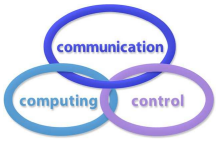
INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
Online ISSN 1841-9844, ISSN-L 1841-9836, Volume: 20, Issue: 6, Month: December, Year: 2025
Article Number: 6928, https://doi.org/10.15837/ijccc.2025.6.6928



CCC Publications

Late Adapter Tuning: A Cost-Effective Approach to Parameter-Efficient Fine-Tuning for Large Language Models

Z. Gao, R. Li, Y. Fan, M. Liao, X. Song

Zhengjie Gao*
School of Electronic and Information Engineering
Geely University of China, Chengdu 641423, Sichuan, China
*Corresponding author: gaozhengjie@guc.edu.cn

Rongcheng Li
School of Computer Science
Chengdu University of Information and Technology
Chengdu 610225, Sichuan, China
lrch137@163.com

Yuxin Fan
School of Electronic and Information Engineering
Geely University of China, Chengdu 641423, Sichuan, China
lu.alex275104@gmail.com

Min Liao
School of Electronic and Information Engineering
Geely University of China, Chengdu 641423, Sichuan, China
liaomin@guc.edu.cn

Xinyu Song
School of Electronic and Information Engineering
Geely University of China, Chengdu 641423, Sichuan, China
songxinyu@guc.edu.cn

Abstract

Fine-tuning large language models (LLMs) is computationally prohibitive for individual researchers, especially in resource-constrained scenarios. While parameter-efficient fine-tuning (PEFT) methods address this challenge, existing approaches suffer from inefficiencies due to long backpropagation paths and hidden vector distortion. To overcome these limitations, we propose **Late Adapter Tuning (LAT)**, a novel PEFT method that optimizes training costs by fine-tuning only a single hidden layer near the model's output. LAT integrates a customized adapter architecture with hard prompting to preserve hidden vector dimensions and shorten gradient propagation paths. Experiments on four classification datasets demonstrate that LAT reduces training time by 2.4×, decreases GPU memory usage by 76.5%, and improves accuracy by 4.31% compared to full-parameter fine-tuning. Our work provides a practical solution for deploying LLMs in low-resource environments while advancing the theoretical understanding of gradient-efficient adaptation strategies.

Keywords: Large Language Models, Parameter-Efficient Tuning, Adapter Tuning, Text Classification, Computational Efficiency

1 Introduction

Pre-trained Language Models (PLM) [1, 2, 3, 4, 5] are the cornerstone of Natural Language Processing (NLP) to solve downstream tasks, and full-parameter fine-tuning has always been the default method to adapt pre-trained language models to downstream tasks [6, 7, 8, 9]. In the training stage, it is a general consensus in the industry that the higher the number of parameters of the benchmark model, the stronger its ability to solve downstream tasks [10]. Therefore, within a certain training cost, choosing a model with a higher number of parameters often means having better performance in downstream tasks. However, with the increasing number of parameters of pre-trained language models, the cost of fine-tuning all parameters is difficult for individual researchers to bear. At the same time, the computing power in some scenarios is very limited, for example, the computing power in the edge computing scenarios itself is very limited [11]. Therefore, it is necessary to find a low-cost method to adapt pre-trained language models to downstream tasks.

Parameter-Efficient Fine-Tuning (PEFT) [12, 13] is one of the mainstream low-cost methods to adapt pre-trained language models to downstream tasks. This method performs well in simple classification tasks and is mainly used in few-shot learning. The main idea is to first cancel the gradient of all parameters of the original pre-trained language model, and then select some of the specific parameters for gradient update. As shown in Figure 1, in the process of multi-task training, the parameter-efficient fine-tuning method gets rid of the dilemma that a pre-trained language model needs to be fine-tuned for a task in the past [14], and only needs to fine-tune a small number of specific parameters of the pre-trained language model to complete the adaptation to downstream tasks. In the PEFT method, specific parameters only account for about 1% of the parameters of the original model [15, 16]. This makes the PEFT method not only optimize the training cost, but also optimize the storage cost, which is more suitable for multi-task learning.

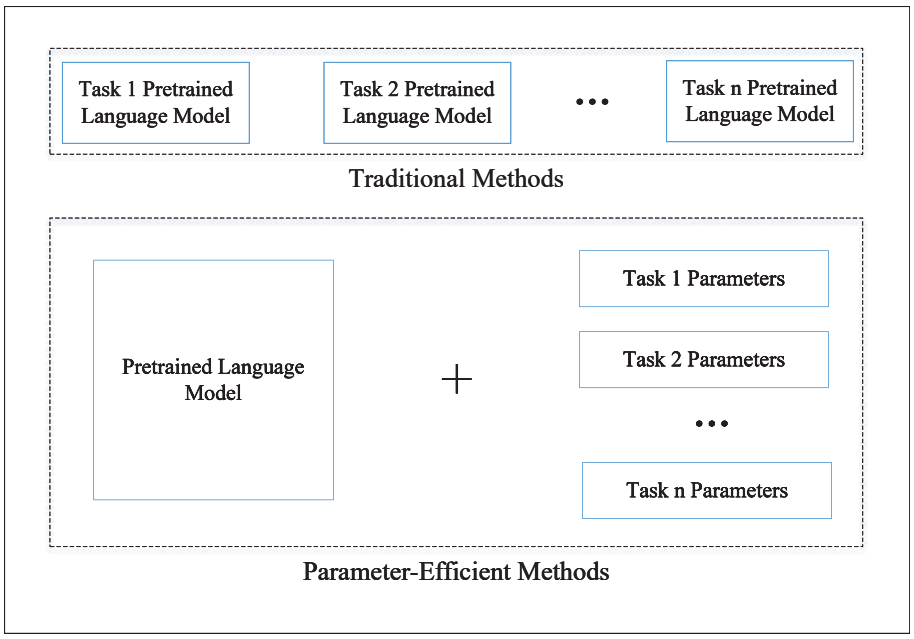


Figure 1: Multitasking Storage Form

Traditional parameter efficiency methods pay more attention to the type of updating parameters and only consider the maximization of performance, ignoring the impact of other factors on cost. In the process of updating, the increase of the length of the backpropagation [17] path will make the calculation of gradient updating larger and time-consuming, and the storage gradient will be too much and occupy high GPU memory usage. LPT (Late Prompt Tuning) [18] was keen to spot this problem and for the first time limited the selected parameters to areas close to the input. SPT (Selective Prompt Tuning) [19] further optimizes the problem of the number of layers. However, the above methods directly concatenate additional parameters on the hidden vector in the middle of the model, which destroys the dimension of the original hidden vector, and changes the dimension of

the subsequent hidden vector in the forward reasoning process, making its performance no advantage compared with other PEFT methods. Black-box tuning [20] adopts the idea of soft prompt, and changes the parameter to gradient-free optimization [21, 22, 23] to reduce the training cost. However, the performance effect of gradient-free optimization on full data is very different from that of other parametric high efficiency methods. At present, it is only applicable to the model of GPT-3 [3], which is a large model to provide services to the outside.

In order to consider the calculation cost and feasibility, this paper continues to follow the idea of LPT, optimizes the impact of backpropagation on the cost, and proposes a new PEFT method. The main contributions are as follows:

(1) Combine the neural network and the hidden vector within the model to solve the problem of LPT destroying the hidden vector dimension. A parallel adapter method was used to perform downstream task adaptation for hidden vectors near the model's output. A new parameter-efficient method called LAT (Late Adapter Tuning) is proposed.

(2) Combined with the hard prompt method, solve the problem of the mismatch between the model in the pre-training stage and the downstream task training stage, so as to improve the performance of the model.

(3) A new adapter architecture is proposed to make it more suitable for the update of hidden vectors, and found that it further shortens the distance of the backpropagation path during the gradient update, and improves the performance while reducing the training cost.

2 Related Work

The parameter-efficient methods are divided into three methods according to the different specific parameter processing methods: (1) Add additional parameters to the original model, and only fine-tune the parameters of the added part. (2) Selectively fine-tune some parameters in the original model. (3) Reparameterize some parameters in the original model. (4) Further improve the performance of fine-tuning in downstream tasks by mixing several previous different types of fine-tuning methods.

2.1 Additive PEFT

The new parameter method is to cancel the gradient of the parameters of the original pre-trained language model, and add the parameters with the gradient to the original model for subsequent training update. According to the location of adding parameters, it can be divided into two types: adding parameters on the architecture and adding parameters on the hidden vectors. The most typical method of adding parameters to the architecture is Adapter [13, 24, 25, 26, 27], which adds a bottleneck architecture network to the Transformer [28] of each layer of the model. This bottleneck structure consists of a down-projection, a non-linear transformation, and an up-projection, which allows for the injection of task-specific information with minimal additional parameters. In contrast, the most typical method of adding parameters to the hidden vector is Soft Prompt [29], which splicing a randomly initialized word vector with gradient directly next to the word vector, and then updates the word vector with gradient in the update. This approach has been further explored and refined in subsequent studies [30, 31, 32]. Both Adapter and Soft Prompt techniques are designed to maintain the vast majority of the original model's parameters static, thereby preserving the extensive knowledge encoded during pre-training. By doing so, these methods not only enable PEFT but also facilitate efficient transfer learning, allowing the model to quickly adapt to new tasks and domains with minimal computational overhead. Furthermore, these techniques have been shown to mitigate the risk of overfitting to the target task, as the additional parameters are few in number and do not disrupt the pre-trained model's core representations.

2.2 Selective PEFT

Selective PEFT strategies involve finetuning only a select few parameters of the pre-trained model, while the majority of the parameters remain frozen. This approach allows for a more targeted update

that can be computationally efficient and reduce the risk of overfitting. Liu, et al. [33] introduced a novel parameter-efficient fine-tuning method called $(IA)^3$, which involves scaling the activations using learned vectors, thereby achieving enhanced performance with a minimal number of additional parameters. Additionally, they proposed a recipe based on the T0 model that requires no task-specific tuning or modifications, demonstrating its effectiveness on a variety of unseen tasks. Bitfit [34] is another example of selective PEFT, where only the bias terms of the network are updated during training, while all other weight parameters are kept frozen. This method surprisingly shows that updating a small fraction of the parameters can lead to significant performance gains. Lee, et al. [35] found that only updating the last two Transformer layers of the model in the training process can achieve 80% of the performance of full parameter fine-tuning. In summary, selective PEFT methods offer a balance between the flexibility of fine-tuning and the efficiency of parameter usage, making them attractive for scenarios where resources are limited or where rapid adaptation to new tasks is required.

2.3 Reparameterized PEFT

The original reparameterization of parameters is to use two low rank matrices to replace the change of parameters. This method originated from the study of the intrinsic dimension [16] of the pre-trained language model, which found that the model can be learned in low-dimensional space when adapting to downstream tasks. Low Rank Adapter (LoRA) [36] decomposes the matrices Q and V in the attention mechanism [28] to low rank, and the two low-rank matrices are used to simulate the updates of the original matrices. Building upon the foundation laid by LoRA, QLoRA [37] emerges as a variant that further optimizes memory usage during fine-tuning of large language models. QLoRA employs model quantization, which significantly reduces the GPU memory peak usage by approximately 75% compared to LoRA, making it particularly advantageous in environments with memory constraints. Moreover, QLoRA supports larger batch sizes, which is beneficial for training efficiency. While LoRA is known for its faster training speed due to fewer trainable parameters, QLoRA strikes a balance by offering similar performance with the added benefit of substantial memory savings. In addition to LoRA, QLoRA, there are several variants based LoRA. Such as, DyLoRA [38], ERAT-DLoRA [39], MELoRA [40].

2.4 Hybrid PEFT

Hybrid PEFT combines elements from different PEFT methods to optimize the trade-off between performance and parameter efficiency. Karimi Mahabadi, et al. [15] propose Compacter, Compacter accomplishes a better trade-off between task performance and the number of trainable parameters by building on top of ideas from adapters, low-rank optimization, and parameterized hypercomplex multiplication layers. Specifically, Compacter inserts task-specific weight matrices into a pretrained model's weights, which are computed efficiently as a sum of Kronecker products between shared slow weights and fast rank-one matrices defined per Compacter layer. Different parameter-efficient fine-tuning methods may perform rather differently on the same task, making it nontrivial to select the most appropriate method for a specific task, especially considering the fast-growing number of new parameter-efficient fine-tuning methods and tasks. Mao, et al. [41] propose a unified framework called UniPELT, which incorporates different parameter-efficient fine-tuning methods as submodules and learns to activate the ones that best suit the current data or task setup via gating mechanism.

3 Method

The method of PEFT is to change the adaptation of the model in the downstream task from full parameter adaptation to a few parameters adaptation. As shown in Figure 2, the newly proposed PEFT method **Late Adapter Tuning (LAT)** model first needs to freeze other irrelevant parameters, and the key steps are divided into three modules.

(1) Prompt template construction. Convert the classification task into cloze filling, modify the original sentence to sentences with [MASK] symbols, and mark the position of [MASK].

(2) Adapter layer design. Send the hidden vectors obtained through the *k-layer* coding layer into the custom Adapter structure for update, and then send the new hidden vectors obtained through the residual connection into the next coding layer.

(3) Answer mapping. Extract the word vector at [MASK] position in the hidden vector passing through the *z-th* coding layer, and then send it to the word mapping layer to map the answer to the custom dictionary.

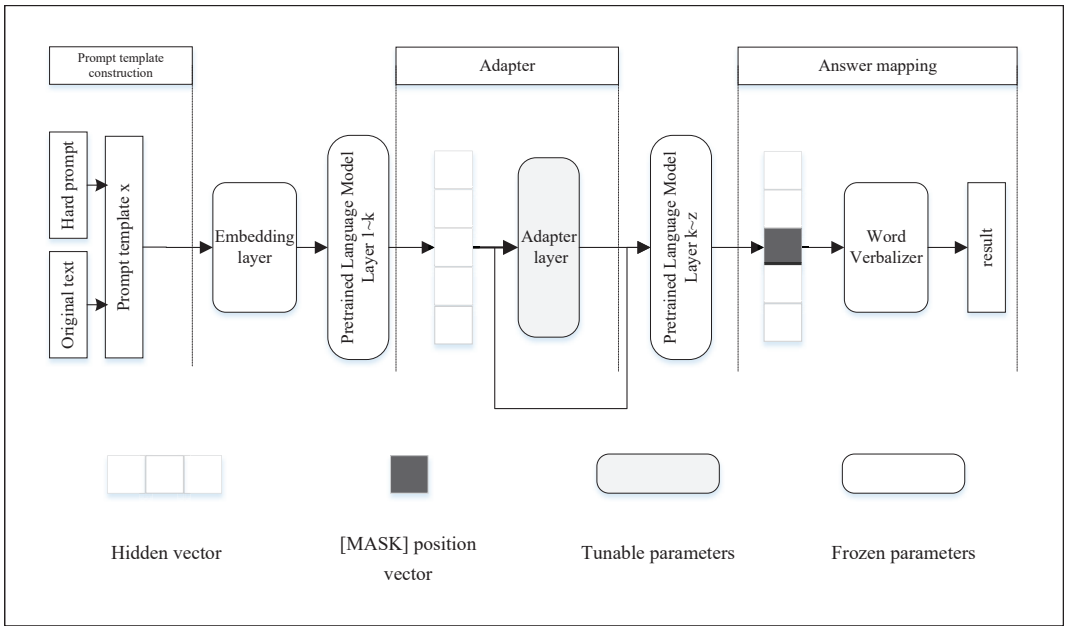


Figure 2: The basic structure of the LAT model, where k is the location of the specified hidden layer and z is the number of coding layers of the model.

3.1 Prompt Template Construction

In the mask language model, it is unsupervised in the pre-training stage with words that predict the mask position. When using a mask language model to adapt to downstream classification tasks, the last layer of hidden layer is often used to complete the classification task directly through the softmax classifier. In order to narrow the gap between the downstream task and the model pre-training stage, prompt learning [42] was proposed.

Table 1: Hard prompt template

Dataset	**Hard prompt template X (S represents original sentence)**
MPQA[43]	S. It was [mask].
MR[44]	S. It was [mask].
SUBJ[44]	S. It was [mask].
TREC[45]	[mask]:S.

The core idea of hard prompt is to modify the original sentence into prompt text marked with [MASK], that is, to modify the traditional classification task to a cloze task that predicts the words in the [MASK] position, so as to make full use of the ability learned by the pre-trained language model in the pre-training stage. Table 1 shows the different hard prompt templates X on the four categorical datasets. As shown in the table 1, X consists of the original sentence S connected with the [MASK]

marked prompt P and can be represented by the following formula.

$$S = [s_1, s_2, \ldots, s_m] \tag{1}$$
$$P = [p_1, \ldots, [\text{MASK}], \ldots, p_n] \tag{2}$$
$$X = [S, P] \tag{3}$$

Where m represents the original sentence length, n represents the length of the prompt text, $s_i(1 \leq i \leq m)$ represents the token in the sentence S, and $p_j(1 \leq j \leq n)$ represents the token in the prompt text.

It can also be seen from the table that the hard prompt template adds some contextual information related to the problem compared to the original input, which helps the model adapt to downstream tasks to a certain extent.

3.2 Adapter Layer Design

The adapter layer uses customized adapters to adapt to different downstream tasks. This module modifies the traditional multiple downstream tasks corresponding to multiple large language models into multiple downstream tasks corresponding to one large language model and multiple adapter layers. It not only optimizes the storage cost for multi-tasks, but also proposes a new PEFT method for pre-training language model adaptation to downstream tasks. The method is divided into two parts: selection of update hidden vectors and adapter network design.

3.2.1 Layer Selection

First of all, you need to confirm the position of the adapter. In order to avoid the problem that the backpropagation path distance is too long, the range of hidden vectors is limited to the hidden vectors inside the model. For the sake of performance and cost, the original adapter [13] was modified to act between two coding layers within each coding layer of the model. The process of the whole model after the change is as follows:

As with the original input information, the hard prompt template X obtained in § 3.1 and the position information X_pos are put into the word embedding layer trained by the pre-trained language model to obtain the word embedding vector V, as shown below:

$$V = WordEmbedding(X, X_{pos}) \tag{4}$$

Location information is used to identify the location of a word. Then, the obtained word embedding vector is fed into the coding layer of the first k layer of the language model to obtain the hidden vector H that needs to be updated, as follows.

$$H = Encoder_{1\sim k}(V) \tag{5}$$

where $Encoder$ is the original coding layer that comes with the model, and k is the number of layers that need to be customized. After the hidden vector is passed through a custom adapter, residual concatenation is performed to obtain an updated hidden vector H_{new}, which is put into the remaining coding layers to obtain a final hidden vector H_{last}, as shown below.

$$H_{new} = H + Adapter(H) \tag{6}$$
$$H_{last} = Encoder_{k\sim z}(H_{new}) \tag{7}$$

Where z is the number of layers encoded by the language model, H, H_{new} and H_{last} have the same dimension, and $H \in \mathbb{R}^{(m+n)\times d}$, where d is the dimension of the hidden vector of the model.

3.2.2 Adapter Architecture

This paper has further modified the network architecture in the Adapter method. As shown in Figure 3, the difference between the new adapter and the original is that it actually updates the hidden vector in higher dimensions, and adds the normalization in the input and the non-linear layer in the output. The reason for this is that the original Adapter structure first reduces the input hidden vector dimension (down projection) and then rises to the same dimension as the model dimension (up projection). This order causes information loss because the down projection drops a portion of the information. The proposed LAT method first raises the input hidden vector dimension to a higher dimension than the model dimension (up projection), and then decreases to the same dimension as the model dimension (down projection). This order can retain more information because the up projection does not lose information and only adds dimensionality. However, with the number of parameters, to solve this problem, the LAT method adds a normalization layer to the input. The normalization layer can effectively reduce the number and computational parameters because it can scale and offset the input data to reduce the size of the parameters. Moreover, the normalization layer can also speed up the model training and improve its stability.

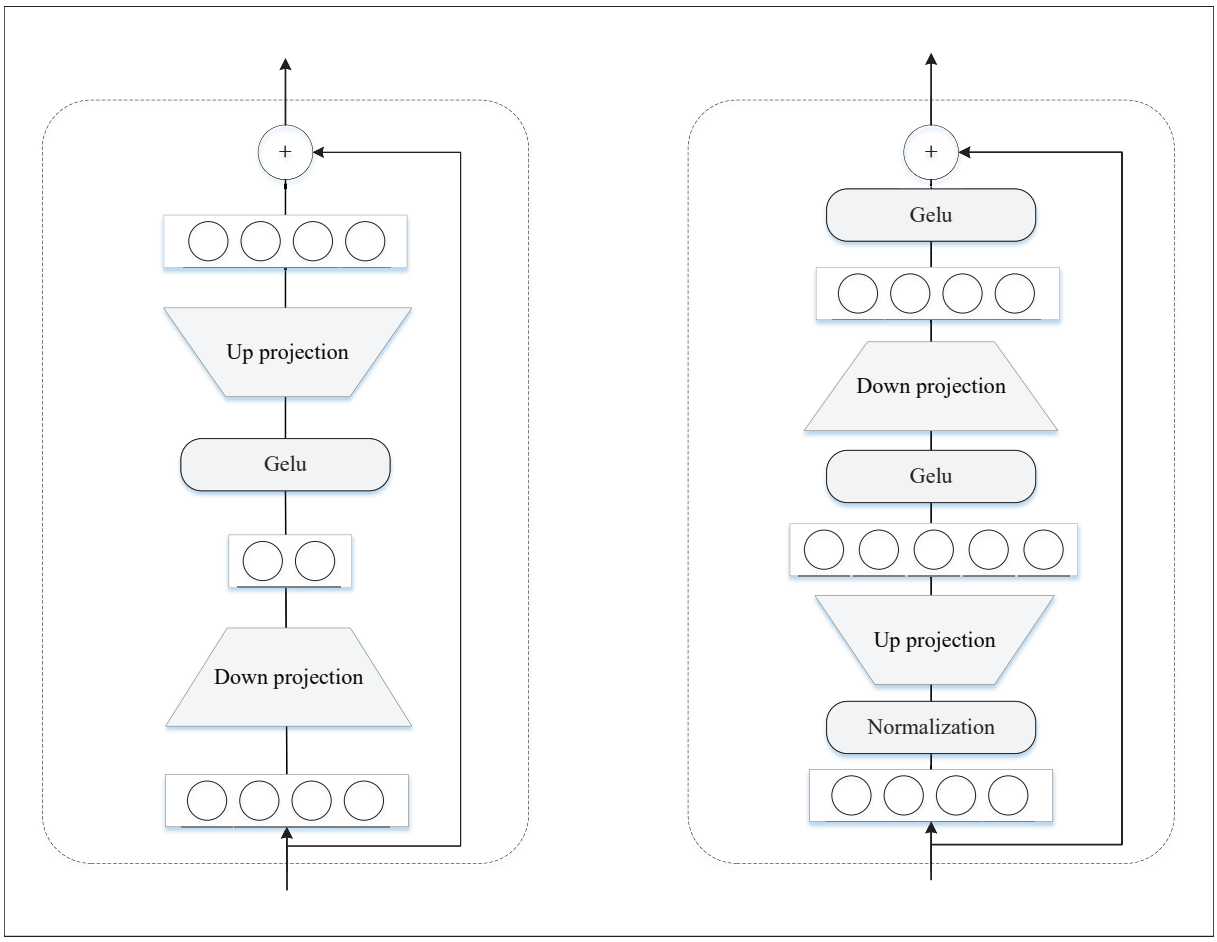


Figure 3: The left picture shows the original architecture of the adapter, and the right picture shows the improved adapter

$$Adapter(H) = Gelu(w_{down}Gelu(w_{up}Norm(H))) \tag{8}$$

where $Norm$ is the normalization of vectors and $Gelu$ is the Gaussian error linear unit activation function. $w_{\text{up}} \in \mathbb{R}^{d\times u}$, $w_{\text{down}} \in \mathbb{R}^{u\times d}$, where d is the hidden layer dimension of the pre-trained language model and u is the upward projected dimension.

3.3 Answer Mapping

The answer mapping layer limits the final output of the model, so that the model can make predictions on the custom word list, and selects the word with the highest probability as the final prediction result. Table 2 shows the answer label D set on the four datasets in this paper.

Table 2: The labels of each answer on the dataset

Dataset	Label
MPQA	positive, negative
MR	positive, negative
SUBJ	subjective, objective
TREC	abbr., entity, description, human, loc., num.

Let $H_{last} = [h_1, h_2, \ldots, h_{mask}, \ldots, h_d]$, the last layer hidden vector obtained in § 3.2.1, and d be the dimensions of the model hidden vector. Extract the word vector h_{mask} of the last hidden vector corresponding to [MASK] in the hard prompt template. The mask language model header (MLM_{head}) trained by the pre-trained language model maps the hidden vector h_{mask} to the thesaurus dimension of the pre-trained language model, as follows:

$$Y_v = MLM_{head}(h_{mask}) \tag{9}$$

In this case, $Y_v = [y_1, y_2, y_3, \ldots, y_v]$, where v is the vocabulary length of the pre-trained language model. H_v is the probability of each word appearing at [MASK] in the vocabulary. Extract the word probability value corresponding to the answer label in Y_v to form the probability distribution Y of the final answer. The vector in Y comes from the vector in Y_v corresponding to the answer mapping D, and the length is consistent with D. The word with the highest probability within the answer label range is extracted as the final prediction result y, as shown below:

$$y = argmax(Y) \tag{10}$$

LAT algorithmic details are provided in Algorithm 1.

Algorithm 1: LAT Training

1. **Input**: Pre-trained model M, dataset $\mathcal{D}$.
2. **Freeze** all parameters of M.
3. **Insert** Adapter at layer k.
4. **Construct Prompt**: Convert $S \in \mathcal{D}$ to $X = [S; P]$.
5. **Forward Pass**: Compute H_{last} .
6. **Backward Pass**: Update adapter parameters via gradient descent.
7. **Output**: Fine-tuned adapter weights.

4 Experiment and Result Analysis

4.1 Experimental Dataset

In order to evaluate the effectiveness of the method in this paper, this experiment was conducted on the publicly available datasets MPQA, MR, SUBJ, TREC datasets, and the distribution of the data is shown in Table 3. The four datasets are uniformly distributed in training set, validation set and test set, among which MPQA is the binary opinion dataset, including 3311 positive texts and 7293 negative texts. MR is the dichotomy emotion analysis dataset. SUBJ is the binary emotion polarity dataset. TREC is the sixth problem classification dataset.

Table 3: Dataset split details

Dataset	Training set	Validation set	Test set
MPQA	7606	1000	2000
MR	7662	1000	2000
SUBJ	7000	1000	2000
TREC	4952	500	500

4.2 Experimental Parameter Setting and Evaluation

The experiment in this paper uses the Ubuntu20.04.5 operating system, equipped with a 3090 GeForce RTX 3090 24GB graphics card, using the PyThon3.7 development environment and PyTorch 1.13.1 development framework, as well as the development framework of HuggingFace [46]. The experimental hyperparameters are the same on the four datasets, as shown in Table 4.

Table 4: Experimental setting

Hyperparameters	Value
pre-trained language model	RoBERTa-large[2]
learning rate	1.00E-03
maximum sentence length	256
weight decay	0.1
hidden layer	20
project dimensions upward	4096
learning rate warmup	0.06
epoch	10
batch size	16

We use accuracy as the evaluation metric and record it with I_{ACC}, and the number of correctly classified samples is recorded as T, and the total number of data samples is recorded as N. The evaluation indicators are calculated as follows:

$$I_{ACC} = T/N \tag{11}$$

4.3 Results and Analysis

To verify the performance of the designed parameter efficient method, we compare it with other mainstream parameter efficient methods as follows.

- Adapter[13]: The adapter is connected inside each coding layer of the pre-trained language model, which only updates the parameters of the adapter.
- AdapterDrop[47]: To pruned them on the basis of Adapter.
- Prompt Tuning[29]: Stitch a soft prompt to the input data, and only update the soft prompt during training.
- P-tuning V2[48]: Soft prompts are added as input at each layer of the Transformer network, and only soft prompts are updated during training.
- S-IDPG-PHM[49]: The soft prompts are re-parameterized and connected to each hidden layer of the pre-trained language model. Only the soft prompts are updated during the training process.
- Bitfit[34]: Only the weights in the network are updated during training.

- LoRA[36]: The Q and V matrices in each layer of attention mechanism in the pre-training language model are updated and reparameterized for low-rank decomposition, and the training process only updates the reparametric parameters.
- LPT[18]: The re-parameterized soft cues are connected to the hidden vectors obtained after passing a single encoding layer inside the model, and only the soft cues are updated during the training process.

4.3.1 Training Cost Comparison

To verify the effect of the distance from backpropagation on cost, Table 5 compares the computational efficiency of LAT against full-parameter fine-tuning and state-of-the-art PEFT methods. It can be seen from the table that the number of parameters involved in training and the training cost are not a simple linear relationship. Even though the parameter quantity of LAT is not superior to other parameter efficient methods, its training speed and GPU memory consumption are significantly superior to other methods. LAT achieves a 2.4× faster training speed (27.8 tokens/ms vs. 11.6 tokens/ms for full tuning) and reduces GPU memory usage by 76.5% (5.46 GB vs. 23.3 GB). This efficiency stems from two key design choices:

(1) Shortened Backpropagation Path. By limiting gradient updates to the final hidden layer (layer 20 in RoBERTa-large), LAT avoids redundant gradient calculations for frozen parameters. This aligns with theoretical analyses showing that gradient computation scales linearly with propagation depth [17, 18].

(2) Lightweight Adapter Architecture. LAT's inverted bottleneck adapter (up → down projection) introduces only 8.4M trainable parameters (2.4 of the model size), significantly reducing memory overhead compared to methods like Adapter (1.6M parameters but higher memory usage due to multi-layer tuning).

These improvements enable LAT to operate on edge devices with limited GPU resources (e.g., IoT systems [11]), making large-scale LLM deployment feasible in real-world scenarios.

Table 5: Comparison of training costs for efficient mainstream parameters

Method	Trainable Parameters	Training Speed (tokens/ms)	GPU Memory Usage (GB)
Model Tuning	355M	11.6	23.3
Adapter	1.6M	15.5	16.5
AdapterDrop	811K	21.6	9.5
Prompt Tuning	21K	16.9	17.8
P-tuning V2	985K	19.2	16.8
S-IDPG-PHM	114K	12.0	16.8
Bitfit	273K	16.5	15.7
LoRA	788K	16.4	16.2
LPT	792K	23.2	10.285
LAT	8.4M	27.8	5.455

4.3.2 Comparative Experiment and Result Analysis

All methods were run with 3 random seeds, and the variance of mean results is in parentheses. Except LPT, other methods were all from literature [19]. As shown in Table 6, the LAT method proposed in this paper achieves better classification performance than other comparison methods on all four datasets. LAT method has a large gap with other methods on MR and MPQA datasets, reaching 98.1% and 99.9% respectively, and 7.9% and 8.6% compared with other second place performance, while maintaining high stability. It can also be seen from the table that the performance of a series of

methods such as Prompt Tuning, P-tuning V2, S-IDPG-PHM, LPT, etc., which add soft hints to the hidden vector dimension, is lower than that of the remaining methods without destroying the hidden vector dimension. LAT solves the problem of destroying the hidden vector dimension of LPT to give full play to the basic ability of the original model.

Table 6: Performance of Methods on Different Datasets

Method	MPQA	MR	SUBJ	TREC
Model Tuning	90.2	91.3	96.8	97.6
Adapter	89.2(0.5)	91.6(0.4)	96.8(0.4)	97.0(0.3)
AdapterDrop	89.1(0.7)	91.0(0.5)	95.3(0.6)	95.7(0.5)
Prompt Tuning	88.8(0.8)	89.6(0.5)	93.9(0.6)	86.4(0.7)
P-tuning V2	89.9(0.6)	91.4(0.4)	96.5(0.2)	95.8(0.6)
S-IDPG-PHM	89.5(0.6)	90.8(0.5)	95.9(0.6)	89.3(0.4)
Bitfit	89.2(0.9)	91.8(0.5)	96.9(0.1)	96.2(0.3)
LoRA	90.1(0.3)	92.0(0.1)	97.1(0.4)	96.8(0.6)
LPT	90.9(0.3)	91.2(0.2)	96.2(0.1)	94.7(0.5)
LAT	98.1(0.1)	99.9(0.1)	97.3(0.3)	97.9(0.3)

To further verify the effectiveness of the new adapter architecture, ablation experiments were performed on the adapter model architecture, as shown in Table 7. As you can see from the table, in the newly designed architecture, the most beneficial thing for the model is to change the downward projection and upward projection of the original adapter to the upward projection and downward projection, and normalization also has a great impact on the result. In the experiment, the optimal number of layers of the original downward projection type adapter LAT-down is located at the 18^{th} layer of the hidden layer, while the optimal number of layers of the improved LAT is located at the 20^{th} layer of the hidden layer. This phenomenon indicates that LAT not only has superior performance compared to the pre-improved adapter, but also further reduces the cost. Compared with LAT-down method of the original adapter architecture, LAT training memory consumption is reduced by 1%. due to the large number of upward projection parameters, the calculation amount is increased, and the training speed is completely consistent. The validity of the new architecture adapter is further verified.

Table 7: Results of ablation experiments

Method	MPQA	MR	SUBJ	TREC
LAT	98.1(0.1)	99.9(0.1)	97.3(0.3)	97.9(0.3)
LAT-down	92.2(0.3)	92.5(0.2)	96.6(0.3)	97.0(0.2)
Without normalization	97.1(0.2)	99.0(0.1)	97.2(0.2)	97.9(0.1)
Without nonlinear layer	97.8(0.1)	99.2(0.6)	97.2(0.1)	97.8(0.4)

4.3.3 Hidden Layer Selection

Since the LAT method updates the hidden layer, this method needs to confirm the number of layers using hidden vectors. The closer the hidden vector is to the output, the shorter the length of the backpropagation path, and the smaller the training cost of the corresponding model will be. Therefore, to ensure performance as much as possible, choose the hidden vector close to the output. The results are shown in Figure 4. The performance of the last two layers has a great impact. Due to comprehensive considerations of performance and training cost, the 20th layer was finally selected as the number of layers of this method.

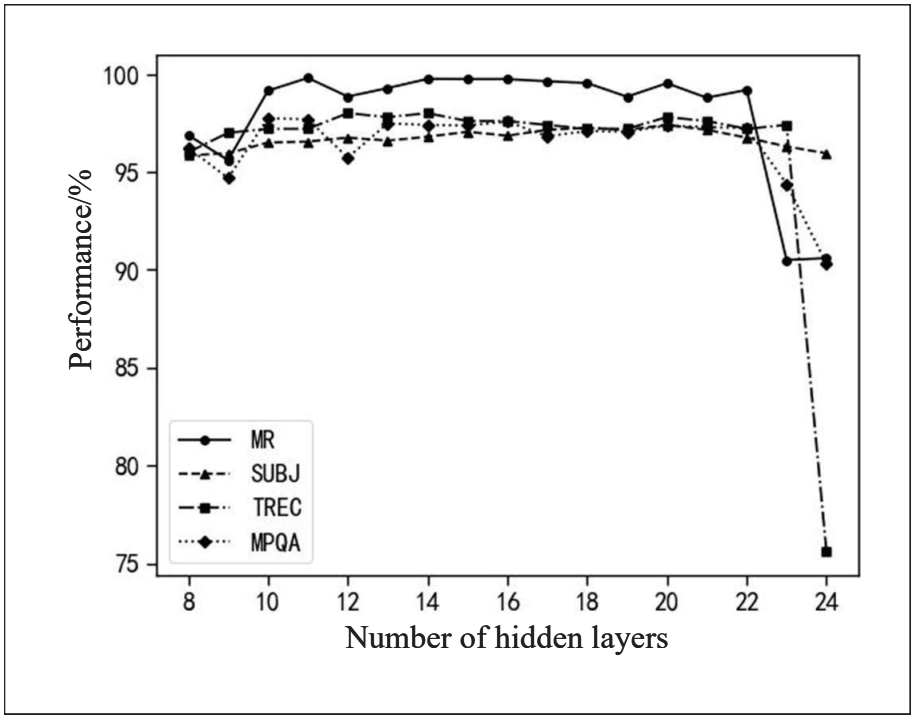


Figure 4: Performance of hidden vectors with different layers

5 Discussion

The experimental results demonstrate that LAT achieves significant improvements in computational efficiency and task performance compared to both full-parameter fine-tuning and existing PEFT methods. Below, we interpret these findings from theoretical and practical perspectives.

5.1 Theoretical Implications

Gradient Propagation Efficiency. LAT's design restricts parameter updates to the final hidden layer, shortening the backpropagation path from z layers (full tuning) to $z - k$ layers (where $k \approx z$). This aligns with the theoretical insight that gradient computation costs grow linearly with path length [17, 18]. By minimizing the propagation distance, LAT reduces redundant gradient calculations for frozen parameters, explaining its 2.4× faster training speed and 76.5% lower GPU memory usage (Table 5).

Task-Pre-training Alignment. The integration of hard prompts (e.g., *"It was [MASK]"*) bridges the gap between pre-training (mask prediction) and downstream tasks (classification). This aligns with the theoretical framework of prompt-based learning [42], where task reformulation enhances parameter efficiency by leveraging pre-trained capabilities.

5.2 Practical Implications

Resource-Constrained Deployment. LAT's 76.5% reduction in GPU memory usage (5.455 GB vs. 23.3 GB for full tuning) enables deployment on edge devices with limited computational resources, such as IoT systems [11]. For example, LAT could power real-time sentiment analysis on mobile platforms without requiring cloud-based infrastructure.

Cost-Effective Multi-Task Adaptation. By freezing 99% of parameters and storing only lightweight adapters (8.4M trainable parameters), LAT optimizes storage costs for multi-task scenarios. This is critical for applications like customer service chatbots, where models must handle diverse tasks (e.g., intent detection, emotion classification) without prohibitive hardware upgrades.

6 Conclusion

LAT method is an efficient and high-performance approach for parameter-efficient fine-tuning of pre-trained language models. By focusing on fine-tuning a single hidden layer near the model's output, LAT significantly reduces the number of training parameters, leading to lower training costs in terms of training time and GPU memory usage. Experimental results across four classification datasets demonstrate that LAT achieves a 2.4x reduction in training time, a 76.5% decrease in GPU memory usage, and a 4.31% improvement in accuracy compared to full-parameter fine-tuning. These findings highlight LAT as a cost-effective and high-performance solution for downstream task adaptation in large language models.

However, LAT also has some limitations. It is primarily suitable for downstream tasks that require fewer parameters for fine-tuning, and its performance may not be as good as other PEFT methods for tasks requiring a large number of parameters, such as machine translation. Additionally, the performance of the LAT method is affected by the choice of pre-trained language models, and choosing suitable pre-trained language models can improve the performance of the LAT method.

Future research can further explore the application scenarios and improvement directions of the LAT method. One direction is to explore the use of other deep learning neural network models to construct the Adapter layer, such as convolutional neural networks, recurrent neural networks, and others, to further enhance the model's expressive power. Another direction is to apply the LAT method to other pre-trained language models, such as those with open-source code, and verify its performance on different models. Additionally, the LAT method can be applied to other downstream tasks, such as question-answering systems [50], text summarization [51], and others, and explore its performance on different tasks.

Funding

The APC was funded by Dazhou key Laboratory of Government data security (No.ZSAQ202313), Foundation of Geely University of China (No.2024JG30063, 2024xzkzd011).

Author contributions

Zhengjie Gao provided the research idea and resources for this work, and revised the manuscript for multiple times. Rongcheng Li and Yuxin Fan conducted the experiments and wrote the manuscript. Min Liao and Xinyu Song provided comments and guided experiments for the revised version. All authors reviewed the manuscript.

Conflict of interest

The authors declare no conflict of interest.

References

[1] Devlin J, Chang M W, Lee K, et al. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding, *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. pp. 4171–4186.

[2] Liu, Y. (2019). Roberta: A robustly optimized bert pretraining approach, *arXiv preprint arXiv:1907.11692*, vol. 364.

[3] Gao, T.; Fisch, A.; Chen, D. (2021). Making Pre-trained Language Models Better Few-shot Learners, In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 3816–3830.

[4] Raffel, C.; et al. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer, *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67.

[5] Chowdhery, A.; et al. (2023). Palm: Scaling language modeling with pathways, *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113.

[6] Canchila, S.; Meneses-Eraso, C.; Casanoves-Boix, J.; Cortés-Pellicer, P.; Castelló-Sirvent, F. (2024). Natural language processing: An overview of models, transformers and applied practices, *Computer Science and Information Systems*, no. 00, pp. 31–31.

[7] Menta, A.; Garcia-Serrano, A. (2024). Reaching quality and efficiency with a parameter-efficient controllable sentence simplification approach, *Computer Science and Information Systems*, no. 00, pp. 17–17.

[8] Liu, H., Ma, Y., Gao, C., Qi, J. & Zhang, D. (2023). Chinese Named Entity Recognition Method for Domain-Specific Text. *Tehnički vjesnik*, 30 (6), 1799-1808.

[9] Cheng, Y., Wan, Y., Sima, Y., Zhang, Y., Hu, S. & Wu, S. (2022). Text Detection of Transformer Based on Deep Learning Algorithm. *Tehnički vjesnik*, 29 (3), 861-866.

[10] Kaplan, J.; et al. (2020). Scaling laws for neural language models, *arXiv preprint arXiv:2001.08361*.

[11] Muntean, I.; Mois, G. D.; Folea, S. C. (2021). Development and Analysis of Low-Cost IoT Sensors for Urban Environmental Monitoring, *International Journal of Computers Communications & Control*, vol. 16, no. 5.

[12] Ding, N.; et al. (2023). Parameter-efficient fine-tuning of large-scale pre-trained language models, *Nature Machine Intelligence*, vol. 5, no. 3, pp. 220–235.

[13] Houlsby, N.; et al. (2019). Parameter-efficient transfer learning for NLP, In *International Conference on Machine Learning*, PMLR, pp. 2790–2799.

[14] Peters, M. E.; Ruder, S.; Smith, N. A. (2019). To Tune or Not to Tune? Adapting Pretrained Representations to Diverse Tasks, In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pp. 7–14.

[15] Karimi Mahabadi, R.; Henderson, J.; Ruder, S. (2021). Compacter: Efficient low-rank hypercomplex adapter layers, *Advances in Neural Information Processing Systems*, vol. 34, pp. 1022–1035.

[16] Aghajanyan, A.; Gupta, S.; Zettlemoyer, L. (2021). Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning, In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 7319–7328.

[17] Rumelhart, D. E.; Hinton, G. E.; Williams, R. J. (1986). Learning representations by back-propagating errors, *Nature*, vol. 323, no. 6088, pp. 533–536.

[18] Liu, X.; Sun, T.; Huang, X.-J.; Qiu, X. (2022). Late Prompt Tuning: A Late Prompt Could Be Better Than Many Prompts, In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 1325–1338.

[19] Zhu, W.; Tan, M. (2023). Improving Prompt Tuning with Learned Prompting Layers, *arXiv preprint arXiv:2310.20127*.

[20] Sun, T.; Shao, Y.; Qian, H.; Huang, X.; Qiu, X. (2022). Black-box tuning for language-model-as-a-service, In *International Conference on Machine Learning*, PMLR, pp. 20841–20855.

[21] Hansen, N.; Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies, *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195.

[22] Hansen, N.; Müller, S. D.; Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES), *Evolutionary Computation*, vol. 11, no. 1, pp. 1–18.

[23] Rios, L. M.; Sahinidis, N. V. (2013). Derivative-free optimization: a review of algorithms and comparison of software implementations, *Journal of Global Optimization*, vol. 56, no. 3, pp. 1247–1293.

[24] Lei, T.; et al. (2023). Conditional adapters: Parameter-efficient transfer learning with fast inference, *Advances in Neural Information Processing Systems*, vol. 36, pp. 8152–8172.

[25] Pfeiffer, J.; Kamath, A.; Rückle, A.; Cho, K.; Gurevych, I. (2021). AdapterFusion: Non-Destructive Task Composition for Transfer Learning, In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 487–503.

[26] Zhao, H.; Fu, J.; He, Z. (2023). Prototype-based HyperAdapter for Sample-Efficient Multi-task Tuning, In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4603–4615.

[27] Chronopoulou, A.; Peters, M. E.; Fraser, A.; Dodge, J. (2023). AdapterSoup: Weight Averaging to Improve Generalization of Pretrained Language Models, In *Findings of the Association for Computational Linguistics: EACL 2023*, pp. 2054–2063.

[28] Vaswani, A. (2017). Attention is all you need, *Advances in Neural Information Processing Systems*.

[29] Lester, B.; Al-Rfou, R.; Constant, N. (2021). The Power of Scale for Parameter-Efficient Prompt Tuning, In *Proceedings of the 2021 Conference on Empirical Methods in Natural*.

[30] Li, J.; Aitken, W.; Bhambhoria, R.; Zhu, X. (2023). Prefix Propagation: Parameter-Efficient Tuning for Long Sequences, In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1408–1419.

[31] Zhang, Z.-R.; Tan, C.; Xu, H.; Wang, C.; Huang, J.; Huang, S. (2023). Towards Adaptive Prefix Tuning for Parameter-Efficient Language Model Fine-tuning, In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1239–1248.

[32] Liu, X.; et al. (2024). GPT understands, too, *AI Open*, vol. 5, pp. 208–215.

[33] Liu, H.; et al. (2022). Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning, *Advances in Neural Information Processing Systems*, vol. 35, pp. 1950–1965.

[34] Zaken, E. B.; Goldberg, Y.; Ravfogel, S. (2022). BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models, In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1–9.

[35] Lee, J.; Tang, R.; Lin, J. (2019). What would elsa do? Freezing layers during transformer fine-tuning, *arXiv preprint arXiv:1911.03090*.

[36] Hu, E. J.; et al. (2021). Lora: Low-rank adaptation of large language models, *arXiv preprint arXiv:2106.09685*.

[37] Dettmers, T.; Pagnoni, A.; Holtzman, A.; Zettlemoyer, L. (2024). Qlora: Efficient finetuning of quantized LLMs, *Advances in Neural Information Processing Systems*, vol. 36.

[38] Valipour, M.; Rezagholizadeh, M.; Kobyzev, I.; Ghodsi, A. (2023). DyLoRA: Parameter-Efficient Tuning of Pre-trained Models using Dynamic Search-Free Low-Rank Adaptation, In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 3274–3287.

[39] Luo, D.; Zheng, K.; Wu, C.; Wang, X.; Wang, J. (2025). ERAT-DLoRA: Parameter-efficient tuning with enhanced range adaptation in time and depth aware dynamic LoRA, *Neurocomputing*, vol. 614, p. 128778.

[40] Ren, P.; et al. (2024). Melora: Mini-ensemble low-rank adapters for parameter-efficient fine-tuning, In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3052–3064.

[41] Mao, Y.; et al. (2022). UniPELT: A Unified Framework for Parameter-Efficient Language Model Tuning, In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6253–6264.

[42] Liu, P.; Yuan, W.; Fu, J.; Jiang, Z.; Hayashi, H.; Neubig, G. (2023). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing, *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35.

[43] Wiebe, J.; Wilson, T.; Cardie, C. (2005). Annotating expressions of opinions and emotions in language, *Language Resources and Evaluation*, vol. 39, pp. 165–210.

[44] Pang, B.; Lee, L. (2005). Seeing Stars: Exploiting Class Relationships for Sentiment Categorization with Respect to Rating Scales, In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pp. 115–124.

[45] Voorhees, E. M.; Tice, D. M. (2000). Building a question answering test collection, In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 200–207.

[46] Wolf, T.; et al. (2020). Transformers: State-of-the-art natural language processing, In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45.

[47] Rücklé, A.; et al. (2021). AdapterDrop: On the Efficiency of Adapters in Transformers, In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7930–7946.

[48] Liu, X.; et al. (2022). P-Tuning: Prompt Tuning Can Be Comparable to Fine-tuning Across Scales and Tasks, In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 61–68.

[49] Wu, Z.; et al. (2022). IDPG: An Instance-Dependent Prompt Generation Method, In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5507–5521.

[50] Zaib, M.; Zhang, W. E.; Sheng, Q. Z.; Mahmood, A.; Zhang, Y. (2022). Conversational question answering: A survey, *Knowledge and Information Systems*, vol. 64, no. 12, pp. 3151–3195.

[51] Alshammari, A.; Alzaidi, S. A.; SK, K. (2024). Enhancing Text Summarization with Linguistic Prompting and Reinforcement Learning: A Human-Centered Approach, *Tehnički Vjesnik*, vol. 31, no. 5, pp. 1431–1437.

Journal's webpage: http://univagora.ro/jour/index.php/ijccc/

This journal is a member of, and subscribes to the principles of,
the Committee on Publication Ethics (COPE).
https://publicationethics.org/members/international-journal-computers-communications-and-control

Cite this paper as:

Zhengjie, Gao.; Rongcheng, Li.; Yuxin, Fan.; Min, Liao.; Xinyu, Song. (2025). Late Adapter Tuning: A Cost-Effective Approach to Parameter-Efficient Fine-Tuning for Large Language Models, *International Journal of Computers Communications & Control*, 20(6), 6928, 2025.
https://doi.org/10.15837/ijccc.2025.6.6928