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

Lingling Xu, Haoran Xie, S. Joe Qin, Xiaohui Tao, Fu Lee Wang

***Abstract*—With the continuous growth in the number of parameters of the 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, extensive experiments are conducted 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], [5] have demonstrated remarkable performance across a wide range of NLP tasks. To fully harness the potential of PLMs, fine-tuning is commonly employed to adapt them to task-specific data, thereby enhancing performance on downstream tasks. However, traditional fine-tuning involves updating all the pretrained parameters, which is both computationally expensive and time-consuming. This challenge becomes even more pronounced with the increasing size of PLMs, such as BERT [2] with 110 million parameters and T5 [5] with 770 million parameters. As a result, the computational resource demands of full fine-tuning have become a significant barrier to widespread application.

The advent of LLMs [6], [7], [8] has further exacerbated this challenge. For instance, LLaMA-3.1-405B [9], which boasts 405 billion parameters, requires approximately 3.25 TB of memory for task-specific full fine-tuning[1]. The enormous computational resource requirements make full fine-tuning of LLMs infeasible for all but major industry players, rendering it impractical for many.

To address these challenges, a prominent method known as PEFT [10] has emerged as a promising solution. By reducing the number of trainable parameters, PEFT significantly lowers the computational cost of adapting PLMs to specific tasks while maintaining performance comparable to full fine-tuning [10], [11], [12]. Specifically, PEFT typically involves introducing a limited number of trainable parameters, applying low-rank reparameterization to weight updates, or fine-tuning only a specific subset of the pretrained weights, preserving the knowledge captured by the PLM and reducing the risk of catastrophic forgetting. Moreover, PEFT helps mitigate overfitting by selectively updating parameters, which is particularly beneficial when task-specific datasets are much smaller than the pretraining data.

Recently, there has been a significant surge in interest regarding PEFT methods, as demonstrated by the growing number of studies depicted in Fig. 1. While this has led to the publication of a few surveys on PEFT, existing reviews have several limitations. For instance, Ding et al. [13] conducted a comprehensive study but only experimented with four PEFT methods and did not cover much of the latest work in the field. Lialin et al. [14] delved into the ideas and implementations of PEFT methods in detail but did not perform quantitative experiments. These gaps highlight the need for a more thorough and systematic investigation of PEFT methods.

In this paper, we aim to address these gaps by providing a comprehensive and systematic study of PEFT methods for PLMs in NLP. We categorize PEFT methods into five types: additive, partial, reparameterized, hybrid, and unified fine-tuning. This categorization, as shown in Fig. 2, facilitates a deeper understanding of PEFT approaches. Extensive quantitative evaluations compare PEFT and full fine-tuning

The research described in this article has been supported by a research grant entitled "Medical Text Feature Representations based on Pre-trained Language Models" (871238); the Faculty Research Grants (DB24A4 and SDS24A8), and the Direct Grant (DR25E8) of Lingnan University, Hong Kong; and two grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (R1015-23 and UGC/FDS16/E17/23). *(Corresponding author: Haoran Xie.)*

Lingling Xu is with the School of Science and Technology, Hong Kong Metropolitan University, Hong Kong, and also with the School of Data Science, Lingnan University, Hong Kong (email: xxiao199409@gmail.com).

Haoran Xie and S. Joe Qin are with the School of Data Science, Lingnan University, Hong Kong (email: hrxie@ln.edu.hk; joeqin@ln.edu.hk).

Xiaohui Tao is with University of the School of Mathematics, Physics and Computing, Southern Queensland, Queensland, Australia (email: xtao@usq.edu.au).

Fu Lee Wang is with the School of Science and Technology, Hong Kong Metropolitan University, Hong Kong (email: pwang@hkmu.edu.hk)

[1]https://huggingface.co/blog/llama31.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2026.3657354

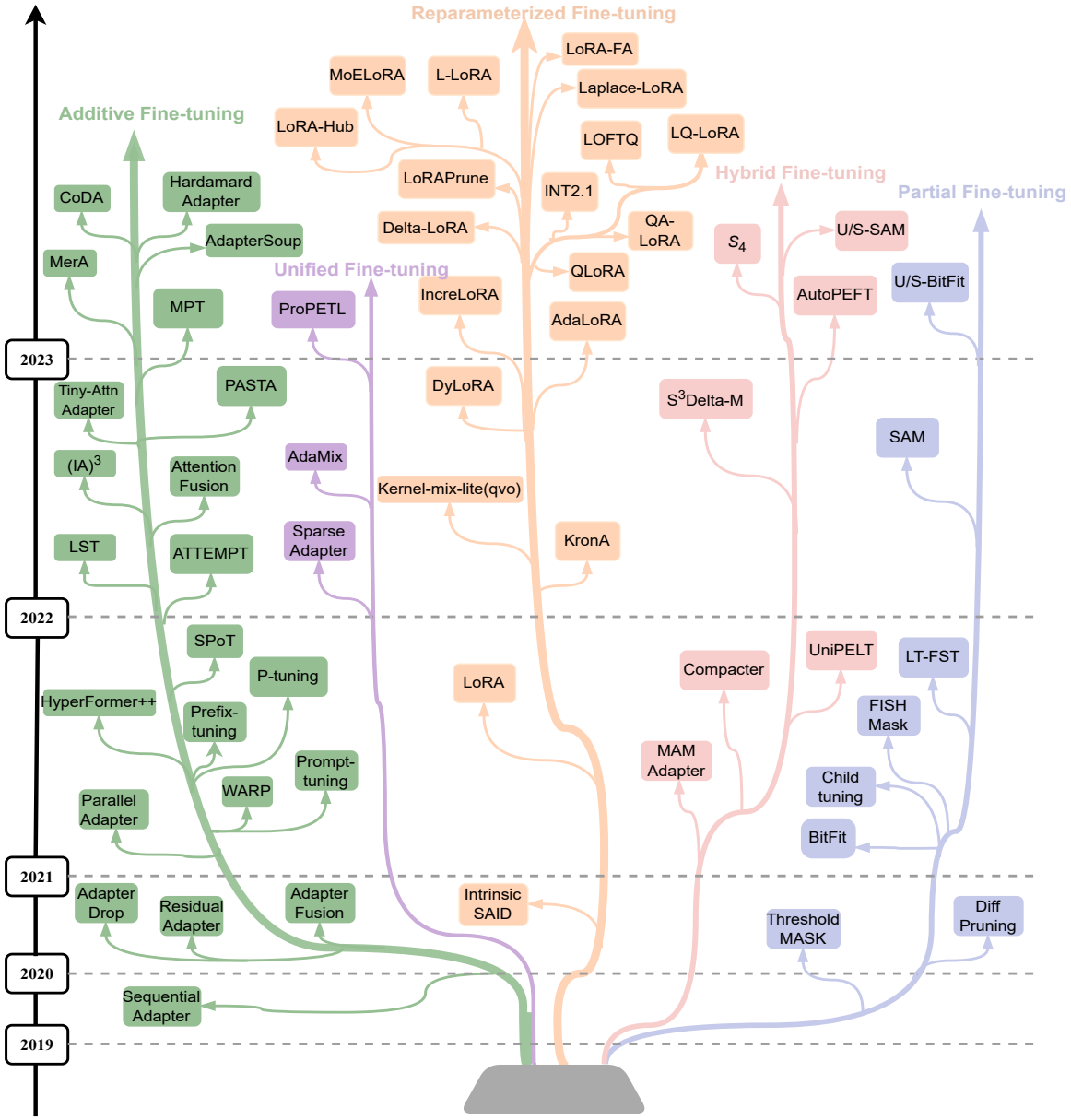


Fig. 1. The evolutionary development of PEFT methods in recent years. Each branch corresponds to subsections in Section III: **Additive Fine-tuning** (Section III-A), **Partial Fine-tuning** (Section III-B), **Reparameterized Fine-tuning** (Section III-C), **Hybrid Fine-tuning** (Section III-D), and **Unified Fine-tuning** (Section III-E). The vertical position of the models shows the timeline of their release dates. 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.

in terms of performance, parameter efficiency, and memory efficiency across various NLP tasks, including natural language understanding (NLU) and machine translation (MT). Beyond this, we explore the applications of PEFT in multi-task learning, cross-lingual transfer, and backdoor attack and defense scenarios, shedding light on its versatility and 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:

- **Comprehensive Review.** We present an in-depth analysis and review of PEFT methods for the Transformer-based PLMs, identifying key techniques and approaches.
- **Systematic Categorization.** We classify PEFT methods into five distinct categories: additive, partial, reparameterized, hybrid, and unified fine-tuning, providing a structured framework for understanding these methods.
- **Extensive Experiments.** We conduct extensive experiments to evaluate the effectiveness of several representative PEFT methods, focusing on their parameter efficiency and memory usage, and analyzing the trade-off between task performance and memory usage.

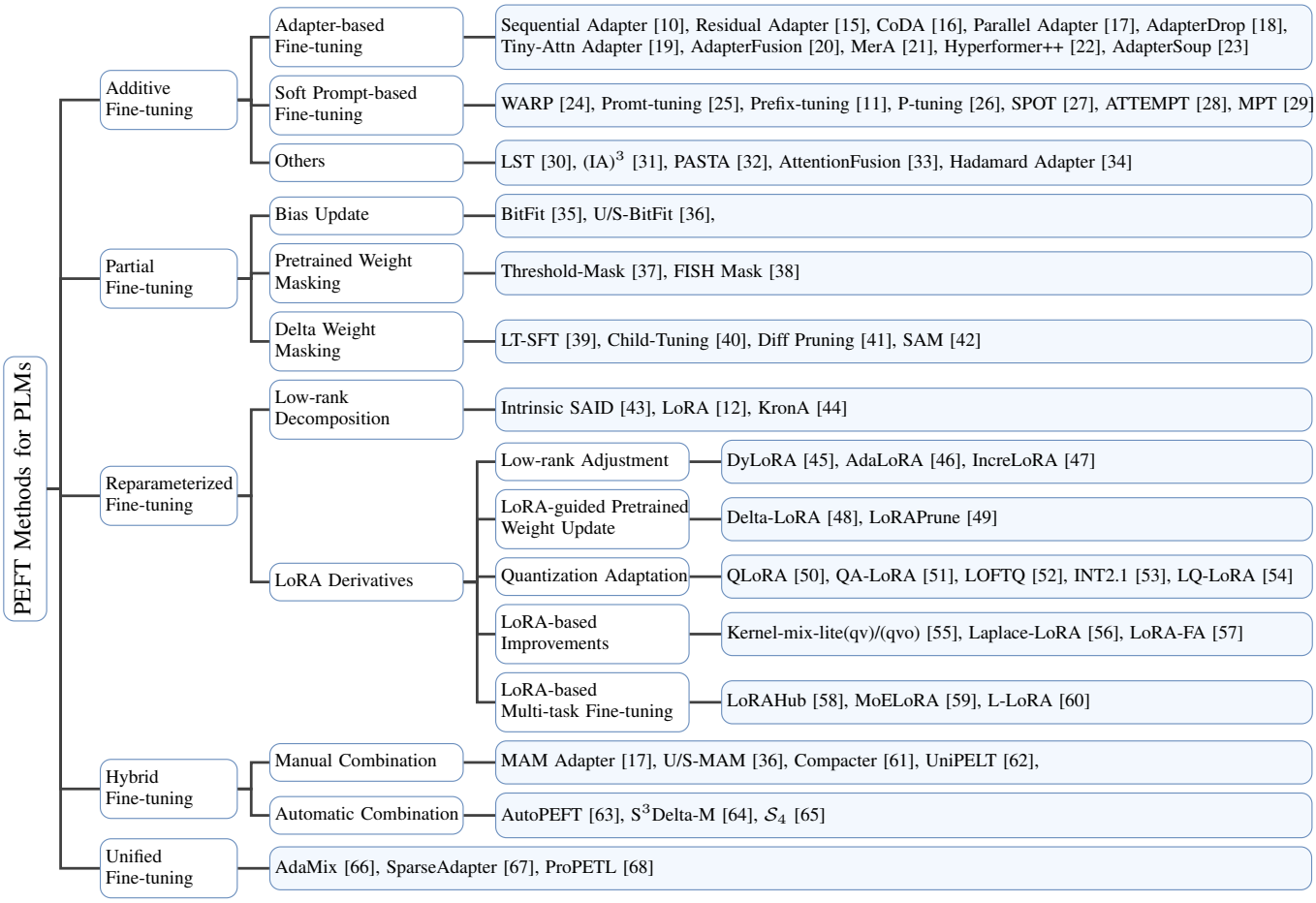


Fig. 2. Taxonomy of PEFT Methods for PLMs in NLP. Based on their characteristics, PEFT methods are categorized into five main categories: **Additive Fine-tuning**, **Partial Fine-tuning**, **Reparameterized Fine-tuning**, **Hybrid Fine-tuning**, and **Unified Fine-tuning**. Each category encompasses various approaches, such as adapter-based, soft prompt-based, and other fine-tuning methods under Additive Fine-tuning.

II. Preliminaries

A. Transformer

The Transformer architecture [1] has become a cornerstone of numerous PLMs. It adopts an encoder-decoder structure, with both components incorporating self-attention mechanisms. Each encoder and decoder consists of a multi-head self-attention (MHA) layer and a feed-forward network (FFN) layer, interconnected by a residual connection [69] followed by layer normalization [70]. 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.

The MHA layer employs the self-attention function with h heads in parallel. Given an input sequence $X \in \mathbb{R}^{n\times d}$, where n is the sentence length and d is the hidden dimension, the query (**Q**), key (**K**), and value (**V**) vectors are computed as linear transformations of the input:

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

where $Q, K, V \in \mathbb{R}^{n\times d}$. Here, W_k, W_q, W_v are learnable weight matrices, and b_k, b_q and b_v are learnable bias vectors. These parameters enable the model to capture context-dependent representations by adjusting the query vectors to better align with the corresponding keys. The self-attention output for the input X is computed as:

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

then MHA aggregates information from multiple subspaces:

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

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

where W_Q^i,W_K^i,W_V^i are head-specific projection matrices, and W^O is the output projection matrix.

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 and W_2 are learnable weight matrices, b_1 and b_2 are bias vectors. Most PEFT methods focus on self-attention and FFN layers, allowing models like encoder-only RoBERTa [3], encoder-decoder T5 [5], and decoder-only LLaMA [8] to leverage relevant techniques for parameter 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. PLMs are initially trained on large-scale datasets with unsupervised learning objectives like language modeling or masked language modeling (MLM) to learn general language representations [2], [3], [5], [8]. However, these PLMs may not perform optimally when applied to specific tasks like sentiment analysis, question answering, or translation due to lack of appropriate domain knowledge [71], [72], [73]. Full fine-tuning provides an effective solution to address this limitation.

During full fine-tuning, the PLM is initialized with pretrained weights and trained on task-specific data using techniques like backpropagation and gradient descent [74], [75]. All model parameters, including pretrained weights, are updated to minimize a task-specific loss that quantifies the disparity between predicted outputs and ground truth. This process allows the model to learn task-specific patterns and nuances from the labeled data, enabling predictions tailored to the target tasks [76]. However, full fine-tuning requires substantial computational resources, as all model parameters must be updated from scratch for each task. With the growing size of PLMs and the emergence of LLMs containing billions of parameters, these computational demands have become even more pronounced. In contrast, PEFT methods mitigate this burden by selectively updating or modifying only specific parts of the PLM, while still achieving performance comparable to full fine-tuning [35], [40]. Moreover, full fine-tuning can lead to overfitting when task-specific data is limited or when the pretrained model is already well aligned with the target task [20], [77].

III. Parameter-Efficient Fine-Tuning Methods

A detailed introduction to PEFT methods for PLMs is presented in this section. To provide a clearer understanding, PEFT methods are categorized into five main types, as depicted in Fig. 2: (A) **Additive Fine-tuning**, where lightweight modules are added to PLMs; (B) **Partial Fine-tuning**, which selectively fine-tunes specific components of PLMs; (C) **Reparameterized Fine-tuning**, which modifies the underlying reparameterization of pretrained weight matrix; (D) **Hybrid Fine-tuning**, which combines different PEFT methods manually or automatically; and (E) **Unified Fine-tuning**, which unifies various PEFT methods with a unified framework.

A. Additive Fine-tuning

Additive fine-tuning approaches involve introducing extra trainable parameters for task-specific fine-tuning. They are classified into three groups: 1) *Adapter-based Fine-tuning*, which integrates lightweight adapter modules into the Transformer to enable modular and efficient task adaptation; 2) *Soft Prompt-based Fine-tuning*, where task-specific information is encoded in soft prompts or prefix vectors that modify input embeddings or hidden states; and 3) *Others*, including various methods that introduce additional parameters but reduce the number of trainable parameters.

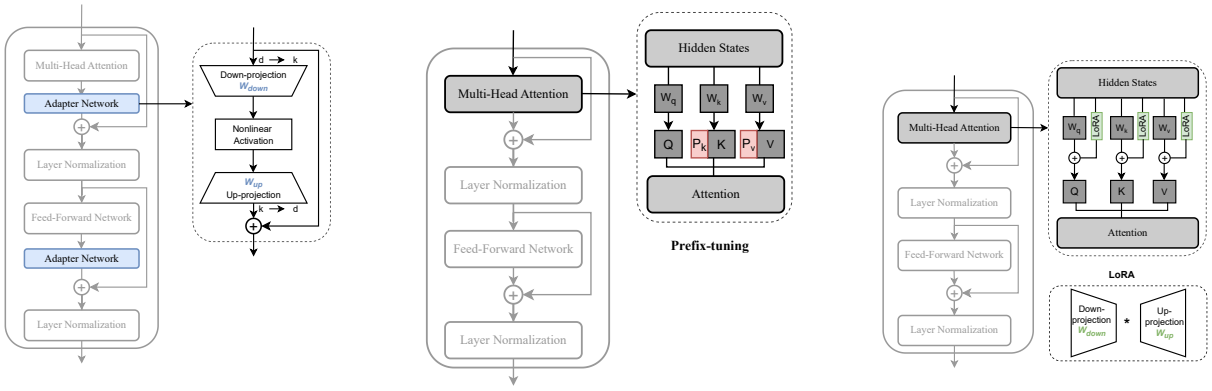
1) Adapter-based Fine-tuning: The idea of Adapter is first introduced in multi-domain image classification [78], allowing for the efficient transfer of knowledge across multiple visual domains. Adapter-based fine-tuning in NLP typically involves removing, dropping, or modifying adapter modules, as well as integrating multiple adapters.

Sequential Adapter [10] extends and applies Adapter to NLP tasks by inserting the adapter (trainable modules) into the Transformer block. These adapters are fine-tuned to adapt PLMs to downstream tasks. Specifically, adapter networks are sequentially inserted after the self-attention and feed-forward layers of the Transformer. Each adapter is a low-rank module that consists of (1) a *down-projection*, which reduces the dimensionality of the input from the original high-dimensional space ($\mathbb{R}^d$) to a smaller intermediate space ($\mathbb{R}^k$, where $k \ll d$); (2) a *non-linear activation function* (e.g., ReLU), which introduces non-linearity and helps the adapter capture complex patterns; and (3) an *up-projection*, which maps the intermediate representation back to the original dimensionality ($\mathbb{R}^d$). Additionally, a residual connection is added to combine the adapter's output with the original input, ensuring efficient information flow. For the input X, the output of sequential adapter with the ReLU activation is defined in Equation 6. During fine-tuning, not only the parameters of adapter network W_{up} and W_{down}, but also the parameters of LayerNorm layers need to be updated to make the PLMs adapt to the specific downstream tasks. The specific architecture of the sequential adapter is illustrated in Fig. 3(a).

$$X = (\text{ReLU}(XW_{down}))W_{up} + X, \tag{6}$$

Inspired by sequential adapter, many adapter-based PEFT methods have been proposed. **Residual Adapter** [15] further improves parameter efficiency by inserting the adapter module only after the FFN and layer normalization. **Parallel Adapter** [17], [79] inserts the adapter network in parallel with both the attention layer and the FFN layer, allowing for more efficient integration of the adapter module into the Transformer. **AdapterDrop** [18] removes adapters in each layer of the Transformer that are not important to the given task to improve inference efficiency. While **CoDA** (Condition Adapter) [16] employs a parallel adapter for task-specific fine-tuning and remains most pretrained parameters fixed. However, unlike prior methods that process all input tokens with the Transformer, CoDA employs a router function to select k important input tokens for conditional computation. This approach enhances both parameter efficiency and inference efficiency. **Tiny-Attn Adapter** (Tiny-Attention Adapter) [19] introduces a dot-product attention module between the down- and up-projections, which can be seen as a multi-head attention module with its per-head dimensionality to be extremely small. Moreover, 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 injected right after the multi-head attention layer.

AdapterFusion [20] integrates multiple task-specific adapters into a single module, allowing for effective knowl-



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

Fig. 3. Architecture of three representative PEFT methods: (a) **Sequential Adapter**, (b) **Prefix-tuning**, and (c) **LoRA**. **Sequential Adapter** introduces lightweight adapter networks into the Transformer layers, using down-projection (W_{down}) and up-projection (W_{up}) operations with a nonlinear activation. **Prefix-tuning** prepends trainable prefix vectors to the hidden states at each layer, modifying the input to the attention mechanism. **LoRA** applies low-rank decomposition to specific weight matrices (e.g., Query (Q), Key (K), and Value (V)) in the attention mechanism, with down-projection (W_{down}) and up-projection (W_{down}) layers operating in parallel to the frozen pretrained weights. These methods represent a broad range of PEFT strategies aimed at reducing the number of trainable parameters while preserving model performance, forming the foundation for numerous subsequent PEFT methods.

edge transfer across related tasks without modifying the original pretrained model. It 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 model fine-tuning. However, AdapterFusion requires additional trainable parameters in the composition layers, which increases the computational costs. **MerA** (Merging Pretrained Adapters) [21] addresses this limitation by merging pretrained adapter parameters through summation and averaging strategies, avoiding the introduction of other extra trainable parameters. Employing the optimal transport method [80], [81] to align adapter weights and activations, MerA achieves better performance with fewer trainable parameters compared to AdapterFusion. **Hyperformer++** [22] utilizes a shared hypernetwork [82] to learn task-specific and layer-specific adapter parameters conditioned on task and layer ID embeddings. By sharing knowledge across tasks via hypernetworks and enabling the model to adapt to each task through task-specific adapters, significantly reducing the number of trainable parameters. **AdapterSoup** [23] is developed to address cross-domain task adaptation, which first trains multiple adapters across various domains and then employs domain clustering [83] 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. Additionally, AdapterSoup can 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.

WARP (Word-level Adversarial RePrograming) [24] appends special prompt tokens $[P_1], [P_2], \cdots, [P_l]$ and a [Mask] token before or after the sentences, depending 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, significantly reducing the number of trainable parameters. **Prompt-tuning** [25] 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 $\hat{X}$, as defined in Equation 7. During fine-tuning, only the prompt parameters of P are updated via gradient descent, while pretrained parameters remain frozen. The parameter cost of prompt-tuning is determined by the prompt length and token embedding dimension.

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

Prefix-tuning [11] proposes to prepend soft prompts $P = [P_1], [P_2], \cdots, [P_l]$ (l denotes the prefix length) to the hidden states of the multi-head attention layer. To stabilize training, an FFN is used to parameterize the soft prompts, as directly optimizing their values can cause instability. Two sets of prefix vectors, $\hat{P}_k$ and $\hat{P}_v$, are concatenated with the original key (K) and value (V) vectors of the attention layer. The self-attention mechanism with prefix-tuning is represented by Equation 8 and 9. During training, only $\hat{P}_k$, $\hat{P}_v$, and the FFN parameters are optimized, while all other parameters of PLMs remain frozen. After training, the FFN is discarded, and only P_k and P_v are used for inference. The structure of prefix-tuning is shown in Fig. 3(b). **P-tuning** [26] also leverages soft prompts, $[P_1], \cdots, [P_i], [P_{i+1}], \cdots, [P_l]$, but differs in its approach. It concatenates these prompts to form a template, which is mapped to the sequence $\{h_1^p, \cdots, h_i^p, e(x), h_{i+1}^p, \cdots, h_l^p, e(x)\}$, where e represents pretrained embedding layer. The training goal is to optimize the continuous prompts $\{h_1^p, \cdots, h_l^p\}$. P-tuning is particularly

effective in few-shot learning scenarios because only the continuous prompts are fine-tuned. To initialize the embedding of soft prompts, P-tuning employs a bidirectional long short-term memory network (LSTM) with a ReLU-activated multilayer perceptron (MLP) through MLP(LSTM($h_1^p, \cdots, h_i^p$): LSTM($h_i^p, \cdots, h_l^p$)).

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

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

SPOT (Soft Prompt Transfer) [27] is a multi-task prompt method that builds upon prompt-tuning, in which "prompt pertaining" is introduced between PLMs and prompt-tuning. 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. 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) [28] 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 prompts. This enables modular multi-task learning, as pretrained soft prompts can be flexibly combined, reused, or removed to leverage knowledge from different tasks. **MPT** (multi-task prompt tuning) [29] utilizes multi-task 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 t-th source task, denoted as $\hat{P}_t$, can be decomposed into the shared prompts across all source tasks P^* and a task-specific low-rank matrix W_t. The decomposition is given by $\hat{P}_t = P^* \odot W_t = P^* \odot (u_t \otimes v_t^T)$, where $\odot$ denotes the Hadamard product, $\otimes$ denotes the Kronecker product, and u_t and v_t are task-specific vectors for the task t.

3) Others: Apart from adapter family and soft prompt-based fine-tuning methods, several other approaches also incorporate extra trainable parameters during fine-tuning. They involve adding a ladder side network operating alongside the Transformer, introducing additional vectors to rescale the attention, incorporating extra vectors for special token representations, using the late fusion technique to integrate additional attention weight, and combing an extra joint importance weight for each token representation.

LST (Ladder Side-Tuning) [30] trains a ladder side network in conjunction with the pretrained network and takes intermediate activations as input via shortcut connections, known as ladders. Since all training parameters are stored in the ladder side network, back-propagation occurs through the side networks and ladder connections rather than pretrained networks, reducing the number of fine-tuned parameters. To further boost parameter efficiency, LST employs structural pruning [84] to retrieve a smaller, pruned network to initialize the side network and drops certain layers of the side network. **(IA)**^{3} (Infused Adapter by Inhibiting and Amplifying Inner Activations) [31] introduces three learned vectors, l_k, l_v, and l_{ff}, to rescale the key and value vectors in the attention networks and hidden activations in the position-wise FFN. The attention output and hidden activation output are rescaled as:

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

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

where $\odot$ denotes 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 the 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) [32] enhances parameter efficiency by modifying the special token representations (e.g., [SEP] and [CLS]) through an extra trainable vector inserted before the self-attention layer in each Transformer. Given an input $H = \{h_i\}_{i=1}^N$ to a Transformer layer, PASTA modifies the inputs as:

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

where the special token adaptation g_i is defined as:

$$g_i = \begin{cases} e(v_p) & \text{if token } i \text{ is the } p\text{-th special token,} \\ 0 & \text{otherwise.} \end{cases}$$

Here, $e(v_p)$ is trainable vector added to the hidden representation of the p-th special token. By updating only $e(v_p)$, PASTA drastically reduces the number of trainable parameters. The use of [CLS] and [SEP] as special tokens in PASTA is justified for two reasons: the [CLS] representation offers a global representation of input text, and attention scores in PLMs are primarily allocated to these tokens across attention heads [85], [86]. **AttentionFusion** [33] employs the late fusion technique, which combines features or representations from diverse tasks or layers to generate a final joint representation, dynamically adjusting the importance of each token representation. For a given task t, let the attention query vector be Q^t, and the representation of token i at layer j be V_i^j. The token representation for task t, $\hat{V}_i^j$, is defined 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 equal to the hidden dimension of the pretrained encoder. By employing these attention weights as extra trainable parameters, AttentionFusion adjusts the importance of each token representation dynamically. **Hadamard**

TABLE I
WEIGHT UPDATE METHODS FOR PRETRAINED WEIGHTS (W) USING MASKING IN PARTIAL FINE-TUNING.

Method	Weight Update	Mask Criterion	Mask Matrix
Threshold-Mask	$\hat{W} = M \odot W$	Threshold	$M_{i,j} = \mathbb{I}_{S_{i,j} > \tau}$
FISH Mask	$\hat{W} = M \odot W$	Fisher information of pretrained weight	$M_{i,j} = \mathbb{I}_{f_{i,j} \in \text{top-k}(f)}$
LT-SFT	$\hat{W} = W + M \odot \nabla_W \mathcal{L}(W)$	Absolute difference of parameters	$M_{i,j} = \mathbb{I}_{\lvert W_{1,i,j} - W_{0,i,j} \rvert \in \text{top-k}(\lvert W_1 - W_0 \rvert)}$
Child-Tuning$_F$	$\hat{W} = W + M \odot \eta \nabla_W \mathcal{L}(W)$	Bernoulli distribution	$M_{i,j} \sim \text{Bernoulli}(\text{p}_F)$
Child-Tuning$_D$	$\hat{W} = W + M \odot \eta \nabla_W \mathcal{L}(W)$	Fisher information of child network	$M_{i,j} = \mathbb{I}_{f_{i,j} \in \text{top-k}(f_{\text{child}})}$
Diff Pruning	$\hat{W} = W + M \odot w_\tau$	Fixed sparsity	$M \in \{0,1\}^d, w_\tau \in \mathbb{R}^d$, sparsity $= \frac{\lVert M \rVert_0}{d}$
SAM	$\hat{W} = W + M \Delta W$	Analytical solution	$M_{i,j} = 0, \forall i \neq j$; $M_{i,i} \in \{0,1\}$

Adapter [34] introduces a weight and bias vector in the Transformer layer, matching the dimensions of the multi-head attention module's output. These weight and bias vectors are injected immediately after the multi-head attention layer, performing element-wise multiplication (Hadamard product) with the attention outputs. The number of Hadamard adapter is the same as that of Transformer layers in PLMs. During fine-tuning, only the parameters in the Hadamard adapter, layer normalization, and classifier are updated.

B. Partial Fine-tuning

Partial fine-tuning methods reduce the number of trainable parameters by updating a subset of pretrained parameters deemed crucial to downstream tasks while discarding unimportant ones. They are categorized into three groups: 1) *Bias Update*, where only the bias terms in the attention, FFN layers, and layer normalization of the Transformer are updated; 2) *Pretrained Weight Masking*, where pretrained weights are masked using various pruning criteria; and 3) *Delta Weight Masking*, where delta weights are masked via pruning techniques and optimization approximation. A detailed analysis of weight update with masking is provided in Table I.

1) Bias Update: It involves modifying only the bias terms in the attention layer, feed-forward layer, and layer normalization of the Transformer. Additionally, it may include pruning bias terms using various optimization algorithms.

Bit-Fit (Bias-term Fine-tuning) [35] 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 and value, and combining multiple attention heads, as well as in the fee-forward and layer normalization layers. Further, **U/S-BitFit** [36] combines the neural architecture search (NAS) algorithm [87] 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.

2) Pretrained Weight Masking: Pretrained weight masking employs pruning criteria, such as threshold and Fisher information, to measure the importance of pretrained weight to construct a binary mask matrix for masking.

Threshold-Mask [37] utilizes the threshold to construct a binary mask matrix M to select pretrained weights W of the attention and FFN layers through element-wise multiplication, expressed as $\hat{W} = W \odot M$. A random, uniformly distributed, real-valued matrix S that shares the same dimensions as W and M is created. A binary mask matrix is generated by assigning 1 to positions where an element in S exceeds a predefined global threshold τ, and 0 otherwise. **FISH Mask** (Fisher-Induced Sparse uncHanging) [38] uses the Fisher information of pretrained weights to measure their importance and construct a sparse binary mask M. It selects the top-k parameters with the largest Fisher information to construct this mask, setting the positions corresponding to these top-k parameters to 1, while the remaining positions are set to 0. The value of 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 involves employing various approaches or pruning techniques to construct a binary mask matrix to reduce trainable parameters.

LT-SFT (Lottery Ticket Sparse Fine-Tuning) [39] is a novel PEFT method inspired by the Lottery Ticket Hypothesis[2] [88]. LT-SFT begins by fine-tuning the PLM on target data using the initial parameters W_0, resulting in fully fine-tuned parameters W_1. It then identifies the top-k pretrained parameters with the greatest absolute differences ($|W_1 - W_0|$). These selected top-k parameters are further fine-tuned using a binary mask M, where positions corresponding to the chosen parameters are set to 1, and all other positions are set to 0. The model parameters are then reset to their original pretrained weights W_0, but only the selected k parameters are fine-tuned, expressed as $\delta = M \odot \nabla_W \mathcal{L}(W)$. By iteratively repeating this process, LT-SFT gradually fine-tunes only a small fraction of the model's parameters. **Child-Tuning** [40] defines a "child network" as the subset of parameters to be updated and masks out the gradients of non-child networks to improve parameter efficiency. The parameter updates in Child-Tuning are expressed as $\delta = M \odot \eta \nabla_W \mathcal{L}(W)$ (η denotes learning rate). Child-Tuning provides two variants: **Child-Tuning**$_F$ (F denotes Task-Free) and **Child-Tuning**$_D$ (D denotes Task-

[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.

Driven). Child-Tuning$_F$ generates the binary mask matrix M using Bernoulli distribution with a probability denoted as p_F. Increasing p_F updates more parameters, and Child-Tuning$_F$ is equivalent to full fine-tuning when $\mathrm{p}_F = 1$. Child-Tuning$_D$ uses Fisher information estimation to identify a subset of parameters (the child network) that are highly correlated with a specific downstream task. The binary mask matrix M in Child-Tuning$_D$ is created by assigning 1 to positions corresponding to the child network and 0 to the non-child network.

Diff Pruning [41] introduces a task-specific, sparse diff vector δ during fine-tuning while keeping the pretrained weight W fixed. To induce sparsity, δ is decomposed as $\delta = M \odot w_\tau$, where M is a learnable binary mask and w_τ is a dense diff weight. The binary mask M is optimized with the Hard-Concrete relaxation, which provides a differentiable approximation to the L0-norm and makes most entries of δ to zero. Due to only the sparse diff vector needing to be stored for each task, Diff Pruning is particularly suitable for on-device deployment settings, where tasks may arrive sequentially or come from different providers. **SAM** (Second-order Approximation Method) [42] 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 [42] views the PEFT methods as p-sparse fine-tuned models by representing fine-tuned parameters as $W = W_0 + M\Delta W$, where M is a mask matrix, and the optimization problem is

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

where $m = \dim(W)$ and p is fixed sparsity. SAM obtains gradient $\nabla\mathcal{L}(W_0)_i$ for the 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 operation with high-dimensional matrices (e.g., pretrained weights). These methods are categorized into two groups: 1) *Low-rank Decomposition*, where various low-rank decomposition techniques are used to reparameterize the updated matrix; and 2) *LoRA Derivatives*, where various PEFT methods are developed based on LoRA. Specific details of delta weight (ΔW) parameter reparameterization with 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) [43] leverages the concept of intrinsic dimensionality to reduce the number of parameters during fine-tuning. Intrinsic dimensionality refers to the minimum dimensionality required to solve a high-dimensional optimization problem. Instead of optimizing the empirical loss in the original parameterization, Intrinsic SAID fine-tunes the model by reparametrizing the model in a lower-dimensional space, i.e., $\Delta W = \mathcal{F}(W^r)$, where W^r is the parameter being optimized and $\mathcal{F} : \mathbb{R}^r \rightarrow \mathbb{R}^d$ is a Fastfood transform[3] [89] 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 model parameters.

Inspired by Intrinsic SAID, **LoRA** (Low-Rank Adaptation) [12] introduces two trainable low-rank matrices, down-projection matrix W_{down} and up-projection matrix W_{up}, to update pretrained weight W. Specifically, these matrices are used in parallel with the query, key, and value matrices in attention layer of Transformer, as shown in Fig. 3(c). For pretrained weight $W \in \mathbb{R}^{d\times k}$, LoRA updates weight as $\Delta W = W_{down}W_{up}$, where $W_{down} \in \mathbb{R}^{d\times r}$ projects the input from the high-dimensional ($\mathbb{R}^d$) to a lower-dimensional intermediate space ($\mathbb{R}^r$), $W_{up} \in \mathbb{R}^{r\times k}$ maps the compressed representation back to the original output space ($\mathbb{R}^k$). Low-rank decomposition dramatically reduces the number of trainable parameters, as $r \ll min\{d, k\}$. During training, the weights of PLMs are frozen, and only W_{down} and W_{up} are fine-tuned, enabling parameter-efficient adaptation to new tasks. A scaling factor is typically added to the LoRA module.

KronA (Kronecker Adapter) [44] is structurally similar to LoRA but replaces the low-rank decomposition in LoRA with Kronecker product decomposition, $\Delta W = W_{down} \otimes W_{up}$. This decomposition maintains the rank of the input matrix ($\text{rank}(A \otimes B) = \text{rank}(A) \times \text{rank}(B)$), 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 by avoiding the explicit reconstruction of the Kronecker product matrix. KronA has two variants: **KronA**$_B$ and **KronA**$^B_{\textbf{res}}$. 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: This includes a range of PEFT methods built upon LoRA, including **Low-Rank Adjustment**, which explores dynamic rank adjustment strategies to improve LoRA's adaptability; **LoRA-guided Pretrained Weight Update**, which leverages LoRA to guide the update of pretrained weight; **Quantization Adaptation**, which integrates various quantization techniques to enhance the efficiency of LoRA-based fine-tuning and inference; **LoRA-based Improvements**, which incorporates novel techniques into LoRA for improvements; and **LoRA-based Multi-task Fine-tuning**, which combines multiple LoRA modules for effective cross-task transfer.

Low-rank Adjustment. DyLoRA (Dynamic LoRA) [45] is introduced to overcome two limitations of LoRA: (a) its fixed rank, which cannot be adjusted after training, and (b)

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

TABLE II
DELTA WEIGHT (ΔW) REPARAMETERIZATION OF VARIOUS REPARAMETERIZED FINE-TUNING METHODS.

Method	ΔW **Reparameterization**	**Notes**
Intrinsic SAID	$\Delta W = \mathcal{F}(W^r)$	$\mathcal{F}: \mathbb{R}^r \to \mathbb{R}^d$, $W^r \in \mathbb{R}^r$ is parameters to be optimized, $r \ll d$.
LoRA	$\Delta W = W_{down} W_{up}$	$W_{down} \in \mathbb{R}^{k \times r}$, $W_{up} \in \mathbb{R}^{r \times d}$, low-rank matrices with $r \ll \{k, d\}$.
KronA	$\Delta W = W_{down} \otimes W_{up}$	$\text{rank}(W_{down} \otimes W_{up}) = \text{rank}(W_{down}) \times \text{rank}(W_{up})$, $\otimes$ denotes Kronecker product.
DyLoRA	$\Delta W = W_{down\downarrow r_b} W_{up\downarrow r_b}$	$W_{down\downarrow r_b} = W_{down}[: r_b, :]$, $W_{up\downarrow r_b} = W_{up}[:, : r_b]$, $r_b \in \{r_{min}, \cdots, r_{max}\}$.
AdaLoRA	$\Delta W = U \Lambda R$	$UU^T = U^T U = I = RR^T = R^T R$, $\Lambda = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r)$, $\sigma_i \neq 0$.
IncreLoRA	$\Delta W = W_{down} \Lambda W_{up}$	$\Lambda = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r)$, $\sigma_i \neq 0$.
DeltaLoRA	$\Delta W = W_{down} W_{up}$	$\Delta W = W_{down}^{(t+1)} W_{up}^{(t+1)} - W_{down}^{(t)} W_{up}^{(t)})$.
LoRAPrune	$\Delta W = (W_{down} W_{up}) \odot M$	M is a binary mask matrix, $\odot$ denotes Hadamard product.
QLoRA	$\Delta W = W_{down}^{\text{BF16}} W_{up}^{\text{BF16}}$	Quantized low-rank matrices (BF16) with double quantization for memory efficiency.
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	$\Delta W = \text{SVD}(W - Q_t)$	SVD applied to residual updates, Q_t is quantization function.
Kernel-mix	$\Delta W^h = (B_{\text{LoRA}}, B^h) \begin{pmatrix} A^h_{\text{LoRA}} \\ A^h \end{pmatrix}$	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}.

the need for exhaustive search to determine the optimal rank. To overcome these issues, DyLoRA trains LoRA modules across a range of ranks, allowing for adaptability. Specifically, DyLoRA iterates over ranks within a predefined range $r \in [r_{min}, r_{max}]$. In each iteration, DyLoRA randomly selects a specific rank r_b from $\{r_{min}, \cdots, r_{max}\}$, truncates the down-projection matrix as $W_{down\downarrow r_b} = W_{down}[: r_b, :]$ and the up-projection matrix as $W_{up\downarrow r_b} = W_{up}[:, : r_b]$ and only update truncated parameter matrices. The parameter updates for each iteration are expressed as $\Delta W = W_{down\downarrow r_b} W_{up\downarrow r_b}$. 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) [46] extends LoRA by dynamically adjusting the matrix rank to optimize parameter allocation. It reparameterizes the incremental update ΔW using singular value decomposition (SVD) as $\Delta W = U \Lambda R$, where U and R are orthogonal matrices, and Λ is a diagonal matrix containing the singular values. During training, U and R are initialized with Gaussian distribution with a regularizer to ensure the orthogonality, while Λ is initialized to zero and iteratively pruned to adjust the rank. AdaLoRA employs the sensitivity-based importance scoring [90], [91] with a new metric to prune unimportant singular values, improving parameter efficiency and optimizing allocation budgets. **IncreLoRA** [47] dynamically increases the ranks of LoRA modules during training, guided by importance scores assigned to each module. Less important modules are assigned lower ranks (potentially zero, indicating no updates), while more important ones receive higher ranks. Parameter updates follow $\Delta W = W_{down} \Lambda W_{up}$, where Λ is a rank-r diagonal matrix (with r being the rank of the LoRA module). To control parameter growth, an upper bound on the rank is set for each module. IncreLoRA also introduces a pretraining technique called "advance learning", which ensures newly added parameters are initialized effectively, preventing insufficient training of incremental parameters. Unlike LoRA, which operates on the query, key, and value projection modules in the attention layer, IncreLoRA applies parameter updates to all linear layers, enabling broader adaptation.

LoRA-guided Pretrained Weight Update. Delta-LoRA [48] updates the pretrained weight W as well as two low-rank matrices W_{down} and W_{up}, while maintaining the same memory as the original LoRA. The low-rank matrices W_{down} and W_{up} are updated as usual, but the pretrained weight W leverages the mathematical property for parameter updates: $\nabla_W \mathcal{L}(W, W_{down}, W_{up}) = \nabla_{W_{down} W_{up}} \mathcal{L}(W, W_{down}, W_{up})$. This is achieved by removing the dropout layer in the original LoRA module. 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** [49] introduces a LoRA-guided pruning criterion that utilizes the weights and gradients of LoRA, rather than those of pretrained weights, to estimate parameter importance and prune both LoRA and pretrained parameters. To address the substantial memory overhead of unstructured pruning and dependency-aware structured pruning, LoRAPrune devises a structured iterative pruning procedure that selectively eliminates redundant channels and heads. Specifically, the pruning criterion uses the low-rank matrices W_{down} and W_{up}, and their corresponding gradients $\nabla_{W_{down}}$ and $\nabla_{W_{up}}$, to calculate the importance score[4]. Weights with low importance scores are pruned. LoRAPrune not only prunes structured weights, such as heads and channels, from the pretrained weights but also prunes the corresponding weights in the LoRA modules. The pruned weights are represented as $\delta = (W + W_{down} W_{up}) \odot M$, $M \in \{0, 1\}^{1 \times G}$, where G is the group number. Binary mask M is set to 0 for unimportant groups and 1 for important ones. After pruning and fine-tuning, the LoRA weights seamlessly merge with the pretrained weights, ensuring no additional computational overhea during inference.

Quantization Adaptation. QLoRA [50], a quantized variant of LoRA, addresses the computational resource limitations of LoRA by quantizing the Transformer model to 4-bit NormalFloat (NF4) precision using double quantization processing, and a paged optimizer to mitigate memory spikes. While QLoRA quantizes pretrained weight W from FP16

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

into NF4 so that LLMs can be fine-tuned with fewer GPUs, the low-rank weight matrices W_{down} and W_{up} reverts the final weight to FP16 again after fine-tuning, limiting its efficiency. To this end, **QA-LoRA** (Quantization-Aware Low-rank Adaptation) [51] introduces group-wise quantization with low-rank adaptation by partitioning each column of the pretrained weight W into L groups for quantization. This design ensures that both pretrained and adaptation weights remain quantized after fine-tuning, enabling faster and more accurate inference. Similarly, **LOFTQ** (LoRA-Fine-Tuning-aware Quantization) [52] proposes a quantization-aware initialization strategy that approximates the high-precision pretrained weights using N-bit quantized weights and low-rank matrices W_{down} and W_{up}. This initialization mitigates the quantization discrepancy observed in QLoRA and significantly improves generalization, especially effective in 2-bit and 2/4-bit mixed-precision settings. While **INT2.1** [53] leverages the GPTQ quantization method [92] to compress pretrained weights and integrates LoRA for fine-tuning of LLMs. It introduces a low-rank error correction mechanism that enhances performance by compensating for quantization error, even under extremely low-bit conditions (e.g., 2-bit). **LQ-LORA** [54] is a mixed-precision PEFT method that decomposes pretrained weight into fixed quantized component and trainable low-rank component. It formulates the quantization process as an integer linear programming problem, allowing dynamic configuration of quantization parameters (e.g., bit-width, block size) under a global memory budget. This allows aggressive sub-3-bit quantization (e.g., 2.75-bit) with minimal performance degradation.

LoRA-based Improvements. Kernel-wise Adapter [55] 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 base model to combine kernel-wise adaptation for its flexibility in parameter assignment across different weight matrices. Kernel-wise adapter has two variants: **Kernel-mix-lite (qv)** and **Kernel-mix-lite (qvo)**. Kernel-mix-lite (qv) is designed for scenarios with limited parameter budgets, while Kernel-mix (qvo) is suitable for 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** [56] incorporates Bayesian inference into LoRA parameters to address overconfidence and improve calibration. A key challenge lies in obtaining the posterior distribution for Bayesian inference, which is resolved by using Laplace approximation [93]. Laplace-LoRA approximates the posterior distribution over LoRA parameters, enabling it to maintain standard pretraining and fine-tuning procedures while reducing the dimensionality of Bayesian inference. **LoRA-FA** (LoRA with Frozen-A) [57] reduces the activation memory of LoRA by keeping the pretrained weight W and down-projection matrix W_{down} frozen and only updating the up-projection matrix W_{up}. W_{down} is decomposed into Q_{down} and R_{down} via QR decomposition, and $\Delta W = W_{down}W_{up} = Q_{down}R_{down}W_{up} = Q_{down}\hat{W}_{up} = \sum_{i=1}^{r} Q_{down,:,i}\hat{W}_{up,i,:}$, where $\{Q_{down,:,i}\}_{i=1}^{r}$ are orthogonal unit vectors (r is the rank of W_{down}). This ensures that ΔW resides in a low-rank orthogonal subspace, avoiding full-rank activation storage.

LoRA-based Multi-task Fine-tuning. LoRAHub [58] leverages multiple trained LoRA modules for cross-task transfer to fine-tune model on new tasks. It trains task-specific LoRA modules in various tasks to obtain a synthesized module, $\hat{m} = (w_1 W_{down}^1 + \cdots + w_N W_{down}^N)(w_1 W_{up}^1 + \cdots + w_N W_{up}^N)$, which is amalgamated with LLMs for new task adaptation. 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 [94]. **MOELoRA** [59] combines LoRA with mixture-of-experts (MoE) for multi-task fine-tuning, where each expert is a LoRA module for learning task-specific knowledge. Additionally, a task-motivated gate function is devised to produce distinct, fine-tuned parameters for various tasks. **L-LoRA** (Linearized LoRA) [60] 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 first-order Taylor expansion, as illustrated in Equation 14. 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 [95], [96], ties-merging [97], and LoRAhub [58] are employed for multi-task fusion.

$$f_{\theta_0}(x;\phi(t)) \approx f_{\theta_0}^{\text{lin}}(x;\phi(t)) = f_{\theta_0}(x;\phi(0)) + \nabla_\phi f_{\theta_0}(x;\phi(0))^T(\phi(t)-\phi(0)). \quad (14)$$

D. Hybrid Fine-Tuning

Hybrid fine-tuning aims to combine multiple PEFT strategies like adapter, prefix-tuning, and LoRA, to leverage their strengths and mitigate their weaknesses. By integrating features from different PEFT methods, hybrid fine-tuning achieves improved overall performance compared to individual PEFT methods. These methods are categorized into two groups: 1) *Mannual Combination*, which combines multiple PEFT methods manually by sophisticated design; and 2) *Automatic Combination*, where PEFT methods are integrated automatically via structure search or optimization algorithms.

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) [17] combines the strengths of scaled parallel adapter and prefix-tuning. Concretely, it employs prefix-tuning with smaller bottleneck dimensions at the attention layer and allocates more parameter budget to modify the representation of FFN using the scaled parallel adapter, which introduces a scaling factor to adjust the adapter output. Further, **U-MAM** (Unstructured MAM) and **S-MAM** (Structured MAM) [36] are proposed, combining NAS algorithm [87] with pruning techniques to automatically determine which parameters of the network should be fine-tuned. NAS algorithm takes the maximum number of parameters required for PEFT architectures as input and applies the pruning operation to reduce trainable parameters. The pruning criterion is 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 [61] integrates adapters, low-rank optimization, and parameterized hypercomplex multiplication (PHM) layers [98] to improve parameter efficiency. It follows a structure similar to adapters, consisting of a down-projection, a nonlinear activation function, and an up-projection. However, Compacter replaces the down- and up-projection in the adapters with a low-rank PHM layer. Structurally, PHM layer resembles a fully connected layer, but its weight matrix W is represented as a sum of Kronecker products, $W = \sum_{i=1}^{n} A_i \otimes B_i$, where A_i is a shared parameter across all adapter layers, while B_i represents adapter-specific parameters. Compacter extends this idea by reparameterizing B_i as the product of two low-rank matrices, enabling further parameter reduction. The weight matrix in each low-rank PHM layer is calculated as:

$$W = \sum_{i=1}^{n} A_i \otimes B_i = \sum_{i=1}^{n} A_i \otimes (s_i t_i^{\mathrm{T}}), \tag{15}$$

where s_i and t_i are low-rank matrices. **Compacter++** is a variant of Compacter that inserts a Compacter layer after the FFN layer of the Transformer module and requires fewer parameters to be updated than Compacter.

UniPELT [62] incorporates sequential adapter, prefix-tuning, and LoRA via a gating mechanism. In UniPELT, adapters are added after the FFN layer, prefix-tuning is applied to the key and value vectors of the MHA layer, and LoRA is used in attention matrices of W_q and W_v. Each PEFT module is equipped with a gating mechanism comprising a linear function with the dimension of the output being 1, a sigmoid function, and a mean function. This mechanism controls the activation of each submodule, dynamically assigning higher weights to submodules that make positive contributions to the task. The trainable parameters encompass LoRA matrices W_{down} and W_{up}, prefix-tuning parameters P_k and P_v, adapter parameters, and weights for the gating function. While UniPELT requires more parameters and inference time than adapter, prefix-tuning, and LoRA, it achieves superior performance by leveraging the strengths of all three approaches.

2) Automatic Combination: This explores how to configure PEFT methods like adapters, prefix-tuning, and LoRA to different layers of the Transformer automatically using various structure search and optimization approaches. However, it typically requires more time and cost due to the need to perform optimization searches.

AutoPEFT [63] integrates serial adapter, parallel adapter, and prefix-tuning into the Transformer block. The serial adapter receives the hidden state from the FFN output as input, while parallel adapter takes the hidden state before the FFN layer as its input. In addition, 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 NAS algorithm, AutoPEFT employs a Bayesian optimization approach to automatically search for suitable architectures that selectively activate certain layers to incorporate these PEFT modules. Bayesian optimization is sample-efficient, zeroth-order, and well-suited for multi-objective setups, enabling cost-efficient optimization while balancing performance and resource usage. Moreover, it is more parallelizable during search, decreasing memory usage.

S^{3}Delta-M (Search for Sparse Structure of Delta Tuning Mix) [64] is a mixture of LoRA, Compacter (low-rank adapter), BitFit, and LNFit[5]. Different from incorporating PEFT techniques directly, S^{3}Delta-M employs a differentiable delta tuning structure search to explicitly control sparsity and identify the optimal combination of these techniques in a unified search space. Each PEFT module is strategically inserted into the corresponding layers of the PLM, with the specific combination and placement determined through the sparsity-guided structure search. $\boldsymbol{\mathcal{S}_4}$ [65] builds on this idea by combining sequential adapter, prefix-tuning, BitFit, and LoRA. Unlike methods that utilize the same PEFT module uniformly across all Transformer layers, $\mathcal{S}_4$ introduces a more granular design by grouping layers into four "spindle-patterned" groups: G_1, G_2, G_3, G_4. In this pattern, 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, trainable parameters are allocated uniformly, ensuring the number of trainable parameters in each layer remains consistent across all groups. Each group is equipped with a unique combination of sequential adapters (A), prefix-tuning (P), BitFit (B), and LoRA (L). Extensive experimental results demonstrate that equipping groups with tailored combinations of PEFT methods leads to improved performance compared to uniform application across layers.

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

E. Unified Fine-tuning

Unified fine-tuning provides a cohesive framework that simplifies the integration of diverse fine-tuning methods into a unified architecture, ensuring consistency and efficiency in model adaptation and optimization. Unlike hybrid fine-tuning, which combines multiple PEFT methods, unified fine-tuning typically relies on a single PEFT method for adaptation.

AdaMix [66] leverages a mixture of adaptation module approaches to create a unified framework for fine-tuning. Motivated by sparsely-activated MoE [99], 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 adaptation module to learn multiple views for the given task but poses a challenge in deciding which adaptation module to use during inference. To this end, Adamix utilizes consistency regularization and adaptation module merging (i.e., average weights of all down- and up-projection matrices) to select the trained module. This

[5]LNFit is trained only on the variance vectors in the layer normalization module of the PLMs, inspired by [88] which trains only on the batch normalization module in convolutional neural networks.

approach ensures the same computational cost as using a single module during inference. Notably, adaptation modules in Adamix could be adapters like sequential adapter [10] or low-rank decomposition matrices like LoRA [12].

SparseAdapter [67] utilizes network pruning to construct a unified framework in which various PEFT methods, including the adapter family and LoRA [10], [12], [17], can be further pruned to improve parameter efficiency. SparseAdapter assigns a score z to all parameters of adapters and LoRA and prunes those with scores below a threshold z_s (the s-th lowest percentile of z). The target sparsity is denoted as s, and the score z can be computed using pruning methods such as random pruning, magnitude pruning [100], or SNIP [101], with SNIP-based SparseAdapter yielding the best results. SparseAdapter exhibits improved performance over full fine-tuning in the "Large-Sparse" setting, which involves larger bottleneck dimensions and higher sparsity ratios. Additionally, its network pruning technique is a plug-in method that can be applied to any adapter variants, such as MAM Adapter [17] and AdapterFusion [20]. The optimized parameters in SparseAdapter are represented as $\hat{W} = W \odot M$, where M is a binary mask matrix with $M = \mathbb{I}_{\{z \geq z_s\}}$ and $z = \text{score}(W)$.

ProPETL [68] introduces a single prototype network (e.g., adapter, prefix-tuning, and LoRA) shared across layers and tasks, while constructing different sub-networks for each layer using binary masks. Inspired by ALBERT [102], ProPETL leverages parameter sharing within the prototype network modules across the Transformer layers, enhancing parameter efficiency and reducing storage requirements. In ProPETL, binary masks $M \in \{0,1\}^n$ are introduced in each layer of the Transformer, where n is the number of parameters in a single PEFT module. Each binary mask corresponds to a specific sub-network of the shared prototype network. This approach allows layers to share parameters from the same prototype network while utilizing distinct sub-networks to capture meaningful semantic representations for different tasks. Task adaptation objective for the PLMs is formulated as:

$$\max_{\theta_{pro}, m_1, m_2, \cdots, m_L} \sum_{i=0}^{N} \log P(Y_i | X_i; \theta_{lm}, \theta_{sub}), \qquad (16)$$
$$\theta_{sub} = [\theta_{pro} \odot m_1, \theta_{pro} \odot m_2, \cdots, \theta_{pro} \odot m_L].$$

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

IV. Experiments

This section presents the experimental settings and results on performance, parameter, and memory efficiency, and discusses the performance-memory trade-offs of PEFT methods across models and tasks, as shown in Figure 4.

A. Experimental Settings

1) PLMs and Datasets: Encoder-only models RoBERTa (base and large) [3] are used to evaluate on the GLUE benchmark [103]; encoder-decoder models T5 (base and large) [5] and mT5 (small, base, and large) [104] are utilized to evaluate on the WMT16 Ro-En dataset[6]; and decoder-only models LLaMA (7B and 13B) [8] fine-tuned with the Alpaca dataset [105] are employed to evaluate on the MMLU benchmark [106]. Alpaca is an instruction dataset containing 52k samples. See more in Supplemental Material.

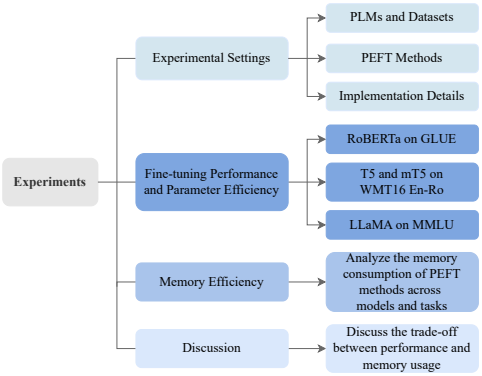


Fig. 4. Overview of Section IV: Experimental Settings, Fine-tuning Performance and Parameter Efficiency, Memory Efficiency, and Discussion on trade-off between performance and memory usage.

2) Implementation Details: We leverage the PEFT library[7] to implement prompt-tuning, prefix-tuning, $(IA)^3$, LoRA, and AdaLoRA. For BitFit, Child-Tuning, MAM adapter, QLoRA, AutoPEFT, and ProPETL, we experiment using their original code. While Adapter^S is implemented using code from MAM adapter. All 12 methods are evaluated on the GLUE benchmark, while a representative subset is applied to the T5, mT5, and LLaMA experiments. All experiments are conducted on NVIDIA A800. Detailed hyperparameter settings are provided in Supplemental Material.

B. Fine-tuning Performance and Parameter Efficiency

1) RoBERTa on GLUE: Table III compares full fine-tuning and 12 PEFT methods on the GLUE benchmark using RoBERTa-base and RoBERTa-large. Results are the mean and standard deviation computed over five runs for each dataset. Some PEFT methods (e.g., AutoPEFT and ProPETL) outperform full fine-tuning while significantly reducing trainable parameters. Specifically, it is observed:

- All PEFT methods reduce the number of trainable parameters while most PEFT methods achieve comparable or even improved performance with full fine-tuning, and a few PEFT methods achieve inferior performance to full fine-tuning.
- AutoPEFT$_{\text{RTE}}$ utilizes hyperparameters identified through NAS that deliver the best performance on RTE tasks, which are then applied to other NLU tasks. In contrast, AutoPEFT$_{\text{Task}}$ selects the optimal hyperparameters for each individual task through NAS. Overall, AutoPEFT$_{\text{Task}}$ outperforms AutoPEFT$_{\text{RTE}}$ across tasks.
- QLoRA-Int8 uses the fewest trainable parameters due to its lower LoRA rank and Int8 quantization, but it falls short of full fine-tuning performance.

[6]https://huggingface.co/datasets/wmt/wmt16/viewer/ro-en
[7]https://huggingface.co/docs/peft/index

TABLE III
FINE-TUNING ROBERTA MODELS ON THE GLUE BENCHMARK. THE NUMBER OF TRAINABLE PARAMETERS (# TPS) OF EACH METHOD IS PRESENTED, EXCLUDING CHILD-TUNING$_D$ DUE TO ITS RANDOMNESS DURING NETWORK PRUNING. WE BOLD THE BEST AND UNDERLINE THE WORST VALUES.

Model	PEFT	#TPs	CoLA	SST2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	Avg.
RoB_B	FT	124.6M	$55.65_{6.13}$	$93.67_{0.40}$	$87.79_{1.06}$/$91.23_{0.88}$	$90.83_{0.16}$/$90.60_{0.33}$	$90.77_{0.14}$/$87.78_{0.11}$	$87.08_{0.36}$	$92.26_{0.21}$	$75.38_{3.40}$	$84.18_{0.48}$/$84.21_{0.54}$
	Additive Fine-tuning										
	Adapter$^\text{S}$	7.41M	$63.42_{0.20}$	$94.48_{0.17}$	$\mathbf{89.90_{0.47}/92.71_{0.39}}$	$91.26_{0.09}$/$90.99_{0.06}$	$90.69_{0.37}$/$86.38_{0.59}$	$87.19_{0.11}$	$92.03_{0.17}$	$70.29_{9.77}$	$84.91_{1.24}$/$84.68_{1.22}$
	Prompt-tuning	0.61M	$\underline{38.99_{7.49}}$	$\underline{93.19_{0.42}}$	$\underline{73.48_{1.58}}$/$82.90_{0.74}$	$\underline{81.24_{1.52}}$/$81.98_{1.07}$	$\underline{82.07_{1.33}}$/$\underline{77.44_{1.06}}$	$\underline{78.39_{1.50}}$	$\underline{81.55_{0.94}}$	$\underline{57.04_{2.25}}$	$\underline{73.24_{1.08}}$/$\underline{73.94_{1.14}}$
	Prefix-tuning	0.96M	$55.30_{2.79}$	$93.81_{0.41}$	$85.88_{1.62}$/$90.11_{0.97}$	$88.45_{0.50}$/$88.32_{0.4}$	$87.10_{0.05}$/$82.79_{0.39}$	$85.10_{0.15}$	$90.84_{0.21}$	$62.17_{6.01}$	$81.08_{0.69}$/$81.06_{0.74}$
	$(\text{IA})^3$	0.66M	$59.34_{0.31}$	$93.76_{0.43}$	$86.62_{1.35}$/$90.42_{0.94}$	$90.43_{0.20}$/$90.40_{0.13}$	$88.51_{0.08}$/$84.78_{0.22}$	$84.85_{0.10}$	$91.19_{0.03}$	$73.79_{3.82}$	$83.56_{0.47}$/$83.57_{0.49}$
	Partial Fine-tuning										
	BitFit	0.69M	$59.65_{1.44}$	$94.27_{0.25}$	$88.97_{0.62}$/$92.04_{0.45}$	$90.33_{0.18}$/$90.26_{0.18}$	$86.74_{0.35}$/$82.83_{0.27}$	$83.81_{0.07}$	$90.23_{0.58}$	$75.96_{2.13}$	$83.74_{0.44}$/$83.63_{0.46}$
	Child-Tuning$_\text{D}$	-	$60.24_{0.56}$	$93.65_{0.28}$	$87.94_{0.89}$/$91.34_{0.66}$	$90.83_{0.22}$/$90.63_{0.20}$	$\mathbf{90.88_{0.09}/87.88_{0.11}}$	$\mathbf{87.29_{0.23}}$	$\mathbf{92.43_{0.35}}$	$76.10_{1.70}$	$84.92_{0.32}$/$84.95_{0.29}$
	Reparameterized Fine-tuning										
	LoRA	0.89M	$58.47_{2.02}$	$94.02_{0.33}$	$86.76_{1.54}$/$90.22_{1.09}$	$90.72_{0.24}$/$90.37_{0.18}$	$88.91_{0.09}$/$85.30_{0.11}$	$86.48_{0.11}$	$91.93_{0.12}$	$73.07_{0.91}$	$83.80_{0.34}$/$83.73_{0.29}$
	AdaLoRA	1.03M	$57.70_{2.17}$	$93.46_{0.64}$	$87.30_{0.88}$/$90.84_{0.52}$	$90.84_{0.10}$/$90.60_{0.13}$	$88.58_{0.07}$/$84.96_{0.15}$	$86.35_{0.07}$	$91.51_{0.16}$	$72.42_{2.52}$	$83.52_{0.19}$/$83.48_{0.21}$
	QLoRA Int8	**0.29M**	$55.39_{2.66}$	$93.53_{0.53}$	$86.13_{0.71}$/$89.96_{0.62}$	$89.17_{0.61}$/$89.37_{0.44}$	$88.87_{0.10}$/$85.26_{0.09}$	$86.39_{0.09}$	$91.89_{0.29}$	$72.20_{1.95}$	$82.94_{0.45}$/$83.00_{0.44}$
	Hybrid Fine-tuning										
	MAM Adapter	46.78M	$61.17_{0.79}$	$94.74_{0.17}$	$89.25_{0.98}$/$92.17_{0.76}$	$90.85_{0.13}$/$90.56_{0.16}$	$88.53_{0.20}$/$83.46_{0.27}$	$85.78_{0.19}$	$90.54_{0.27}$	$75.25_{2.22}$	$84.51_{0.36}$/$84.21_{0.35}$
	AutoPEFT$_\text{RTE}$	7.40M	$62.63_{1.75}$	$\mathbf{94.79_{0.49}}$	$88.28_{0.88}$/$91.50_{0.69}$	$90.19_{0.31}$/$90.05_{0.09}$	$89.78_{0.10}$/$86.56_{0.15}$	$86.80_{0.19}$	$91.91_{0.23}$	$74.95_{1.50}$	$84.92_{0.46}$/$84.90_{0.47}$
	AutoPEFT$_\text{Task}$	10.44M	$64.05_{0.92}$	$\mathbf{94.79_{0.49}}$	$89.85_{0.68}$/$92.61_{0.45}$	$90.51_{0.32}$/$90.54_{0.20}$	$89.92_{0.36}$/$87.30_{1.69}$	$86.84_{0.16}$	$91.99_{0.35}$	$76.54_{1.05}$	$\mathbf{85.56_{0.29}/85.58_{0.41}}$
	Unified Fine-tuning										
	ProPETL$_\text{Adapter}$	1.87M	$\mathbf{64.99_{2.32}}$	$93.92_{0.41}$	$87.84_{1.78}$/$91.30_{1.27}$	$91.21_{0.24}$/$90.85_{0.22}$	$88.26_{0.08}$/$84.41_{0.14}$	$85.23_{0.57}$	$91.98_{0.24}$	$\mathbf{77.27_{1.09}}$	$85.09_{0.47}$/$84.99_{0.41}$
	ProPETL$_\text{Prefix}$	10.49M	$64.28_{1.89}$	$94.27_{0.52}$	$87.94_{1.13}$/$91.27_{0.85}$	$90.66_{0.37}$/$90.96_{0.33}$	$87.87_{0.06}$/$83.93_{0.14}$	$85.18_{0.32}$	$90.56_{0.17}$	$72.81_{3.11}$	$84.25_{0.56}$/$84.16_{0.54}$
	ProPETL$_\text{LoRA}$	1.77M	$64.96_{1.52}$	$94.11_{0.33}$	$88.53_{1.02}$/$91.86_{0.75}$	$\mathbf{91.37_{0.12}/91.08_{0.13}}$	$88.43_{0.06}$/$84.83_{0.09}$	$86.18_{0.34}$	$91.46_{0.08}$	$71.22_{3.87}$	$84.53_{0.50}$/$84.46_{0.47}$
RoB_L	FT	355.3M	$61.24_{5.02}$	$95.32_{0.61}$	$89.07_{0.93}$/$92.09_{0.83}$	$91.95_{0.18}$/$91.91_{0.13}$	$87.97_{1.45}$/$83.94_{2.27}$	$89.47_{0.42}$	$93.53_{0.40}$	$76.75_{13.47}$	$85.66_{1.85}$/$85.53_{1.83}$
	Additive Fine-tuning										
	Adapter$^\text{S}$	19.77M	$67.13_{0.87}$	$\mathbf{96.33_{0.12}}$	$89.90_{0.24}$/$92.62_{0.21}$	$92.54_{0.07}$/$92.35_{0.08}$	$\mathbf{91.74_{0.66}/88.18_{0.90}}$	$90.32_{0.15}$	$94.06_{0.54}$	$73.57_{15.59}$	$86.95_{2.00}$/$86.82_{2.04}$
	Prompt-tuning	1.07M	$\underline{43.81_{7.33}}$	$94.79_{0.31}$	$\underline{73.63_{2.20}}$/$\underline{82.56_{1.55}}$	$\underline{79.54_{7.65}}$/$\underline{79.59_{7.62}}$	$\underline{75.97_{3.95}}$/$\underline{67.20_{7.71}}$	$\underline{77.47_{12.59}}$	$\underline{76.70_{8.84}}$	$\underline{53.50_{3.98}}$	$\underline{71.93_{3.42}}$/$\underline{71.95_{3.14}}$
	Prefix-tuning	2.03M	$62.13_{2.08}$	$95.46_{0.24}$	$87.25_{0.63}$/$90.87_{0.46}$	$91.26_{0.27}$/$91.15_{0.17}$	$87.91_{0.24}$/$84.03_{0.35}$	$89.19_{0.10}$	$93.56_{0.26}$	$74.73_{5.99}$	$85.19_{0.85}$/$85.14_{0.87}$
	$(\text{IA})^3$	1.22M	$63.34_{4.25}$	$95.37_{0.25}$	$87.25_{1.29}$/$90.81_{0.77}$	$92.02_{0.05}$/$91.88_{0.08}$	$90.06_{0.08}$/$86.68_{0.13}$	$89.58_{0.10}$	$94.07_{0.12}$	$83.39_{1.42}$	$86.89_{0.62}$/$86.89_{0.58}$
	Partial Fine-tuning										
	BitFit	1.32M	$68.13_{1.38}$	$95.99_{0.27}$	$90.25_{0.76}$/$92.81_{0.67}$	$91.85_{0.10}$/$91.69_{0.10}$	$88.30_{0.26}$/$84.46_{0.54}$	$88.93_{0.18}$	$93.41_{0.32}$	$84.91_{1.87}$	$87.72_{0.23}$/$87.54_{0.23}$
	Child-Tuning$_\text{D}$	-	$65.34_{2.00}$	$\underline{94.20_{1.84}}$	$85.88_{9.79}$/$90.65_{5.28}$	$92.15_{0.31}$/$91.99_{0.26}$	$80.13_{15.47}$/$\underline{53.10_{48.48}}$	$\underline{56.34_{31.10}}$	$\underline{67.60_{23.86}}$	$77.61_{17.04}$	$77.41_{4.59}$/$74.60_{7.72}$
	Reparameterized Fine-tuning										
	LoRA	1.84M	$63.40_{3.86}$	$95.78_{0.58}$	$88.04_{0.72}$/$90.07_{1.29}$	$92.02_{0.21}$/$91.74_{0.18}$	$90.12_{0.10}$/$86.88_{0.15}$	$\mathbf{90.62_{0.11}}$	$\mathbf{94.74_{0.17}}$	$82.24_{1.58}$	$87.12_{0.44}$/$86.93_{0.55}$
	AdaLoRA	2.23M	$62.11_{4.50}$	$95.44_{0.34}$	$88.83_{0.89}$/$92.03_{0.49}$	$92.13_{0.09}$/$91.91_{0.12}$	$89.55_{0.04}$/$86.22_{0.08}$	$90.07_{1.17}$	$94.62_{0.13}$	$79.42_{2.81}$	$86.52_{0.37}$/$86.48_{0.39}$
	QLoRA Int8	**0.79M**	$61.54_{4.07}$	$95.48_{0.48}$	$87.89_{1.34}$/$91.27_{1.14}$	$91.69_{0.17}$/$91.44_{0.17}$	$90.01_{0.12}$/$86.82_{0.17}$	$90.43_{0.18}$	$94.18_{0.45}$	$82.46_{2.05}$	$86.71_{0.63}$/$86.70_{0.60}$
	Hybrid Fine-tuning										
	MAM Adapter	122.2M	$67.57_{0.22}$	$96.03_{0.16}$	$90.30_{0.65}$/$92.82_{0.44}$	$92.55_{0.19}$/$92.29_{0.16}$	$90.88_{0.31}$/$86.80_{0.48}$	$90.12_{0.34}$	$94.02_{0.40}$	$\mathbf{86.94_{0.63}}$	$88.55_{0.11}$/$88.32_{0.12}$
	AutoPEFT$_\text{RTE}$	9.60M	$66.47_{1.13}$	$96.33_{0.24}$	$89.51_{0.50}$/$92.28_{0.42}$	$92.14_{0.15}$/$91.91_{0.14}$	$90.88_{0.05}$/$88.01_{0.08}$	$90.48_{0.15}$	$94.41_{0.17}$	$84.48_{1.44}$	$88.09_{0.18}$/$88.05_{0.17}$
	AutoPEFT$_\text{Task}$	9.60M	$67.43_{1.36}$	$96.33_{0.24}$	$\mathbf{91.28_{0.82}/93.61_{0.58}}$	$92.28_{0.18}$/$91.98_{0.16}$	$90.96_{0.19}$/$88.10_{0.22}$	$90.53_{0.13}$	$94.46_{0.07}$	$86.28_{1.44}$	$\mathbf{88.69_{0.38}/88.59_{0.36}}$
	Unified Fine-tuning										
	ProPETL$_\text{Adapter}$	5.40M	$69.22_{2.60}$	$96.01_{0.15}$	$88.73_{0.77}$/$91.84_{0.56}$	$92.55_{0.22}$/$92.25_{0.24}$	$89.52_{0.46}$/$86.16_{0.82}$	$90.28_{0.40}$	$94.63_{0.09}$	$71.37_{16.25}$	$86.54_{2.27}$/$86.47_{2.23}$
	ProPETL$_\text{Prefix}$	26.85M	$68.06_{2.32}$	$95.85_{0.32}$	$87.45_{1.54}$/$90.89_{1.08}$	$92.31_{0.23}$/$92.46_{0.25}$	$83.92_{11.72}$/$75.70_{22.43}$	$89.84_{0.15}$	$93.89_{0.41}$	$81.01_{3.08}$	$86.54_{1.36}$/$85.96_{2.62}$
	ProPETL$_\text{LoRA}$	4.19M	$\mathbf{70.64_{1.97}}$	$95.96_{0.34}$	$88.53_{0.89}$/$91.63_{0.77}$	$\mathbf{92.79_{0.41}/92.46_{0.39}}$	$89.86_{0.13}$/$86.56_{0.21}$	$90.51_{0.21}$	$94.45_{0.40}$	$82.88_{2.56}$	$88.20_{0.52}$/$88.14_{0.52}$

- MAM Adapter and AutoPEFT are hybrid PEFT methods that combine multiple components manually and automatically, respectively. AutoPEFT generally outperforms the manually designed MAM Adapter while using fewer parameters; however, its NAS for hyperparameter tuning incurs higher computational overhead.
- $(\text{IA})^3$, BitFit, LoRA, AdaLoRA, and QLoRA underperform compared to full fine-tuning on RoBERTa-base, but outperform it on RoBERTa-large.
- Prompt-tuning underperforms compared to other PEFT methods across GLUE tasks, with performance gap particularly pronounced on CoLA and RTE. These two tasks are both low-resource and linguistically complex, requiring syntactic sensitivity (CoLA) or logical inference (RTE), which prompt-tuning struggles to capture effectively due to its limited parameter updates.
- Child-Tuning$_D$ performs well when fine-tuning RoBERTa-base on the GLUE benchmark, even surpassing full fine-tuning. However, when applied to RoBERTa-large, it exhibits large standard deviations across tasks such as QQP, MNLI, QNLI, and RTE, indicating high sensitivity to random seeds.

2) T5 and mT5 on WMT16 Ro-En: Table IV compares full fine-tuning and 5 PEFT methods on the WMT16 Ro-En dataset using T5 (base, large), and mT5 (small, base, large). While PEFT methods substantially reduce trainable parameters, their effectiveness varies significantly depending on the model architecture and size. It is observed:

- The BLEU scores for T5 models with full fine-tuning consistently surpass those of all mT5 variants. Notably, T5-base (28.16) outperforms even the significantly larger mT5-large (26.47). This performance gap can be attributed to the fact that T5 is specifically trained on English text, whereas mT5 is trained on multi-lingual datasets, which may result in less optimal performance for specific language tasks involving English translation.
- For T5-base, both full fine-tuning and PEFT methods demonstrate robust stability with minimal variance across different random seeds. PEFT methods yield results competitive with full fine-tuning across T5-base and T5-large, validating their effectiveness despite having fewer trainable parameters.
- Full fine-tuning consistently outperforms 5 PEFT methods on mT5 models. This gap is particularly pronounced on smaller architectures (mT5-small and mT5-base). This suggests that PEFT methods, which update only a small number of parameters, lack sufficient capacity to fully adapt the compressed multi-lingual representations of smaller models to a specific

TABLE IV
FINE-TUNING T5-BASE/LARGE, AND MT5-SMALL/BASE/LARGE ON THE WMT16 RO-EN AND EVALUATING PERFORMANCE USING BLEU SCORE. HIGHER BLEU SCORE INDICATES BETTER QUALITY OF TRANSLATION.

PEFT Method	$T5_B$		$T5_L$		$mT5_S$		$mT5_B$		$mT5_L$	
	# TPs	BLEU	# TPs	BLEU	# TPs	BLEU	# TPs	BLEU	# TPs	BLEU
FT	222.9M	**$28.16_{0.10}$**	737.7M	**28.27**	300.2M	**23.38**	582.4M	**26.41**	1229.6M	**26.47**
Prompt-tuning	**0.03M**	$27.59_{0.10}$	**0.04M**	28.16	**0.02M**	$0.0006^{\dagger}$	**0.03M**	$0.12^{\dagger}$	**0.04M**	$0.0022^{\dagger}$
Prefix-tuning	0.37M	$27.35_{0.08}$	0.98M	26.36	0.86M	1.62	1.29M	6.86	3.44M	13.73
$(IA)^3$	0.07M	$27.84_{0.04}$	0.19M	28.12	0.02M	2.27	0.07M	12.42	0.15M	20.33
LoRA	0.88M	$27.93_{0.08}$	2.36M	28.12	0.69M	12.86	1.77M	20.74	4.72M	25.40
QLoRA	0.44M	$27.64_{0.06}$	1.18M	28.11	0.34M	12.74	0.88M	20.49	2.36M	24.95

† The BLEU score of prompt-tuning for mT5-small/base/large is low and potentially unstable in the multi-lingual setting.

downstream task. See Supplemental Material for details.

- Prompt-tuning achieves moderate performance on T5-base and T5-large but performs poorly across mT5 model sizes (small, base, and large), yielding near-zero BLEU scores. Additional experiments (see Supplemental Material) confirm that this is not due to hyperparameters. Similar findings have also been observed in some other recent studies [107], [108], which have reported similar underperformance of prompt-tuning on mT5 across cross-lingual tasks such as named entity recognition (NER), part-of-speech (POS) tagging, and question answering (QA). These results suggest that prompt-tuning may be less effective for mT5 in tasks beyond the translation task, potentially due to limited adaptation capacity.
- There is a distinct positive correlation between model size and PEFT efficacy for mT5. The performance gap between PEFT methods (specifically LoRA, QLoRA and $(IA)^3$) and full fine-tuning narrows significantly as the model size increases. For instance, LoRA on mT5-small lags behind full fine-tuning by 10.52, this gap shrinks to just 1.07 on mT5-large. This indicates that larger multi-lingual models possess sufficient redundant capacity to be effectively adapted via PEFT methods.
- Despite the performance gap with full fine-tuning, LoRA and QLoRA significantly outperform prompt-tuning, prefix-tuning, and $(IA)^3$ across mT5 models. We attribute this is likely because LoRA directly modifies the weight updates, allowing for a more stable optimization and better preservation of features. In contrast, prompt-tuning and prefix-tuning rely on indirect "guidance" via virtual tokens, while $(IA)^3$ uses simple activation rescaling; both appear to lack the sufficient capacity to disentangle the complex multi-lingual representations required for high-quality translation.

3) LLaMA on MMLU: As illustrated in Table V, full fine-tuning of LLaMA-7B and LLaMA-13B produces higher 5-shot MMLU test accuracy than the three PEFT methods. $(IA)^3$, LoRA, and QLoRA methods all greatly reduce the number of trainable parameters with $(IA)^3$ performing best. Though $(IA)^3$ uses only 0.02% of the full fine-tuning parameters, its performance is 2-4% lower.LoRA and QLoRA use approximately 2% of the full fine-tuning parameters and achieve accuracy about 2% lower. Notably, QLoRA uses only half the trainable parameters of LoRA while delivering comparable performance. This efficiency in QLoRA is largely attributed to NF4 quantization, which enables substantial parameter reduction without sacrificing accuracy. Additionally, 5-shot MMLU accuracy is sensitive to the learning rate, with fluctuations observed on the dev set as evaluation steps increase. More details are provided in the Supplemental Material.

C. Memory Efficiency

As presented in Table VI, peak GPU memory usage during fine-tuning varies by models and methods. Generally, GPU memory consumption correlates positively with model size across RoBERTa, T5, mT5, and LLaMA. Most PEFT methods help reduce GPU memory usage compared to full fine-tuning.

For RoBERTa, most PEFT methods substantially reduce memory usage. QLoRA Int8 achieves a peak GPU memory of 2.07GB on RoBERTa-base, a 61.5% reduction compared to full fine-tuning (5.38GB), primarily due to 8-bit integer quantization. However, Adapter^S, Child-Tuning_D, MAM Adapter and AutoPEFT consume more memory than full fine-tuning due to the overhead of intermediate computations and gradient caching. **Adapter^S** introduces adapter modules at each Transformer layer, increasing storage needs for hidden states and gradients. **Child-Tuning_D** employs child networks and applies a binary mask matrix, which increases memory usage for storing and applying the mask matrix during backpropagation, and additional gradient calculations for masked parameters. **MAM Adapter** combines prefix-tuning and scaled parallel adapters, increasing memory usage through cached activations for its extended attention mechanism and gradients for adapter weights. **AutoPEFT** is particularly memory-intensive as it integrates multiple PEFT methods and additional operations for layer selection and optimization, necessitating the caching multiple intermediate representations and gradients.

For T5 and mT5 models, 5 PEFT methods consistently reduce memory usage compared to full fine-tuning, with QLoRA again demonstrating the greatest efficiency. On T5-base, full fine-tuning requires 25.17GB, while QLoRA reduces it to 5.69GB. This efficiency is achieved through NF4 quantization, which minimizes memory required for model parameters; double quantization, which further compresses the quantized weights; and paged optimizers, which offload infrequently accessed data, reducing the GPU footprint. mT5 models, which utilize FP32 precision for fine-tuning, exhibits significantly higher GPU memory usage due to the need to store high-precision parameters and intermediate activations. Even with PEFT methods, memory usage remains high, underscoring the

TABLE V
COMPARISON OF AVERAGE 5-SHOT MMLU TEST ACCURACY FOR LLAMA-7B AND LLAMA-13B MODELS USING ALPACA WITH PEFT METHODS. THE HIGHER MMLU ACCURACY, THE BETTER. # APS REFERS TO THE TOTAL MODEL PARAMETERS.

Model	PEFT Method	# TPs	# APs	% Params	5-shot MMLU Accuracy
LLaMA_{7B}	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
$\text{LLaMA}_{\text{13B}}$	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

TABLE VI
PEAK GPU MEMORY USAGE (GB) DURING FINE-TUNING OF ROBERTA, T5, MT5, AND LLAMA WITH VARIOUS PEFT METHODS.

PEFT	RoB_{B}	RoB_{L}	T5_{B}	T5_{L}	mT5_{S}	mT5_{B}	mT5_{L}	LLaMA_{7B}	$\text{LLaMA}_{\text{13B}}$
FT	5.38	11.96	25.17	30.17	43.92	63.71	63.97	169.36	287.79
Additive Fine-tuning									
Adapter^S	15.29	37.17	-	-	-	-	-	-	-
Prompt-tuning	3.84	7.98	19.35	24.24	42.60	54.16	55.38	-	-
Prefix-tuning	3.56	7.58	13.84	18.79	41.08	47.45	48.36	-	-
$(IA)^3$	3.83	9.00	21.26	25.71	42.51	54.80	54.69	128.57	191.24
Partial Fine-tuning									
BitFit	3.27	7.50	-	-	-	-	-	-	-
Child-Tuning_D	6.02	13.67	-	-	-	-	-	-	-
Reparameterized Fine-tuning									
LoRA	3.59	7.50	19.43	23.77	42.71	42.90	54.76	124.82	174.24
AdaLoRA	3.57	7.43	-	-	-	-	-	-	-
QLoRA	**2.07**	**2.78**	**5.69**	**7.65**	**38.41**	**40.29**	**37.80**	**56.46**	**66.60**
Hybrid Fine-tuning									
MAM Adapter	15.35	37.82	-	-	-	-	-	-	-
AutoPEFT	17.11	19.82	-	-	-	-	-	-	-
Unified Fine-tuning									
$\text{ProPETL}_{\text{Adapter}}$	2.77	6.52	-	-	-	-	-	-	-
$\text{ProPETL}_{\text{Prefix}}$	2.86	7.82	-	-	-	-	-	-	-
$\text{ProPETL}_{\text{LoRA}}$	2.78	6.54	-	-	-	-	-	-	-

influence of precision settings on GPU memory consumption. Although mT5-large possesses significantly more trainable parameters, it requires comparable or less memory than mT5-base when using $(IA)^3$ and QLoRA. This phenomenon occurs because the memory savings from halving the batch size (from 64 to 32) effectively offset the increased memory costs of the larger model architecture.

For LLaMA models, PEFT methods achieve substantial memory savings, especially for larger models. For instance, full fine-tuning of LLaMA-7B-Alpaca requires 169.36GB, while QLoRA reduces this to 56.46GB, a 66.6% reduction. On LLaMA-13B-Alpaca, full fine-tuning requires 287.79GB, but QLoRA reduces this down to 66.60GB, a 76.9% reduction. The efficiency gains of QLoRA are particularly pronounced for larger models due to the combination of quantization and efficient memory management techniques. Moreover, PEFT methods such as LoRA and $(IA)^3$ also achieve a more significant memory reduction on LLaMA-13B compared to LLaMA-7B, suggesting that the benefits of PEFT methods increase with model size. Larger models see greater advantages from reduced parameter storage and optimized cache memory, making PEFT methods increasingly effective as model size grows.

D. Discussion

Experimental results in Tables III, IV, V, and VI reveal a critical trade-off between performance and memory consumption across different PEFT methods. Sequential Adapter, MAM Adapter and AutoPEFT outperform full fine-tuning with RoBERTa but require higher GPU memory. Among these, AutoPEFT achieves the best overall performance but also demands the greatest memory usage, making it suitable for users prioritizing performance with sufficient computational resources. ProPETL offers slightly lower performance than AutoPEFT but excels in the GLUE benchmark with RoBERTa while consuming less memory, striking a good trade-off between performance and memory usage. BitFit and AdaLoRA achieve performance comparable to full fine-tuning with RoBERTa while requiring less memory, offering a balanced performance-memory trade-off. Notably, $(IA)^3$, LoRA, and QLoRA effectively balance performance and memory efficiency, making them ideal options across models and tasks. QLoRA stands out as the most memory-efficient method, achieving performance slightly below full fine-tuning, making it particularly suitable for resource-constrained scenarios. Prompt-tuning and prefix-tuning reduce memory usage but ex-

hibit inconsistent performance across models and tasks. They underperform on GLUE with RoBERTa, perform comparably on WMT16 Ro-En with T5 models, but show poor results with mT5 models, making them less reliable.

V. Applications

A. Multi-task Learning

Multi-task learning leveraging shared information across related tasks to improve performance. PEFT methods like adapters, prompt-tuning, and LoRA enhance this process using pluggable modules. Adapters [20], [22], [23], [77] employ task-specific adapters to learn information for more robust transfer learning, while prompt-tuning [27], [28], [29] transfers knowledge by initializing target prompts from source tasks or learning shared prompts. Similarly, LoRA modules can be composed to transfer knowledge [58], [59]. L-LoRA [60] enhances multi-task fusion by preventing negative inference between task-specific representations. Additionally, [96] utilizes arithmetic operators to merge parameters of various PEFT methods trained on different tasks for multi-task learning.

B. Cross-Lingual Transfer

Cross-lingual transfer moves knowledge between languages, often utilizing the modular design of adapters. Bapna and Firat [109] usesequential adapters [10] to maintain translation performance on high-resource languages, while Artetxe et al. [110] apply them to transfer monolingual models to unseen languages. More complex architectures like MAD-X [77], [111] combines language, task, and invertible adapters to handle vocabulary mismatches, while MAD-G [112] generates language adapters from typological representations to share linguistic knowledge. Furthermore, LT-SFT [39] uses sparse fine-tuning for cross-lingual transfer, and BAD-X [113] trains bilingual adapters for zero-shot transfer.

C. Backdoor Attacks and Defense

Backdoor attacks, which cause models to misclassify inputs containing specific triggers while maintaining normal performance on benign data, pose a major risk to PLMs [114]. Leveraging the vulnerability of pretrained weights, Gu et al. [115] use PEFT methods to construct backdoor attacks by injecting triggers directly into PEFT modules. However, Zhu et al. [116] 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 [117] 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

Many approaches [17], [36], [61], [62], [63], [64], [65] combine multiple PEFT methods to leverage their unique strengths for improved performance. However, existing research primarily focuses on combining adapters, LoRA, prefix-tuning, and BitFit, leaving room to explore a broader range of techniques. Inspired by NAS algorithms, some studies [63], [64] employ optimization to discover effective neural network architectures for configuring these methods. Nonetheless, there remains considerable scope to explore alternative optimization strategies for searching architectures and assigning specific combinations of PEFT modules across different layers. Thus, a promising direction is to investigate strategies that combine multiple PEFT methods to boost performance while minimizing the number of trainable parameters and memory usage.

B. LoRA-derived PEFT Methods

As illustrated in Fig. 1, a multitude of LoRA-based PEFT methods have recently emerged. These variants enhance the original LoRA framework by integrating adaptive rank adjustment, unstructured pruning, weight quantization, and multi-task learning. Particular emphasis should be placed on pruning and quantization techniques. For instance, pruning is utilized in AdaLoRA [46] for rank adjustment, and in LoRAPrune [49] to compress both pretrained and LoRA weights. These techniques effectively reduce trainable parameters and computational requirements of PLMs (especially LLMs), enhancing their utility and scalability across downstream tasks. Future research could explore the conjunction of these techniques with LoRA to unlock synergistic benefits.

C. Developing PEFT Library

Various PEFT methods have been developed, but implementing them from scratch can be challenging. To simplify this process, the PEFT library[8] and AdapterHub[9] have been developed. These libraries integrate commonly used PEFT methods such as prefix-tuning, LoRA, and AdaLoRA, enabling users to apply them with just a few lines of code. Both libraries also offer a range of examples illustrating how to fine-tune various PLMs and LLMs for downstream tasks using these methods. However, not all PEFT methods are currently supported. Expanding the integration of additional methods in these libraries represents a promising direction for enhancing the accessibility and utility of PEFT methods.

D. Explainability of PEFT Methods

Though numerous PEFT methods have been proposed, there is a lack of comprehensive studies explaining why they achieve comparable performance with reduced trainable parameters. Work from [42] unifies PEFT methods under the concept of sparse fine-tuned models, providing a theoretical analysis demonstrating that sparsity can serve as a regularization technique to control stability. Meanwhile, [118] analyzes the express power of LoRA for fully connected neural networks and the 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 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.

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

E. PEFT in Computer Vision and Multi-modal Learning

Though PEFT methods have been extensively studied in NLP, their application in computer vision and multi-modal learning shows great potential for further exploration. The sequential adapter, initially inspired by multi-domain image classification [78], has spurred rapid progress in PEFT methods for PLMs. Recent studies have extended PEFT techniques to computer vision [119], [120] and multi-modal tasks involving language-image and image-audio inputs [121], [122], building upon PEFT methods in NLP [10], [12], [61]. Despite this progress, substantial research opportunities remain, particularly in enabling cross-modality transfer. By fine-tuning pretrained models using PEFT, knowledge acquired from one modality can be effectively transferred to another, enhancing performance in multi-modal tasks.

F. Dequantization of Quantization-aware PEFT Methods

Quantization-aware PEFT methods, such as QLoRA and LOFTQ, significantly reduce memory usage by using 4-bit or 2-bit weights. However, they still rely on dequantization, which involves converting these low-precision weights and activations back to higher precision (e.g., FP16/FP32) during training and inference. While this dequantization process is necessary to maintain model accuracy, it introduces additional memory and computational overhead, thereby diminishing the overall efficiency gains from quantization. Recent study [123] have demonstrated that second-order dequantization, where both weights and scales of groups are dequantized, can account for over 50% of execution time. This underscores the need for more efficient dequantization methods. Future research could investigate asynchronous or parallel dequantization techniques to overlap computation and memory access, thereby reducing latency. Additionally, selective dequantization, in which only the most sensitive layers (e.g., attention layers) are reverted to higher precision, is another promising direction.

G. Integration with Advanced Learning Paradigms

While PEFT has been widely applied to supervised fine-tuning, integrating it with advanced paradigms like reinforcement learning (RL) and self-supervised learning (SSL) offers a promising direction. For example, PE-RLHF [124] employs LoRA adapters for policy and value models within the RL loop. Recent works involving DeepSeek and Qwen further validate this synergy. Tina [125] incorporates LoRA into a Group Relative Policy Optimization (GRPO) framework [126], central to DeepSeek-R1, for efficient performance, while LoRA-PAR [127] uses RL for dynamic parameter allocation in LLMs like Qwen to enhance complex reasoning. In the SSL setting, ExPLoRA [128] introduces a PEFT-based continual self-supervised strategy that applies LoRA to general-purpose models on domain-specific data for efficient domain adaptation. However, most existing methods simply stack adapters onto standard RL or SSL pipelines without addressing the distinctive challenges such as the instability of RL training under low-rank constraints or the limited capacity of adapters to capture rich representations from massive unlabeled data. Future work could explore architecture-aware PEFT designs tailored to specific dynamic of RL and the representation learning objectives of SSL.

VII. Conclusions

This paper presents a comprehensive overview of PEFT methods for PLMs, aiming to inspire further advancements in this field. We categorize PEFT methods in NLP into five classes and provide an in-depth analysis of their mechanisms and unique characteristics. To evaluate their effectiveness, we fine-tune encoder-only RoBERTa, encoder-decoder T5 and mT5, and decoder-only LLaMA on a variety of downstream tasks using several representative PEFT methods. Experimental results reveal that most PEFT methods significantly improve parameter efficiency while achieving performance comparable to, or even surpassing, full fine-tuning. In addition, these methods effectively reduce memory footprints, with QLoRA drastically cutting computational memory requirements and alleviating memory challenges during LLM fine-tuning. Notably, $(IA)^3$, LoRA, and QLoRA achieve a balanced trade-off between performance and memory efficiency across different models and tasks. We also explore common applications of PEFT methods and propose future research directions. As LLMs continue to advance, there is a growing need for PEFT methods that can further minimize computational and memory demands during fine-tuning. This survey aims to provide a clear and comprehensive foundation for future research in this area.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Adv. Neural Inf. Process. Syst.*, vol. 30, pp. 6000–6010, 2017.

[2] 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.

[3] 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.

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

[5] 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.

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

[7] 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.

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

[9] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[10] 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.

[11] 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.

[12] 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.

[13] N. Ding, Y. Qin, G. Yang, F. Wei, Z. Yang, Y. Su, S. Hu, Y. Chen, C.-M. Chan, W. Chen, J. Yi, W. Zhao, X. Wang, Z. Liu, H.-T. Zheng, J. Chen, Y. Liu, J. Tang, J. Li, and M. Sun, "Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models," *arXiv preprint arXiv:2203.06904*, 2022.

[14] 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.

[15] 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.

[16] T. Lei, J. Bai, S. Brahma, J. Ainslie, K. Lee, Y. Zhou, N. Du, V. Zhao, Y. Wu, B. Li, Y. Zhang, and M.-W. Chang, "Conditional adapters: Parameter-efficient transfer learning with fast inference," *Advances in Neural Information Processing Systems*, vol. 36, pp. 8152–8172, 2023.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

[29] 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.

[30] Y.-L. Sung, J. Cho, and M. Bansal, "LST: Ladder side-tuning for parameter and memory efficient transfer learning," in *Adv. Neural Inf. Process. Syst.*, 2022, pp. 12 991–13 005.

[31] 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 *Adv. Neural Inf. Process. Syst.*, 2022, pp. 1950–1965.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] Y.-L. Sung, V. Nair, and C. Raffel, "Training neural networks with fixed sparse masks," in *Adv. Neural Inf. Process. Syst.*, 2021, pp. 24 193–24 205.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

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

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

[51] 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.

[52] 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.

[53] Y. Chai, J. Gkountouras, G. G. Ko, D. Brooks, and G.-Y. Wei, "Int2. 1: Towards fine-tunable quantized large language models with error correction through low-rank adaptation," *arXiv preprint arXiv:2306.08162*, 2023.

[54] H. Guo, P. Greengard, E. P. Xing, and Y. Kim, "Lq-lora: Low-rank plus quantized matrix decomposition for efficient language model finetuning," *arXiv preprint arXiv:2311.12023*, 2023.

[55] 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.

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

[57] 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.

[58] 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.

[59] 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.

[60] 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.

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

[62] 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.

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

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

[65] 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.

[66] 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.

[67] 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.

[68] 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.

[69] 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.

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

[71] L. Xu and W. Wang, "Improving aspect-based sentiment analysis with contrastive learning," *Nat. Lang. Process. J.*, vol. 3, p. 100009, 2023.

[72] 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 *Proc. Web Conf.*, 2020, pp. 2934–2940.

[73] 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.

[74] 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.

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

[76] 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.

[77] 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.

[78] S.-A. Rebuffi, H. Bilen, and A. Vedaldi, "Learning multiple visual domains with residual adapters," *Adv. Neural Inf. Process. Syst.*, vol. 30, pp. 506–516, 2017.

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

[80] 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.

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

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

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

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

[85] 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.

[86] 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.

[87] 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.

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

[89] 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.

[90] 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.

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

[92] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," in *Proc. Int. Conf. Learn. Representations*, 2023.

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

[94] 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.

[95] 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.

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

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

[98] 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.

[99] 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.

[100] 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.

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

[102] 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.

[103] A. Wang, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[104] L. Xue, N. Constant, A. Roberts, M. Kale, R. Al-Rfou, A. Siddhant, A. Barua, and C. Raffel, "mT5: A massively multilingual pre-trained text-to-text transformer," in *Proc. 2021 Conf. North Amer. Chapter Assoc. Comput. Linguist.: Hum. Lang. Technol.*, 2021, pp. 483–498.

[105] 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.
[106] 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.
[107] B. Ma, E. Nie, S. Yuan, H. Schmid, M. Färber, F. Kreuter, and H. Schuetze, "ToPro: Token-level prompt decomposition for cross-lingual sequence labeling tasks," in *Proc. 18th Conf. Eur. Chapter Assoc. Comput. Linguist.*, 2024, pp. 2685–2702.
[108] N. Chirkova and V. Nikoulina, "Key ingredients for effective zero-shot cross-lingual knowledge transfer in generative tasks," in *Proc. 2024 Conf. North Am. Chapter Assoc. Comput. Linguist: Hum. Lang. Technol.*, 2024, pp. 7222–7238.
[109] 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.
[110] 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.
[111] 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.
[112] 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.
[113] 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.
[114] 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.
[115] 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.
[116] 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 *Adv. Neural Inf. Process. Syst.*, 2022, pp. 1086–1099.
[117] L. Hong and T. Wang, "Fewer is more: Trojan attacks on parameter-efficient fine-tuning," *arXiv preprint arXiv:2310.00648*, 2023.
[118] Y. Zeng and K. Lee, "The expressive power of low-rank adaptation," *arXiv preprint arXiv:2310.17513*, 2023.
[119] 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.
[120] 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.
[121] 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.
[122] J. Pan, Z. Lin, X. Zhu, J. Shao, and H. Li, "St-adapter: Parameter-efficient image-to-video transfer learning," *Adv. Neural Inf. Process. Syst.*, vol. 35, pp. 26 462–26 477, 2022.
[123] J. Li, J. Xu, S. Li, S. Huang, J. Liu, Y. Lian, and G. Dai, "Fast and efficient 2-bit llm inference on gpu: 2/4/16-bit in a weight matrix with asynchronous dequantization," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2024, pp. 1–9.
[124] H. Sidahmed, S. Phatale, A. Hutcheson, Z. Lin, Z. Chen, Z. Yu, J. Jin, S. Chaudhary, R. Komarytsia, C. Ahlheim, Y. Zhu, B. Li, S. Ganesh, B. Byrne, J. Hoffmann, H. Mansoor, W. Li, A. Rastogi, and L. Dixon, "Parameter efficient reinforcement learning from human feedback," *arXiv preprint arXiv:2403.10704*, 2024.
[125] S. Wang, J. Asilis, Ö. F. Akgül, E. B. Bilgin, O. Liu, and W. Neiswanger, "Tina: Tiny reasoning models via lora," *arXiv preprint arXiv:2504.15777*, 2025.
[126] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. Li, Y. Wu, and G. Daya, "Deepseekmath: Pushing the limits of mathematical reasoning in open language models," *arXiv preprint arXiv:2402.03300*, 2024.
[127] Y. Huang, B. Li, K. Tang, and M. Chen, "LoRA-PAR: A flexible dual-system LoRA partitioning approach to efficient LLM fine-tuning," in *Findings Assoc. Comput. Linguist.: EMNLP 2025*, 2025, pp. 13 693–13 704.
[128] S. Khanna, M. Irgau, D. B. Lobell, and S. Ermon, "ExPLoRA: Parameter-efficient extended pre-training to adapt vision transformers under domain shifts," in *Int. Conf. Mach. Learn.*, 2025.

Lingling Xu (Student Member, IEEE) received the Ph.D. degree in computer science and technology from Hong Kong Metropolitan University, and the Master degree in Mathematics from Shandong University. She is currently a postdoctoral fellow at the School of Data Science, Lingnan university. Her research interests include parameter-efficient fine-tuning, contrastive learning, representation learning, and aspect-based sentiment analysis.

Haoran Xie (Senior Member, IEEE) received the Ph.D. degree in Computer Science from City University of Hong Kong and an Ed.D degree in Language Learning from the University of Bristol. He is currently a Professor and the Person-in-Charge at the Division of Artificial Intelligence, Director of LEO Dr David P. Chan Institute of Data Science, and Associate Dean of the School of Data Science, Lingnan University, Hong Kong. His research interests include natural language processing, large language models, and AI in education. He has published 468 research publications, including 283 journal articles. He is the Editor-in-Chief of Natural Language Processing Journal, Computers & Education: Artificial Intelligence, and Computers & Education: X Reality. He has been selected as the World's Top 2% Scientists by Stanford University.

S. 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 89, 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.