf. Natural Lang. Process.*, 2021, pp. 4582–4597.

[11] E. J. Hu, yelong shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *Proc. Int. Conf. Learn. Representations*, 2022.

[12] N. Ding, Y. Qin, G. Yang, F. Wei, Z. Yang, Y. Su, S. Hu, Y. Chen, C.-M. Chan, W. Chen *et al.*, "Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models," *arXiv preprint arXiv:2203.06904*, 2022.

[13] V. Lialin, V. Deshpande, and A. Rumshisky, "Scaling down to scale up: A guide to parameter-efficient fine-tuning," *arXiv preprint arXiv:2303.15647*, 2023.

[14] Z. Lin, A. Madotto, and P. Fung, "Exploring versatile generative language model via parameter-efficient transfer learning," in *Proc. Findings Conf. Empir. Methods Natural Lang. Process.*, 2020, pp. 441–459.

[15] T. Lei, J. Bai, S. Brahma, J. Ainslie, K. Lee, Y. Zhou, N. Du, V. Y. Zhao, Y. Wu, B. Li *et al.*, "Conditional adapters: Parameter-efficient transfer learning with fast inference," *arXiv preprint arXiv:2304.04947*, 2023.

[16] J. He, C. Zhou, X. Ma, T. Berg-Kirkpatrick, and G. Neubig, "Towards a unified view of parameter-efficient transfer learning," in *Proc. Int. Conf. Learn. Representations*, 2022.

[17] A. Rücklé, G. Geigle, M. Glockner, T. Beck, J. Pfeiffer, N. Reimers, and I. Gurevych, "AdapterDrop: On the efficiency of adapters in transformers," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2021, pp. 7930–7946.

[18] H. Zhao, H. Tan, and H. Mei, "Tiny-attention adapter: Contexts are more important than the number of parameters," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2022, pp. 6626–6638.

[19] J. Pfeiffer, A. Kamath, A. Rücklé, K. Cho, and I. Gurevych, "AdapterFusion: Non-destructive task composition for transfer learning," in *Proc. Conf. Eur. Chapter Assoc. Comput. Linguistics*, 2021, pp. 487–503.

[20] S. He, R.-Z. Fan, L. Ding, L. Shen, T. Zhou, and D. Tao, "Mera: Merging pretrained adapters for few-shot learning," *arXiv preprint arXiv:2308.15982*, 2023.

[21] R. Karimi Mahabadi, S. Ruder, M. Dehghani, and J. Henderson, "Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks," in *Proc. Annu. Meeting Assoc. Comput. Linguistics, Int. Joint Conf. Natural Lang. Process.*, 2021, pp. 565–576.

[22] A. Chronopoulou, M. Peters, A. Fraser, and J. Dodge, "AdapterSoup: Weight averaging to improve generalization of pretrained language models," in *Proc. Findings Assoc. Comput. Linguistics*, 2023, pp. 2054–2063.

[23] K. Hambardzumyan, H. Khachatrian, and J. May, "WARP: Word-level Adversarial ReProgramming," in *Proc. Annu. Meeting Assoc. Comput. Linguistics, Int. Joint Conf. Natural Lang. Process.*, 2021, pp. 4921–4933.

[24] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2021, pp. 3045–3059.

[25] X. Liu, Y. Zheng, Z. Du, M. Ding, Y. Qian, Z. Yang, and J. Tang, "Gpt understands, too," *arXiv preprint arXiv:2103.10385*, 2021.

[26] T. Vu, B. Lester, N. Constant, R. Al-Rfou', and D. Cer, "SPoT: Better frozen model adaptation through soft prompt transfer," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2022, pp. 5039–5059.

[27] A. Asai, M. Salehi, M. Peters, and H. Hajishirzi, "ATTEMPT: Parameter-efficient multi-task tuning via attentional mixtures of soft prompts," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2022, pp. 6655–6672.

[28] Z. Wang, R. Panda, L. Karlinsky, R. Feris, H. Sun, and Y. Kim, "Multitask prompt tuning enables parameter-efficient transfer learning," in *Proc. Int. Conf. Learn. Representations*, 2023.

[29] Y.-L. Sung, J. Cho, and M. Bansal, "LST: Ladder side-tuning for parameter and memory efficient transfer learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022.

[30] H. Liu, D. Tam, M. Mohammed, J. Mohta, T. Huang, M. Bansal, and C. Raffel, "Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022.

[31] X. Yang, J. Y. Huang, W. Zhou, and M. Chen, "Parameter-efficient tuning with special token adaptation," in *Proc. Conf. Eur. Chapter Assoc. Comput. Linguistics*, 2023, pp. 865–872.

[32] J. Cao, C. Satya Prakash, and W. Hamza, "Attention fusion: a light yet efficient late fusion mechanism for task adaptation in NLU," in *Proc. Findings Assoc. Comput. Linguistics*, 2022, pp. 857–866.

[33] Y. Chen, Q. Fu, G. Fan, L. Du, J.-G. Lou, S. Han, D. Zhang, Z. Li, and Y. Xiao, "Hadamard adapter: An extreme parameter-efficient adapter tuning method for pre-trained language models," in *Proc. 32nd ACM Int. Conf. Inf. Knowl. Manage.*, 2023, pp. 276–285.

[34] E. Ben Zaken, Y. Goldberg, and S. Ravfogel, "BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2022, pp. 1–9.

[35] N. Lawton, A. Kumar, G. Thattai, A. Galstyan, and G. Ver Steeg, "Neural architecture search for parameter-efficient fine-tuning of large pre-trained language models," in *Proc. Findings Assoc. Comput. Linguistics*, 2023, pp. 8506–8515.

[36] M. Zhao, T. Lin, F. Mi, M. Jaggi, and H. Schütze, "Masking as an efficient alternative to finetuning for pretrained language models," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2020, pp. 2226–2241.

[37] Y.-L. Sung, V. Nair, and C. Raffel, "Training neural networks with fixed sparse masks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021.

[38] A. Ansell, E. Ponti, A. Korhonen, and I. Vulić, "Composable sparse fine-tuning for cross-lingual transfer," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2022, pp. 1778–1796.

[39] R. Xu, F. Luo, Z. Zhang, C. Tan, B. Chang, S. Huang, and F. Huang, "Raise a child in large language model: Towards effective and generalizable fine-tuning," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2021, pp. 9514–9528.

[40] D. Guo, A. Rush, and Y. Kim, "Parameter-efficient transfer learning with diff pruning," in *Proc. Annu. Meeting Assoc. Comput. Linguistics, Int. Joint Conf. Natural Lang. Process.*, 2021, pp. 4884–4896.

[41] Z. Fu, H. Yang, A. M.-C. So, W. Lam, L. Bing, and N. Collier, "On the effectiveness of parameter-efficient fine-tuning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 37, no. 11, 2023, pp. 12 799–12 807.

[42] A. Aghajanyan, S. Gupta, and L. Zettlemoyer, "Intrinsic dimensionality explains the effectiveness of language model fine-tuning," in *Proc. Annu. Meeting Assoc. Comput. Linguistics, Int. Joint Conf. Natural Lang. Process.*, 2021, pp. 7319–7328.

[43] A. Edalati, M. Tahaei, I. Kobyzev, V. P. Nia, J. J. Clark, and M. Rezagholizadeh, "Krona: Parameter efficient tuning with kronecker adapter," *arXiv preprint arXiv:2212.10650*, 2022.

[44] M. Valipour, M. Rezagholizadeh, I. Kobyzev, and A. Ghodsi, "DyLoRA: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation," in *Proc. Conf. Eur. Chapter Assoc. Comput. Linguistics*, 2023, pp. 3274–3287.

[45] Q. Zhang, M. Chen, A. Bukharin, P. He, Y. Cheng, W. Chen, and T. Zhao, "Adaptive budget allocation for parameter-efficient fine-tuning," in *Proc. Int. Conf. Learn. Representations*, 2023.

[46] F. Zhang, L. Li, J. Chen, Z. Jiang, B. Wang, and Y. Qian, "Increlora: Incremental parameter allocation method for parameter-efficient fine-tuning," *arXiv preprint arXiv:2308.12043*, 2023.

[47] B. Zi, X. Qi, L. Wang, J. Wang, K.-F. Wong, and L. Zhang, "Delta-lora: Fine-tuning high-rank parameters with the delta of low-rank matrices," *arXiv preprint arXiv:2309.02411*, 2023.

[48] M. Zhang, C. Shen, Z. Yang, L. Ou, X. Yu, B. Zhuang *et al.*, "Pruning meets low-rank parameter-efficient fine-tuning," *arXiv preprint arXiv:2305.18403*, 2023.

[49] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *arXiv preprint arXiv:2305.14314*, 2023.

[50] Y. Xu, L. Xie, X. Gu, X. Chen, H. Chang, H. Zhang, Z. Chen, X. Zhang, and Q. Tian, "Qa-lora: Quantization-aware low-rank adaptation of large language models," *arXiv preprint arXiv:2309.14717*, 2023.

[51] Y. Li, Y. Yu, C. Liang, P. He, N. Karampatziakis, W. Chen, and T. Zhao, "Loftq: Lora-fine-tuning-aware quantization for large language models," *arXiv preprint arXiv:2310.08659*, 2023.

[52] Y. Chen, D. Hazarika, M. Namazifar, Y. Liu, D. Jin, and D. Hakkani-Tur, "Empowering parameter-efficient transfer learning by recognizing the kernel structure in self-attention," in *Proc. Findings Assoc. Comput. Linguistics*, 2022, pp. 1375–1388.

[53] A. X. Yang, M. Robeyns, X. Wang, and L. Aitchison, "Bayesian low-rank adaptation for large language models," *arXiv preprint arXiv:2308.13111*, 2023.

[54] L. Zhang, L. Zhang, S. Shi, X. Chu, and B. Li, "Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning," *arXiv preprint arXiv:2308.03303*, 2023.

[55] C. Huang, Q. Liu, B. Y. Lin, T. Pang, C. Du, and M. Lin, "Lorahub: Efficient cross-task generalization via dynamic lora composition," *arXiv preprint arXiv:2307.13269*, 2023.

[56] Q. Liu, X. Wu, X. Zhao, Y. Zhu, D. Xu, F. Tian, and Y. Zheng, "Moelora: An moe-based parameter efficient fine-tuning method for multi-task medical applications," *arXiv preprint arXiv:2310.18339*, 2023.

[57] A. Tang, L. Shen, Y. Luo, Y. Zhan, H. Hu, B. Du, Y. Chen, and D. Tao, "Parameter efficient multi-task model fusion with partial linearization," *arXiv preprint arXiv:2310.04742*, 2023.

[58] R. Karimi Mahabadi, J. Henderson, and S. Ruder, "Compacter: Efficient low-rank hypercomplex adapter layers," *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, pp. 1022–1035, 2021.

[59] Y. Mao, L. Mathias, R. Hou, A. Almahairi, H. Ma, J. Han, S. Yih, and M. Khabsa, "UniPELT: A unified framework for parameter-efficient language model tuning," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2022, pp. 6253–6264.

[60] H. Zhou, X. Wan, I. Vulić, and A. Korhonen, "Autopeft: Automatic configuration search for parameter-efficient fine-tuning," *arXiv preprint arXiv:2301.12132*, 2023.

[61] S. Hu, Z. Zhang, N. Ding, Y. Wang, Y. Wang, Z. Liu, and M. Sun, "Sparse structure search for delta tuning," *Proc. Adv. Neural Inf. Process. Syst.*, vol. 35, pp. 9853–9865, 2022.

[62] J. Chen, A. Zhang, X. Shi, M. Li, A. Smola, and D. Yang, "Parameter-efficient fine-tuning design spaces," in *Proc. Int. Conf. Learn. Representations*, 2023.

[63] Y. Wang, S. Agarwal, S. Mukherjee, X. Liu, J. Gao, A. H. Awadallah, and J. Gao, "AdaMix: Mixture-of-adaptations for parameter-efficient model tuning," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2022, pp. 5744–5760.

[64] S. He, L. Ding, D. Dong, J. Zhang, and D. Tao, "SparseAdapter: An easy approach for improving the parameter-efficiency of adapters," in *Proc. Findings Conf. Empir. Methods Natural Lang. Process.*, 2022, pp. 2184–2190.

[65] G. Zeng, P. Zhang, and W. Lu, "One network, many masks: Towards more parameter-efficient transfer learning," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2023, pp. 7564–7580.

[66] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.

[67] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[68] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[69] L. Xu and W. Wang, "Improving aspect-based sentiment analysis with contrastive learning," *Natural Language Processing Journal*, vol. 3, p. 100009, 2023.

[70] Y. Xie, W. Yang, L. Tan, K. Xiong, N. J. Yuan, B. Huai, M. Li, and J. Lin, "Distant supervision for multi-stage fine-tuning in retrieval-based question answering," in *Proceedings of The Web Conference*, 2020, pp. 2934–2940.

[71] R. Dabre, A. Fujita, and C. Chu, "Exploiting multilingualism through multistage fine-tuning for low-resource neural machine translation," in *Proc. Conf. Empir. Methods Natural Lang. Process., Int. Joint Conf. Natural Lang. Process.*, 2019, pp. 1410–1416.

[72] M. T. Hosseini, A. Ghaffari, M. S. Tahaei, M. Rezagholizadeh, M. Asgharian, and V. P. Nia, "Towards fine-tuning pre-trained language models with integer forward and backward propagation," in *Proc. Findings Assoc. Comput. Linguistics*, 2023, pp. 1867–1876.

[73] S.-i. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, no. 4-5, pp. 185–196, 1993.

[74] L. Xu, H. Xie, Z. Li, F. L. Wang, W. Wang, and Q. Li, "Contrastive learning models for sentence representations," *ACM Trans. Intel. Syst. Tec.*, vol. 14, no. 4, pp. 1–34, 2023.

[75] J. Pfeiffer, I. Vulić, I. Gurevych, and S. Ruder, "MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2020, pp. 7654–7673.

[76] Y. Zhu, J. Feng, C. Zhao, M. Wang, and L. Li, "Counter-interference adapter for multilingual machine translation," *arXiv preprint arXiv:2104.08154*, 2021.

[77] S.-A. Rebuffi, H. Bilen, and A. Vedaldi, "Learning multiple visual domains with residual adapters," *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.

[78] J. Solomon, F. De Goes, G. Peyré, M. Cuturi, A. Butscher, A. Nguyen, T. Du, and L. Guibas, "Convolutional wasserstein distances: Efficient optimal transportation on geometric domains," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 1–11, 2015.

[79] S. P. Singh and M. Jaggi, "Model fusion via optimal transport," *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 22 045–22 055, 2020.

[80] D. Ha, A. M. Dai, and Q. V. Le, "Hypernetworks," in *Proc. Int. Conf. Learn. Representations*, 2017.

[81] R. Aharoni and Y. Goldberg, "Unsupervised domain clusters in pretrained language models," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2020, pp. 7747–7763.

[82] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *Proc. Int. Conf. Learn. Representations*, 2017.

[83] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, "What does BERT look at? an analysis of BERT's attention," in *Proc. of 2019 ACL Workshop BlackboxNLP*, 2019, pp. 276–286.

[84] O. Kovaleva, A. Romanov, A. Rogers, and A. Rumshisky, "Revealing the dark secrets of BERT," in *Proc. Conf. Empir. Methods Natural Lang. Process., Int. Joint Conf. Natural Lang. Process.*, 2019, pp. 4365–4374.

[85] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, no. 1, pp. 1997–2017, 2019.

[86] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proc. Int. Conf. Learn. Representations*, 2019.

[87] Q. Le, T. Sarlós, and A. Smola, "Fastfood-computing hilbert space expansions in loglinear time," in *Proc. Int. Conf. Mach. Learn.* PMLR, 2013, pp. 244–252.

[88] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 11 264–11 272.

[89] V. Sanh, T. Wolf, and A. Rush, "Movement pruning: Adaptive sparsity by fine-tuning," *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 20 378–20 389, 2020.

[90] D. J. MacKay, "A practical bayesian framework for backpropagation networks," *Neural Comput.*, vol. 4, no. 3, pp. 448–472, 1992.

[91] J. Liu, A. Moreau, M. Preuss, J. Rapin, B. Roziere, F. Teytaud, and O. Teytaud, "Versatile black-box optimization," in *Proc. of the 2020 Genet. and Evolut. Comput. Conf.*, 2020, pp. 620–628.

[92] G. Ilharco, M. T. Ribeiro, M. Wortsman, L. Schmidt, H. Hajishirzi, and A. Farhadi, "Editing models with task arithmetic," in *Proc. Int. Conf. Learn. Representations*, 2023.

[93] J. Zhang, S. Chen, J. Liu, and J. He, "Composing parameter-efficient modules with arithmetic operations," *arXiv preprint arXiv:2306.14870*, 2023.

[94] P. Yadav, D. Tam, L. Choshen, C. Raffel, and M. Bansal, "Resolving interference when merging models," *arXiv preprint arXiv:2306.01708*, 2023.

[95] A. Zhang, Y. Tay, S. Zhang, A. Chan, A. T. Luu, S. Hui, and J. Fu, "Beyond fully-connected layers with quaternions: Parameterization of hypercomplex multiplications with $1/n$ parameters," in *Proc. Int. Conf. Learn. Representations*, 2021.

[96] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017.

[97] J. Frankle, G. K. Dziugaite, D. Roy, and M. Carbin, "Pruning neural networks at initialization: Why are we missing the mark?" in *Proc. Int. Conf. Learn. Representations*, 2021.

[98] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nature communications*, vol. 9, no. 1, p. 2383, 2018.

[99] N. Lee, T. Ajanthan, and P. Torr, "SNIP: Single-shot network pruning based on connection sensitivity," in *Proc. Int. Conf. Learn. Representations*, 2019.

[100] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *Proc. of 2018 EMNLP Workshop BlackboxNLP*, 2018, pp. 353–355.

[101] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," in *Proc. Int. Conf. Learn. Representations*, 2020.

[102] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, "Stanford alpaca: An instruction-following llama model," 2023.

[103] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," in *Proc. Int. Conf. Learn. Representations*, 2021.

[104] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2002, pp. 311–318.

[105] A. Bapna and O. Firat, "Simple, scalable adaptation for neural machine translation," in *Proc. Conf. Empir. Methods Natural Lang. Process., Int. Joint Conf. Natural Lang. Process.*, 2019, pp. 1538–1548.

[106] M. Artetxe, S. Ruder, and D. Yogatama, "On the cross-lingual transferability of monolingual representations," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2020, pp. 4623–4637.

[107] J. Pfeiffer, I. Vulić, I. Gurevych, and S. Ruder, "UNKs everywhere: Adapting multilingual language models to new scripts," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2021, pp. 10 186–10 203.

[108] A. Ansell, E. M. Ponti, J. Pfeiffer, S. Ruder, G. Glavaš, I. Vulić, and A. Korhonen, "MAD-G: Multilingual adapter generation for efficient cross-lingual transfer," in *Proc. Findings Conf. Empir. Methods Natural Lang. Process.*, 2021, pp. 4762–4781.

[109] M. Parović, G. Glavaš, I. Vulić, and A. Korhonen, "BAD-X: Bilingual adapters improve zero-shot cross-lingual transfer," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Hum. Lang. Technol.*, 2022, pp. 1791–1799.

[110] M. Tänzer, S. Ruder, and M. Rei, "Memorisation versus generalisation in pre-trained language models," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2022, pp. 7564–7578.

[111] N. Gu, P. Fu, X. Liu, Z. Liu, Z. Lin, and W. Wang, "A gradient control method for backdoor attacks on parameter-efficient tuning," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2023, pp. 3508–3520.

[112] B. Zhu, Y. Qin, G. Cui, Y. Chen, W. Zhao, C. Fu, Y. Deng, Z. Liu, J. Wang, W. Wu, M. Sun, and M. Gu, "Moderate-fitting as a natural backdoor defender for pre-trained language models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022.

[113] L. Hong and T. Wang, "Fewer is more: Trojan attacks on parameter-efficient fine-tuning," *arXiv preprint arXiv:2310.00648*, 2023.

[114] Y. Zeng and K. Lee, "The expressive power of low-rank adaptation," *arXiv preprint arXiv:2310.17513*, 2023.

[115] S.-A. Rebuffi, H. Bilen, and A. Vedaldi, "Efficient parametrization of multi-domain deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8119–8127.

[116] X. He, C. Li, P. Zhang, J. Yang, and X. E. Wang, "Parameter-efficient model adaptation for vision transformers," in *Proc. AAAI Conf. Artif. Intell.*, vol. 37, no. 1, 2023, pp. 817–825.

[117] Z. Xu, Z. Chen, Y. Zhang, Y. Song, X. Wan, and G. Li, "Bridging vision and language encoders: Parameter-efficient tuning for referring image segmentation," in *IEEE Int. Conf. Comput. Vis.*, 2023, pp. 17 503–17 512.

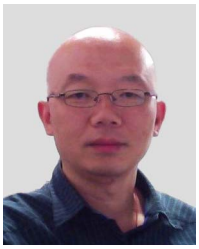
[118] Y.-L. Sung, J. Cho, and M. Bansal, "Vl-adapter: Parameter-efficient transfer learning for vision-and-language tasks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 5227–5237.

[119] J. Pan, Z. Lin, X. Zhu, J. Shao, and H. Li, "St-adapter: Parameter-efficient image-to-video transfer learning," *Proc. Adv. Neural Inf. Process. Syst.*, vol. 35, pp. 26 462–26 477, 2022.

Lingling Xu (Student Member, IEEE) is currently pursuing her Ph.D. degree at Hong Kong Metropolitan University. She received a Master degree in Mathematics from Shandong University. Her research interests include parameter-efficient fine-tuning, contrastive learning, representation learning, and aspect-based sentiment analysis.

Haoran Xie (Senior Member, IEEE) received a Ph.D. degree in Computer Science from City University of Hong Kong and an Ed.D degree in Digital Learning from the University of Bristol. He is currently the Department Head and Associate Professor at the Department of Computing and Decision Sciences, Lingnan University, Hong Kong. His research interests include artificial intelligence, big data, and educational technology. He has published 393 research publications, including 224 journal articles such as IEEE TPAMI, IEEE TKDE, IEEE TAFFC, and IEEE TCVST. He is the Editor-in-Chief of Natural Language Processing Journal, Computers & Education: Artificial Intelligence and Computers & Education: X Reality. He has been selected listed as the World's Top 2% Scientists by Stanford University.

Si-Zhao Joe Qin (Fellow, IEEE) received the B.S. and M.S. degrees in automatic control from Tsinghua University, Beijing, China, in 1984 and 1987, respectively, and the Ph.D. degree in chemical engineering from the University of Maryland, College Park, MD, USA, in 1992. He is currently the Wai Kee Kau Chair Professor and President of Lingnan University, Hong Kong. His research interests include data science and analytics, machine learning, process monitoring, model predictive control, system identification, smart manufacturing, smart cities, and predictive maintenance. Prof. Qin is a Fellow of the U.S. National Academy of Inventors, IFAC, and AIChE. He was the recipient of the 2022 CAST Computing Award by AIChE, 2022 IEEE CSS Transition to Practice Award, U.S. NSF CAREER Award, and NSF-China Outstanding Young Investigator Award. His h-indices for Web of Science, SCOPUS, and Google Scholar are 66, 73, and 84, respectively.



Xiaohui Tao (Senior Member, IEEE) is currently a Full Professor with the University of Southern Queensland, Toowoomba, QLD, Australia. His research interests include artificial intelligence, data analytics, machine learning, knowledge engineering, information retrieval, and health informatics. His research outcomes have been published across more than 150 papers including many top-tier journals and conferences. He is a Senior Member of ACM and the Vice Chair of IEEE Technical Committee of Intelligent Informatics. He was the recipient of an ARC DP in 2022. He is the Editor-in-Chief of Natural Language Processing Journal.

Fu Lee Wang (Senior Member, IEEE) received the B.Eng. degree in computer engineering and the M.Phil. degree in computer science and information systems from The University of Hong Kong, Hong Kong, and the Ph.D. degree in systems engineering and engineering management from The Chinese University of Hong Kong, Hong Kong. Prof. Wang is the Dean of the School of Science and Technology, Hong Kong Metropolitan University, Hong Kong. He has over 300 publications in international journals and conferences and led more than 20 competitive grants with a total greater than HK$20 million. His current research interests include educational technology, information retrieval, computer graphics, and bioinformatics. Prof. Wang is a fellow of BCS, HKIE and IET and a Senior Member of ACM. He was the Chair of the IEEE Hong Kong Section Computer Chapter and ACM Hong Kong Chapter.