Memory-Tuning: A Unified Parameter-Efficient Tuning Method for Pre-Trained Language Models

Wang Qi, Rui Liu, Yuan Zuo, Fengzhi Li, Yong Chen, and Junjie Wu

***Abstract*—Conventional fine-tuning encounters increasing difficulties given the size of current Pre-trained Language Models, which makes parameter-efficient tuning become the focal point of frontier research. Recent advances in this field is the unified tuning methods that aim to tune the representations of both multi-head attention (MHA) and fully connected feed-forward network (FFN) simultaneously, but they rely on existing tuning methods and do not explicitly model domain knowledge for downstream tasks. In this work, we propose *memory-tuning*, a novel unified parameter-efficient tuning method with task-specific knowledge learning, for both MHA and FFN components in Transformer blocks. We also prove that the well-known prefix tuning is also a kind of memory tuning, which further ensures memory tuning is a genuine unified tuning method. Experiments on eight benchmark data sets including both sentence- and token-level tasks demonstrate that our method outperforms the state-of-the-art baselines even full-tuning in most cases.**

***Index Terms*—Pre-trained language model, parameter-efficient tuning, memory mechanism.**

I. Introduction

TRANSFER learning from Pre-trained Language Models (PLMs) [1], [2] is now the prevalent paradigm for natural language processing (NLP), yielding superior performances in many tasks [1], [3], [4]. Fine-tuning is the conventional way of leveraging large PLMs for downstream tasks, but it requires to update and store all the parameters of a PLM. As a result, one has to train and keep a modified replica of PLM for each task, which could be extremely expensive given the immense size of current PLMs.

Received 9 November 2022; revised 18 March 2024; accepted 9 November 2024. Date of publication 27 November 2024; date of current version 24 January 2025. The work of Yuan Zuo was supported by the National Natural Science Foundation of China (NSFC) under Grant 71901012. The work of Yong Chen was supported in part by the NSFC under Grant 62006005 and Grant 62372054, and in part by the National Key Research and Development Program of China under Grant SQ2022YFC3300043. The work of Junjie Wu was supported in part by the NSFC under Grant 72242101 and Grant 72031001, and in part by the Outstanding Young Scientist Program of Beijing Universities under Grant JWZQ20240201002. The associate editor coordinating the review of this article and approving it for publication was Dr. Panayiotis Georgiou. *(Corresponding authors: Yuan Zuo; Junjie Wu.)*

Wang Qi and Rui Liu are with the School of Computer Science and Engineering, Beihang University, Beijing 100191, China (e-mail: qiwangxt@buaa.edu.cn; lr@buaa.edu.cn).

Yuan Zuo, Fengzhi Li, and Junjie Wu are with the School of Economics and Management, Beihang University, Beijing 100191, China, and also with the MIIT Key Laboratory of Data Intelligence and Management Beijing 100191, China (e-mail: zuoyuan@buaa.edu.cn; lifengzhi@buaa.edu.cn; wujj@buaa.edu.cn).

Yong Chen is with the School of Computer Science, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: alphawolf.chen@gmail.com).

Digital Object Identifier 10.1109/TASLP.2024.3507558

Parameter-efficient tuning methods have been proposed to deal with the above challenge, which freezes most of the pre-trained parameters and only tunes a smaller set of them. For instance, the adapter approach inserts trainable bottleneck layers (i.e., adapter) into each layer of the pre-trained network [5], [6], [7], [8]. Inspired by prompting for PLMs, prefix-tuning [9] allows original tokens to attend to several tunable "virtual tokens", and similar works along this line include P-tuning [10], P-tuning v2 [11] and Prompt-tuning [12]. LoRA [13] learns low-rank matrices to approximate parameter updates. The above tuning methods can be categorized into two main classes according to the modified representations. One class includes prefix-tuning and LoRA that tune the representation of multi-head attention (MHA), and the other includes adapters that tune the representation of fully connected feed-forward network (FFN), both are the primary components in each Transformer [14] block.

More recently, research efforts have been made to design unified adapters to tune the representations of MHA and FFN at the same time. For example, He et al. [15] formulate a mathematically unified view of the aforementioned three types of tuning methods and propose the Mix-And-Match adapter (MAM Adapter) that integrates prefix-tuning and an adapter variant. Similarly, Mao et al. [16] propose UniPELT, which is another unified framework that incorporates prefix-tuning, adapters and LoRA as submodules via a gating mechanism. While these unified approaches have achieved excellent performances, they rely entirely on assembling the existing parameter-efficient tuning methods and rarely model task-specific knowledge explicitly — which is deemed very important in transfer learning of PLMs for downstream tasks. This indeed motivates our study on designing a higher-performance unified tuning method.

In this paper, we propose *memory-tuning*, a novel unified parameter-efficient tuning method for the hidden representations of both MHA and FFN components in each Transformer block. Similar to other parameter-efficient tuning methods, memory tuning adjusts only a minimal number of additional parameters. However, the distinctive feature of memory tuning is the incorporation of external memories, designed to retain task-specific knowledge that aids in transferring PLMs to downstream tasks. Importantly, these external memories can be integrated into both the MHA and the FFN, establishing memory tuning as a versatile and unified approach to parameter-efficient tuning. Moreover, we theoretically demonstrate that the well-known prefix-tuning method is actually a form of memory tuning applied to MHA. This insight further underscores our method's broader applicability, as prefix-tuning does not extend to FFN.

Extensive experiments on eight benchmark datasets and two types of tasks demonstrate that memory-tuning outperforms the state-of-the-art parameter-efficient tuning methods and even the full fine-tuning on most datasets. In particular, the superior performances on the token-level tasks demonstrate the evident advantage of memory-tuning in transfer learning of fine-grained NLP tasks. We finally illustrate the ability of memory-tuning in learning and storing task-specific knowledge via the visualization of attention scores over the external memory.

II. Preliminaries

A. Primary Components of Transformer Architecture

Most state-of-the-art PLMs are built upon the Transformer model [1], [17], [18], [19], [20], which consists of L stacked blocks, each containing two primary types of components namely multi-head attention (MHA) and fully connected feed-forward network (FFN). The i-th head of MHA conducts scaled dot-product attention to map queries $\boldsymbol{Q} \in \mathbb{R}^{n\times d_k}$ and key-value pairs $\boldsymbol{K} \in \mathbb{R}^{m\times d_k}$, $\boldsymbol{V} \in \mathbb{R}^{m\times d_v}$ to the output:

$$\begin{aligned}&\text{Attention}\left(\boldsymbol{Q}\boldsymbol{W}_i^{\boldsymbol{Q}}, \boldsymbol{K}\boldsymbol{W}_i^{\boldsymbol{K}}, \boldsymbol{V}\boldsymbol{W}_i^{\boldsymbol{V}}\right)\\ &= \text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{W}_i^{\boldsymbol{Q}}(\boldsymbol{K}\boldsymbol{W}_i^{\boldsymbol{K}})^\top}{\sqrt{d_k}}\right)\boldsymbol{V}\boldsymbol{W}_i^{\boldsymbol{V}}, \qquad (1)\end{aligned}$$

where n and m are the numbers of queries and key-value pairs, respectively, parameter matrices $\boldsymbol{W}_i^{\boldsymbol{Q}} \in \mathbb{R}^{d_k\times d_h}$, $\boldsymbol{W}_i^{\boldsymbol{K}} \in \mathbb{R}^{d_k\times d_h}$ and $\boldsymbol{W}_i^{\boldsymbol{V}} \in \mathbb{R}^{d_v\times d_h}$ are projections of queries, keys and values, respectively. Given a query vector $\boldsymbol{q} \in \mathbb{R}^d$ and a sequence of m contextual vectors $\boldsymbol{C} \in \mathbb{R}^{m\times d}$, MHA with h heads performs attention over $\boldsymbol{C}$ with $\boldsymbol{q}$:

$$\text{MHA}(\boldsymbol{q}, \boldsymbol{C}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\boldsymbol{W}^{\boldsymbol{O}}, \qquad (2)$$

where

$$\text{head}_i = \text{Attention}(\boldsymbol{q}\boldsymbol{W}_i^{\boldsymbol{Q}}, \boldsymbol{C}\boldsymbol{W}_i^{\boldsymbol{K}}, \boldsymbol{C}\boldsymbol{W}_i^{\boldsymbol{V}})$$

and $\boldsymbol{W}^{\boldsymbol{O}} \in \mathbb{R}^{d\times d}$ projects the concatenated heads to obtain the output of MHA, and $d = h \times d_h$ is the model dimension. The other component FFN consists of two linear transformations with a ReLU activation in between:

$$\text{FFN}(\boldsymbol{x}) = \text{ReLU}(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2, \qquad (3)$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{d\times d_f}$, $\boldsymbol{W}_2 \in \mathbb{R}^{d_f\times d}$. Typically, Transformer applies a large d_f, *e.g.*, $d_f = 4d$.

B. Overview of Parameter-Efficient Tuning Methods

Here, we briefly describe several state-of-the-art parameter-efficient tuning methods. Unless otherwise specified, they only tune the additional parameters while the PLM's original ones are frozen.

Adapters: The adapter approach adds a trainable bottleneck layer after the FFN in each Transformer block. A bottleneck layer consists of a pair of projections, namely down projection $\boldsymbol{W}_{\text{down}} \in \mathbb{R}^{d\times d_b}$ and up projection $\boldsymbol{W}_{\text{up}} \in \mathbb{R}^{d_b\times d}$, and a nonlinear activation function $f(\cdot)$. The size of the feedforward layer's output $\boldsymbol{h}$ is firstly shrunk with down projection and then recovered with up projection, *i.e.*, $d_b < d$. These adapters are surrounded by a residual connection, leading to the form:

$$\boldsymbol{h} \leftarrow \boldsymbol{h} + f(\boldsymbol{h}\boldsymbol{W}_{\text{down}})\boldsymbol{W}_{\text{up}}. \qquad (4)$$

Houlsby et al. [6] insert one adapter after the MHA and the other after the FFN in each block of the Transformer. Pfeiffer et al. [7] propose a more efficient adapter variant, which is inserted only after the "Add & Layer Norm" that normalizes the output of FFN.

Prefix-tuning: Inspired by prompting for language models, prefix-tuning [9] allows original tokens to attend to l "virtual tokens" prepended to the keys and values of MHA in each Transformer block. Specifically, two sets of prefix vectors $\boldsymbol{K}^{\boldsymbol{p}} \in \mathbb{R}^{l\times d}$ and $\boldsymbol{V}^{\boldsymbol{p}} \in \mathbb{R}^{l\times d}$ are concatenated with original keys $\boldsymbol{K}$ and values $\boldsymbol{V}$. Then, MHA is performed on the prefixed keys and values. Specifically, the computation of head$_i$ in (2) becomes:

$$\begin{aligned}\text{head}_i = \text{Attention}(&\boldsymbol{q}\boldsymbol{W}_i^{\boldsymbol{Q}},\\ &\text{Concat}(\boldsymbol{K}_i^{\boldsymbol{p}}, \boldsymbol{C}\boldsymbol{W}_i^{\boldsymbol{K}}),\\ &\text{Concat}(\boldsymbol{V}_i^{\boldsymbol{p}}, \boldsymbol{C}\boldsymbol{W}_i^{\boldsymbol{V}})), \qquad (5)\end{aligned}$$

where $\boldsymbol{K}^{\boldsymbol{p}}$ and $\boldsymbol{V}^{\boldsymbol{p}}$ are split into h vectors, one for each head, and $\boldsymbol{K}_i^{\boldsymbol{p}}, \boldsymbol{V}_i^{\boldsymbol{p}} \in \mathbb{R}^{l\times d/h}$ denote the i-head vector. Similar works include P-tuning [10] and P-tuning v2 [11]. Prompt-tuning [12] simplifies prefix-tuning by only prepending to the input word embeddings in the first layer.

LoRA: LoRA [13] injects tunable low-rank matrices into Transformer blocks to approximate the weight updates. For a pre-trained matrix $\boldsymbol{W} \in \mathbb{R}^{d\times k}$, its update is represented with a low-rank decomposition, that is $\boldsymbol{W} + \Delta\boldsymbol{W} = \boldsymbol{W} + \boldsymbol{W}_{\text{down}}\boldsymbol{W}_{\text{up}}$, where $\boldsymbol{W}_{\text{down}} \in \mathbb{R}^{d\times r}$ and $\boldsymbol{W}_{\text{up}} \in \mathbb{R}^{r\times k}$ are learnable parameters ($r \ll \min(d,k)$). Specifically, LoRA applies the above updates to the query and value projection matrices $(\{\boldsymbol{W}_i^{\boldsymbol{Q}}\}_{i=1}^h, \{\boldsymbol{W}_i^{\boldsymbol{V}}\}_{i=1}^h)$ in MHA. Given an input $\boldsymbol{x}$ to the projection matrix in MHA, LoRA modifies the output $\boldsymbol{h}$ as:

$$\boldsymbol{h} \leftarrow \boldsymbol{h} + s \cdot \boldsymbol{x}\boldsymbol{W}_{\text{down}}\boldsymbol{W}_{\text{up}}, \qquad (6)$$

where $s \geq 1$ is a manually adjustable scalar hyperparameter.

Unified Adapter: Recently, research efforts have been devoted to building a unified adapter. He et al. [15] bridge the gap among the aforementioned three tuning methods, and form a mathematically unified view of how they update hidden representation $\boldsymbol{h}$, that is:

$$\boldsymbol{h} \leftarrow a\boldsymbol{h} + b\Delta\boldsymbol{h}, \qquad (7)$$

where $\Delta\boldsymbol{h}$ shares a similar functional form $\text{proj\_down} \rightarrow$ nonlinear $\rightarrow \text{proj\_up}$, and a and b are scalars. Specifically, for prefix-tuning, $a + b = 1, a, b \in (0, 1)$, and the nonlinear function is Softmax; For adapters, $a = b = 1$ and the nonlinear function is ReLU; As to LoRA, $a = 1, b \geq 1$ and the nonlinear function degenerates to the identity function. Based on the above unified view, they propose the Mix-And-Match adapter (MAM Adapter) that applies prefix-tuning for MHA and scaled parallel adapter for FFN, where the latter is an adjusted adapter inspired by prefix-tuning and LoRA. UniPELT [16] is another unified framework that incorporates prefix-tuning, adapters and LoRA as submodules via a gating mechanism.

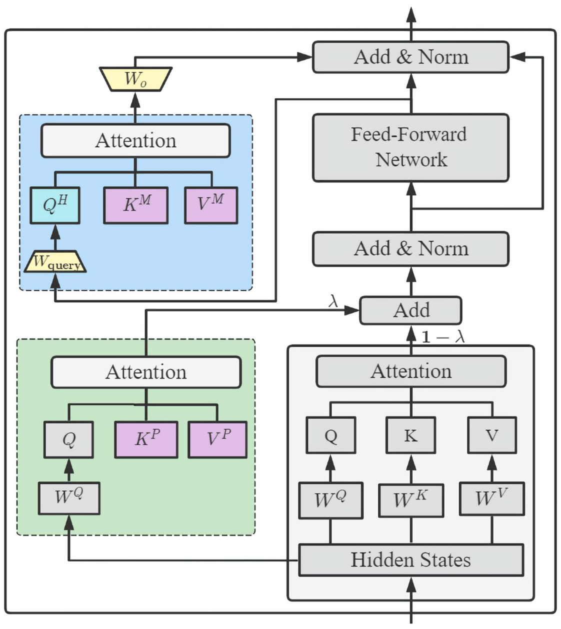


Fig. 1. Overview of the memory-tuning framework.

Others: There are several other parameter-efficient tuning methods such as BitFit [21], which only tunes bias vectors of PLMs, and Diff-pruning [22], which learns a sparse parameter update vector.

Remark: All the hidden representations would be influenced by modifying the ones before them, therefore parameter-efficient methods aiming at tuning particular representations output by MHA or FFN are able to transfer PLMs to downstream NLP tasks, such as prefix-tuning [9], adapters [6], [7], [8], [23] and LoRA [13]. Recent results of unified methods [15], [16] have shown that tuning representations in different locations simultaneously with corresponding parameter-efficient tuning methods brings performance gain. However, these unified methods only make use of some existing parameter-efficient tuning methods and do not explicitly model important domain knowledge for downstream tasks. In contrast, we propose a novel tuning method named *memory-tuning* for both MHA and FFN in each Transformer block, which is a genuine unified method for parameter-efficient transfer learning of PLMs. Moreover, memory-tuning is able to learn and store task-specific knowledge, which helps it outperform existing parameter-efficient baseline methods.

III. A Unified Parameter-Efficient Tuning Method

In this section, we introduce memory tuning, a novel parameter-efficient tuning approach designed to capture and store task-specific knowledge through external memory. This external memory can be seamlessly integrated into both the multi-head attention (MHA) and the fully connected feed-forward network (FFN) within each Transformer block, establishing memory tuning as a unified method. Furthermore, we demonstrate that prefix-tuning is, in essence, a variant of memory tuning. This revelation highlights the broader applicability of our method, especially given that prefix-tuning is not adaptable to FFN.

A. The Memory-Tuning Method

We address the challenge of storing task-specific knowledge in PLMs by integrating external memories into the MHA and FFN components of Transformer blocks. By keeping the original parameters of PLMs frozen, we propose that task-specific knowledge can be efficiently stored in these external memories of PLMs during the transfer to any downstream NLP tasks. The external memory consists of N memory slots and is denoted as $\boldsymbol{M} \in \mathbb{R}^{N\times d_m}$. With two projection matrices $\boldsymbol{W}_{\text{key}} \in \mathbb{R}^{d_m\times d_m}$, $\boldsymbol{W}_{\text{value}} \in \mathbb{R}^{d_m\times d_m}$, we convert memory slots into key and value pairs $\boldsymbol{K}^M \in \mathbb{R}^{N\times d_m}$, $\boldsymbol{V}^M \in \mathbb{R}^{N\times d_m}$.[1] To modify n hidden representations $\boldsymbol{H} \in \mathbb{R}^{n\times d}$, where d is the model dimension of Transformer, the memory-tuning first projects it to query vectors $\boldsymbol{Q}^H \in \mathbb{R}^{n\times d_m}$ via the projection matrix $\boldsymbol{W}_{\text{query}} \in \mathbb{R}^{d\times d_m}$. Then, $\Delta\boldsymbol{H}$ is computed as[2]

$$\begin{aligned}\Delta\boldsymbol{H} &= \boldsymbol{W}_o^\top \text{Attention}(\boldsymbol{Q}^H, \boldsymbol{K}^M, \boldsymbol{V}^M)\\ &= \boldsymbol{W}_o^\top \text{Softmax}(\boldsymbol{Q}^H {\boldsymbol{K}^M}^\top)\boldsymbol{V}^M \qquad (8)\\ s.t. &\begin{cases}\boldsymbol{Q}^H = \boldsymbol{H}\boldsymbol{W}_{query}\\ \boldsymbol{K}^M = \boldsymbol{M}\boldsymbol{W}_{key}\\ \boldsymbol{V}^M = \boldsymbol{M}\boldsymbol{W}_{value}\end{cases},\end{aligned}$$

where $\boldsymbol{W}_o \in \mathbb{R}^{d_m\times d}$ projects the dimension of attention result back to the model dimension of Transformer and thus can be omitted when $d_m = d$. With $\Delta\boldsymbol{H}$, memory-tuning updates $\boldsymbol{H}$ with (7). Specifically, by setting $a = b = 1$ and letting $\boldsymbol{H}$ be the output of FFN, memory-tuning can be applied directly to the tuning of hidden representations of FFN with (7) and (8). In this light, the memory-tuning is characterized by learning representation updates with its external memory. As to MHA, we will demonstrate subsequently that prefix-tuning is indeed a kind of memory-tuning.

Remark: It is worth noting that the dimension of memory-tuning d_m can be set to a very small number, so the additional parameters introduced are almost negligible compared to the total parameters of the PLM. This assures that the memory-tuning is a parameter-efficient tuning method.

B. Prefix-Tuning is Memory-Tuning of MHA

Equation (5) describes how prefix-tuning allows original tokens to attend to l "virtual tokens" prepended to the keys and values of MHA in each Transformer block. Here, we provide an equivalent formulation of (5) and demonstrate how prefix-tuning is indeed a kind of memory-tuning; that is:[3]

$$\begin{aligned}\text{head} = \text{Attention}(&\boldsymbol{q}\boldsymbol{W}^{\boldsymbol{Q}},\\ &\text{Concat}(\boldsymbol{K}^{\boldsymbol{p}}, \boldsymbol{C}\boldsymbol{W}^{\boldsymbol{K}}),\\ &\text{Concat}(\boldsymbol{V}^{\boldsymbol{p}}, \boldsymbol{C}\boldsymbol{W}^{\boldsymbol{V}}))\end{aligned}$$

[1]Nonlinear projections such as fully connection feed-forward networks can also be used instead of linear projection matrices.

[2]Here we apply regular attention instead of the scaled one used in multi-head attention.

[3]Without loss of generalization, we ignore the scaling factor of Softmax for easy of notation.

$$\begin{aligned}&= \text{Softmax}(\boldsymbol{q}\boldsymbol{W}^{\boldsymbol{Q}}\text{Concat}(\boldsymbol{K}^{\boldsymbol{p}}, \boldsymbol{C}\boldsymbol{W}^{\boldsymbol{K}}))\\ &\quad\times \begin{bmatrix}\boldsymbol{V}^{\boldsymbol{p}}\\ \boldsymbol{C}\boldsymbol{W}^{\boldsymbol{V}}\end{bmatrix}\\ &= (1-\lambda(\boldsymbol{q}))\text{F}(\cdot) + \lambda(\boldsymbol{q})\text{G}(\cdot) \qquad (9)\end{aligned}$$

where

$$\begin{aligned}\text{F}(\cdot) &= \text{Softmax}(\boldsymbol{q}\boldsymbol{W}^{\boldsymbol{Q}}\boldsymbol{W}^{\boldsymbol{K}^\top}\boldsymbol{C}^\top)\boldsymbol{C}\boldsymbol{W}^{\boldsymbol{V}}\\ &= \underbrace{\text{Attention}(\boldsymbol{q}\boldsymbol{W}^{\boldsymbol{Q}}, \boldsymbol{C}\boldsymbol{W}^{\boldsymbol{K}}, \boldsymbol{C}\boldsymbol{W}^{\boldsymbol{V}})}_{\text{original attention}},\\ \text{G}(\cdot) &= \text{Softmax}(\boldsymbol{q}\boldsymbol{W}^{\boldsymbol{Q}}\boldsymbol{K}^{\boldsymbol{P}^\top})\boldsymbol{V}^{\boldsymbol{P}}\\ &= \underbrace{\text{Attention}(\boldsymbol{q}\boldsymbol{W}^{\boldsymbol{Q}}, \boldsymbol{K}^{\boldsymbol{P}}, \boldsymbol{V}^{\boldsymbol{P}})}_{\text{memory-tuning}},\end{aligned}$$

and $\lambda(\boldsymbol{q})$ is a scalar that represents the sum of normalized attention weights on the prefixes:

$$\lambda(\boldsymbol{q}) = \frac{\kappa}{\kappa + \chi}$$

with

$$\begin{cases}\kappa = \sum_i \exp(\boldsymbol{q}\boldsymbol{W}^{\boldsymbol{Q}}\boldsymbol{K}^{\boldsymbol{P}^\top})_i\\ \chi = \sum_j \exp(\boldsymbol{q}\boldsymbol{W}^{\boldsymbol{Q}}\boldsymbol{W}^{\boldsymbol{K}^\top}\boldsymbol{C}^\top)_j\end{cases}.$$

Note that the term Attention$(\boldsymbol{q}\boldsymbol{W}^{\boldsymbol{Q}}, \boldsymbol{C}\boldsymbol{W}^{\boldsymbol{K}}, \boldsymbol{C}\boldsymbol{W}^{\boldsymbol{V}})$ in (9) is the standard attention without prefix, while the subsequent term Attention$(\boldsymbol{q}\boldsymbol{W}^{\boldsymbol{Q}}, \boldsymbol{K}^{\boldsymbol{P}}, \boldsymbol{V}^{\boldsymbol{P}})$ has the same form as (8) if we take prefixes $\boldsymbol{K}^{\boldsymbol{P}}$, $\boldsymbol{V}^{\boldsymbol{P}}$ as the key-value pairs $\boldsymbol{K}^M$, $\boldsymbol{V}^M$ in the external memory. In this light, by setting a and b in (7) to $1-\lambda(\boldsymbol{q})$ and $\lambda(\boldsymbol{q})$ respectively, prefix-tuning is actually tuning the representation of MHA with an external memory, which is just a kind of memory-tuning.

Remark: We theoretically demonstrate that the prefix-tuning method is actually a form of memory tuning applied to MHA. This insight further underscores our method's broader applicability, as prefix-tuning does not extend to FFN. Till now we have demonstrated the proposed memory-tuning can be applied to both primary components in each Transformer block. To our best knowledge, it is the first parameter-efficient tuning method that is inherently unified for transfer learning of PLMs.

IV. Experiments

We validate the effectiveness of the proposed method on eight benchmark datasets, two types of downstream tasks, with the presence of five state-of-the-art baselines. Hereinafter, we use "MT-M", "MT-F" and "MT-MF" to denote memory-tuning of multi-head attention (MHA), memory-tuning of fully connected feed-forward network (FFN) and memory-tuning of both MHA and FFN, respectively. Since the prefix-tuning is essentially a kind of memory-tuning of MHA (see Section III-B), we omit the results of "MT-M" to avoid redundancy and only provide the results of "MT-F" and "MT-MF".

A. General Setup

Task Setup: To evaluate the proposed memory-tuning method comprehensively, we conduct both *cross-task* and *cross-scale* experiments. For cross-task validation, we adopt both sentence-level and token level tasks. Specifically, for sentence-level tasks, five datasets are selected from two benchmarks named General Language Understanding Evaluation (GLUE) [24] and SuperGLUE [25]. They cover three types of natural language understanding tasks, inlcuding sentiment analysis (SST-2), similarity and paraphrase tasks (MRPC), and natural language inference (QNLI, RTE, CB). As to token-level tasks, three datasets are all about the named entity recognition task, including CoNLL2003 [26], CoNLL2004 [27] and MSRA (in Chinese) [28]. For brevity, we use CN3 and CN4 to denote CoNLL2003 and CoNLL2004, respectively. The cross-scale validation requires the memory-tuning to be verified on PLMs of different scales. Specifically, the same experiments are conducted on both BERT$_{\text{base}}$ and BERT$_{\text{large}}$ [1], where BERT$_{\text{base}}$ is a relatively small PLM as compared with BERT$_{\text{large}}$.

Baseline Methods: Five state-of-the-art parameter-efficient tuning methods are adopted as baseline methods, including adapter-tuning (adapter) [7], prefix-tuning (prefix) [11], LoRA [13], Bitfit [21] and MAM Adapter (MAM) [15]. Full fine-tuning (FT) is also adopted to setup a top baseline.

Parameter Alignment: We align the stored amount of additional parameters of different methods to ensure a fair comparison, which is accomplished by setting hyperparameters. Detailed calculation of trainable/stored parameters for each parameter-efficient methods in both size of BERT models are shown in Table II. Specifically, for prefix-tuning, the hyperparameter to be adjusted is its the length l of prefix vector while the bottleneck dimension m does not have an impact to the amount of stored parameters after training. For adapter, we adjust the intermediate bottleneck dimension r. For MT-F, we adjust the size of each memory slot s while the number of memory slots k has only very few impact on the amount of parameters stored after training. By setting these hyperparameters to the same small value, i.e., $l = r = s = 32$ in our experiments, we align the tunable amount of parameters of prefix-tuning, adapter-tuning and MT-F. For unified methods, we require the sum of hyperparameters to adjust is the same. Specifically, for MAM, we set $l + r = 32$. As to MT-MF, we set $l + s = 32$. To balance each module in the unified method, we set $l = r = s = 16$ in our experiments.

Implementation Details: We conduct experiments on two NVIDIA GeForce RTX 3090 GPUs. The results are evaluated by different measures as suggested by different tasks. To reduce the interference of randomness, the experiments are repeated three times and the average scores along with the standard deviations are returned as results. According to the recorded experience [6], [9], [15], [23], the learning rates of FT and other methods are set to $2e^{-5}$ and $1e^{-4}$, respectively. For fair comparisons, the common hyperparameters are adjusted according to the statistical characteristics of datasets and are kept consistent for all methods, as listed in Table I. The remaining specific hyperparameters of baseline methods are set to their default values by following the

TABLE I
SETTINGS OF COMMON HYPER-PARAMETERS FOR ALL METHODS

	Sentence-Level Task					**Token-Level Task**		
Hyperparameter	**MRPC**	**RTE**	**CB**	**QNLI**	**SST-2**	**CN3**	**CN4**	**MSRA**
Epoch	50	50	50	50	50	30	30	30
Input length	100	100	300	100	100	100	100	60
Batch Size (BERT$_{\text{large}}$)	16	16	16	64	64	64	8	64
Batch Size (BERT$_{\text{base}}$)	16	16	16	128	128	128	8	128

TABLE II
NUMBER OF TUNED AND STORED PARAMETERS OF TUNING METHODS AND HOW HYPER-PARAMETERS OF BERT$_{\text{LARGE}}$ DIFFER FROM THOSE OF BERT$_{\text{BASE}}$

Method	#Tuned parameters	#Stored parameters	BERT$_{\text{base}}$	BERT$_{\text{large}}$ [1]
Full FT	$(V+2+n)d+(12d^2+13d)L$	$(V+2+n)d+(12d^2+13d)L$	$V=30522, d=768, L=12, n=512$	$d=1024, L=24$
BitFit	$d+11dL$	$d+11dL$	$d=768, L=12$	$d=1024, L=24$
Adapter	$2drL$	$2drL$	$d=768, r=32, L=12$	$d=1024, L=24$
LoRA	$4drL$	$4drL$	$d=768, r=16, L=12$	$d=1024, L=24$
Prefix Tuning	$ld+(2md+2d)L$	$2dlL$	$d=768, l=32, L=12, m=768$	$d=1024, L=24, m=1024$
MAM Adapter	$ld+(2md+2d)L+2drL$	$2drL+2ldL$	$d=768, r=16, l=16, L=12, m=768$	$d=1024, L=24, m=1024$
MT-F	$(2rd+3r^2+kr)L$	$(2rd+3r^2+kr)L$	$d=768, s=32, k=50, L=12$	$d=1024, L=24$
MT-MF	$(2rd+3r^2+kr+2md+2d)L+ld$	$(2rd+3r^2+kr+2dl)L$	$d=768, s=16, l=16, k=50, L=12$	$d=1024, L=24$

[1] We present only the hyper-parameters in which BERT$_{\text{large}}$ differs from BERT$_{\text{base}}$.
In the original BERT models, the vocabulary size is denoted by V, the hidden dimension by d, the maximum sequence length by n, and the number of layers by L.
r denotes the bottleneck size of adapter. k and s denote for number of memory slots and size of each memory slot respectively. l denotes the length of prefix in prefix-tuning.

TABLE III
RESULTS WITH BERT$_{\text{LARGE}}$

			Sentence-Level Task					**Token-Level Task**			**Average**	
Method	**#Tuned**	**#Stored**	**MRPC**	**RTE**	**CB**	**QNLI**	**SST-2**	**CN3**	**CN4**	**MSRA**	**Score**	**Rank**
FT	100%	100%	$86.8_{0.3}$	$56.4_{1.1}$	$83.5_{3.2}$	$91.7_{0.0}$	$\mathbf{94.4}_{0.7}$	$88.3_{0.5}$	$77.2_{1.4}$	$90.7_{0.5}$	$\underline{83.6}_{0.4}$	$\underline{3.5}_{0.3}$
Prefix [11]	15.10%	0.47%	$\mathbf{88.2}_{0.2}$	$51.0_{0.7}$	$84.2_{3.9}$	$\underline{91.8}_{0.2}$	$94.0_{0.3}$	$\underline{88.5}_{0.4}$	$\underline{80.7}_{0.2}$	$90.2_{0.5}$	$\underline{83.6}_{0.6}$	$3.8_{0.1}$
Adapter [7]	0.47%	0.47%	$80.6_{0.7}$	$52.1_{0.2}$	$71.5_{1.2}$	$90.8_{0.2}$	$93.7_{0.4}$	$88.3_{0.8}$	$74.0_{6.6}$	$\underline{90.8}_{0.1}$	$80.2_{1.1}$	$5.8_{0.3}$
LoRA [13]	0.47%	0.47%	$87.4_{1.0}$	$55.5_{1.3}$	$77.2_{2.1}$	$91.2_{0.1}$	$93.8_{0.2}$	$88.2_{0.1}$	$76.4_{0.4}$	$\underline{90.8}_{0.2}$	$82.2_{0.4}$	$4.6_{0.2}$
Bitfit [21]	0.08%	0.08%	$86.1_{0.7}$	$54.0_{1.1}$	$74.3_{0.9}$	$90.1_{0.0}$	$93.7_{0.1}$	$83.7_{0.3}$	$59.3_{3.1}$	$80.3_{0.6}$	$77.7_{0.2}$	$6.7_{0.5}$
MAM [15]	15.32%	0.47%	$85.9_{1.7}$	$56.3_{2.6}$	$\underline{84.7}_{2.3}$	$91.5_{0.2}$	$\underline{94.2}_{0.5}$	$88.0_{0.1}$	$78.1_{1.0}$	$89.8_{0.9}$	$\underline{83.6}_{0.7}$	$4.1_{0.4}$
MT-F	0.50%	0.50%	$86.9_{1.2}$	$\mathbf{57.1}_{2.3}$	$76.2_{2.5}$	$90.8_{0.5}$	$93.5_{0.2}$	$87.6_{0.4}$	$77.0_{1.0}$	$90.2_{0.7}$	$82.4_{0.5}$	$5.5_{0.2}$
MT-MF	15.33%	0.48%	$\underline{88.0}_{0.8}$	$\underline{56.8}_{2.1}$	$\mathbf{85.0}_{1.3}$	$\mathbf{91.9}_{0.2}$	$\underline{94.2}_{0.1}$	$\mathbf{88.7}_{0.3}$	$\mathbf{82.0}_{0.6}$	$\mathbf{90.9}_{0.0}$	$\mathbf{84.7}_{0.6}$	$\mathbf{2.0}_{0.3}$

We report the average score with the standard deviation as the subscript. The best and 2nd best methods on each dataset are in bold and underlined, respectively.

TABLE IV
RESULTS WITH BERT$_{\text{BASE}}$

			Sentence-Level Task					**Token-Level Task**			**Average**	
Method	**#Tuned**	**#Stored**	**MRPC**	**RTE**	**CB**	**QNLI**	**SST-2**	**CN3**	**CN4**	**MSRA**	**Score**	**Rank**
FT	100%	100%	$83.7_{3.8}$	$54.8_{0.8}$	$\mathbf{88.2}_{3.4}$	$89.1_{0.5}$	$\underline{92.1}_{0.1}$	$\mathbf{88.2}_{0.4}$	$\underline{78.9}_{1.5}$	$\mathbf{92.3}_{0.2}$	$\mathbf{83.4}_{0.4}$	$\underline{3.4}_{0.4}$
Prefix [11]	13.08%	0.54%	$\mathbf{87.0}_{0.6}$	$\underline{56.5}_{1.4}$	$81.9_{3.4}$	$\mathbf{90.2}_{0.1}$	$91.9_{0.4}$	$\underline{87.9}_{0.2}$	$78.5_{1.7}$	$88.7_{0.5}$	$82.8_{0.4}$	$3.8_{0.9}$
Adapter [7]	0.54%	0.54%	$84.6_{2.7}$	$54.6_{0.5}$	$78.5_{5.2}$	$89.6_{0.3}$	$\mathbf{92.5}_{0.6}$	$87.2_{0.2}$	$78.2_{1.0}$	$88.6_{0.2}$	$81.7_{0.8}$	$4.8_{0.7}$
LoRA [13]	0.54%	0.54%	$85.8_{1.1}$	$54.8_{0.5}$	$81.4_{3.6}$	$\underline{90.0}_{0.1}$	$91.6_{0.2}$	$\underline{87.9}_{0.4}$	$77.6_{0.5}$	$89.0_{0.4}$	$82.3_{0.7}$	$4.8_{0.6}$
Bitfit [21]	0.09%	0.09%	$83.6_{2.2}$	$54.7_{0.8}$	$72.7_{1.1}$	$87.8_{0.1}$	$\underline{92.1}_{0.3}$	$80.4_{0.3}$	$65.8_{0.5}$	$77.3_{0.2}$	$76.8_{0.2}$	$6.9_{0.2}$
MAM [15]	13.34%	0.54%	$84.8_{3.0}$	$\mathbf{57.0}_{2.3}$	$78.9_{4.6}$	$89.9_{0.5}$	$91.9_{0.3}$	$87.8_{0.9}$	$77.6_{2.4}$	$89.7_{0.6}$	$82.2_{0.7}$	$3.9_{0.6}$
MT-F	0.59%	0.59%	$86.6_{0.7}$	$54.9_{1.3}$	$76.2_{3.9}$	$88.6_{0.4}$	$91.6_{0.5}$	$87.4_{0.6}$	$76.4_{1.0}$	$89.1_{0.1}$	$81.3_{0.5}$	$5.6_{0.3}$
MT-MF	13.35%	0.56%	$\underline{86.9}_{0.8}$	$\mathbf{57.0}_{1.3}$	$\underline{82.3}_{2.0}$	$\underline{90.0}_{0.5}$	$92.0_{0.2}$	$\mathbf{88.2}_{0.3}$	$\mathbf{79.6}_{0.6}$	$\underline{90.6}_{0.3}$	$\underline{83.3}_{0.4}$	$\mathbf{2.8}_{0.6}$

We report the average score with the standard deviation as the subscript. The best and 2nd best methods on each dataset are in bold and underlined, respectively.

corresponding literature or codes. The early stopping strategy is adopted for training to avoid overfitting.

B. Main Results

We report the comparative results based on BERT$_{\text{large}}$ and BERT$_{\text{base}}$ in Tables III and IV, respectively. Note that both the average values (the larger the better) and ranks (the smaller the better) over all datasets are reported as metrics for comprehensive evaluation.

As can be seen from Table III for BERT$_{\text{large}}$, our MT-MF method generally achieves the state-of-the-art performance by outperforming other parameter-efficient tuning methods and even full fine-tuning. Specifically, MT-MF obtains two first places and three second places on the five datasets of the sentence-level task, and performs even better by beating all baseline methods on the three datasets of the token-level task. Similar results can be found from Table IV for BERT$_{\text{base}}$, MT-MF performs the best by the average rank and the second best by the average score (full fine-tuning is slightly better). These results verify the effectiveness of employing parameter-efficient tuning and demonstrate the advantage of MT-MF from learning task-specific knowledge explicitly for transfer learning of PLMs.

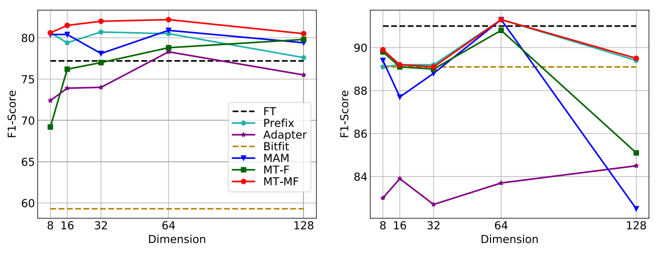
We then compare the performances between MT-MF and two baselines: MT-F and Prefix. MT-F is the memory tuning only for the FFN component, and Prefix is proved as a kind of memory tuning but only for the MHA component. As can be seen from Tables III and IV, Prefix generally performs better than MT-F, but both are inferior to MT-MF. These results indicate that the nature of unified tuning for both MHA and FFN components is important to the success of MT-MF, although the introduction of external memory seems more effective for MHA than FFN. This also illustrates why MAM as a unified tuning method can beat the three non-unified tuning baselines: Adapter, LoRA, Bitfit and MT-F. Nevertheless, it is worth noting that MAM yet performs worse than MT-MF, given that its unification depends on assembling the existing tuning methods. In contrast, MT-MF is a truly unified tuning method that employs a same memory attention structure for both MHA and FFN components in each Transformer block.

It is also interesting to find from Tables III and IV that MT-MF seems to perform better in token-level tasks. Note that sentence-level tasks merely require to tune the representations of "CLS" tokens, but token-level tasks require to tune the representations of all the tokens, which makes them much more fine-grained and domain knowledge related as compared with sentence-level tasks. Therefore, the promising results of MT-MF in the token-level tasks confirm the necessity of learning task-specific knowledge and the advantage of truly unified tuning for modeling semantic relations between tokens.

From Tables III and IV, we also note that both FT and Adapter register an improved performance on smaller PLMs. This phenomenon may be attributed to the fact that FT and Adapter experience reduced overfitting issues when applied to smaller PLMs. This observation indicates that our proposed Memory-Tuning method offers more significant advantages when used with larger-scale PLMs. Given the current trend towards PLMs with vast numbers of parameters and the intrinsic goal of parameter-efficient tuning methods, which seek to provide lightweight fine-tuning for these large-scale models, our method holds considerable promise for practical applications.

C. Robustness Analysis With Varying Number of Tunable Parameters

Setup: Here, we study the effect of modifying the amount of tunable parameters of PLMs for all the comparative methods. All the experiments are conducted with BERT$_{\text{large}}$. The datasets MPRC and CoNLL2004 are chosen to cover both the sentence- and token-level tasks. As it is infeasible to submit considerable runs to the GLUE leaderboard (2 submissions per day), we record the best F1 score of each model on the validation set of MRPC. As to CoNLL2004, we record the F1 score on its test set.



(a) CoNLL2004 (b) MRPC

Fig. 2. Varying number of tunable parameters.

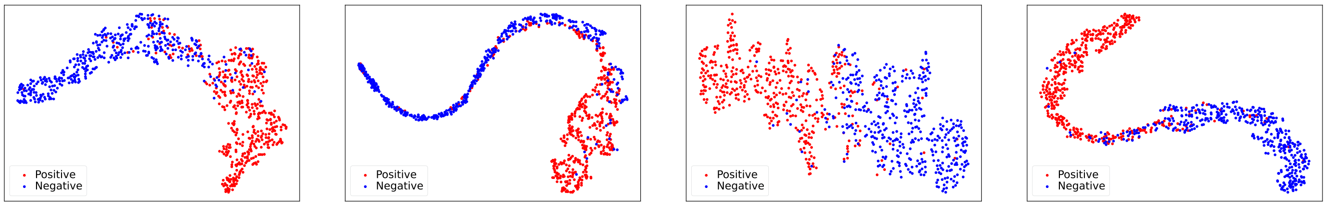
By tuning the hyperparameters described in Section IV-A, we can control the amount of tunable parameters. Specifically, for Prefix and Adapter, we vary l and r from 8 to 128, respectively. For MAM, we follow the literature to set the prefix tuning with a small dimension and to allocate more parameter budgets to the scaled parallel adapter. That is, we keep $l = s$ when varying $l + r$ from 8 to 32, and keep $l = 8$ when r is varied from 64 to 128.[4] As to our MT-MF, we keep $l = s$ when varying $l + s$ from 8 to 128. The results of all methods with varying amount of tunable parameters on two datasets are illustrated in Fig. 2.

Results: Fig. 2(a) illustrates the results on CoNLL2004. As can be seen, our proposed MT-MF consistently outperforms baseline methods with varying amount of tunable parameters. Moreover, the performance of MT-MF is much more stable than other methods, demonstrating the robustness of MT-MF when applied to transfer learning of fine-grained NLP tasks. In contrast, the other unified method MAM is less stable, implying that the inherently unified memory-tuning can tune FFN and MHA in each Transformer block more harmoniously. An interesting phenomenon is that MT-MF consistently outperforms MT-F and Prefix, which again indicates the memory-tuning of FFN and MHA are complementary to each other. The results on MRPC are shown in Fig. 2(b), from which we can find that MT-MF and Prefix achieve similar performances and outperform other methods. In general, the results on both sentence- and token-level tasks illustrate the robustness of MT-MF given its unified tuning nature. Note that the performance of most tuning methods decreases as the dimension rises from 64 to 128. A possible explanation for this might be that as the dimension of the adjustable parameters in the model increases, the model complexity consequently escalates. When the data volume is relatively small or the difficulty of the task is low, the model is likely to overfit, leading to the observed performance degradation.

D. Visualization of Memory Vectors With Task Related Knowledge

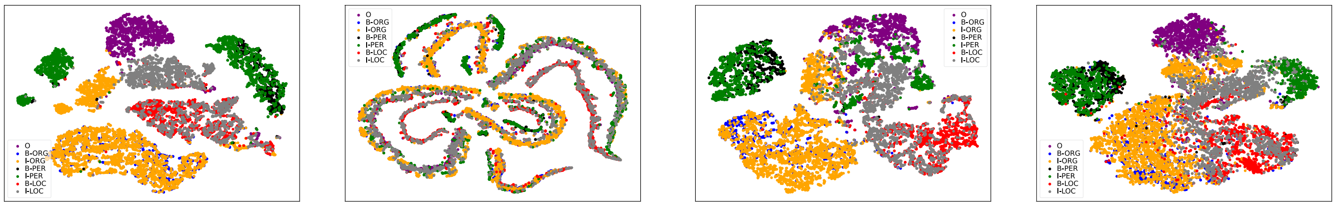
Setup: We further explore whether the external memory introduced for MHA and FFN by memory-tuning can indeed learn and store task-specific knowledge. Along this line, we visualize and observe the pattern of attention scores over the

[4]This is a solution that considers both the approximate parameter alignment with other methods and following the original dimension allocation approach of MAM. It actually allows the amount of tunable parameters of MAM to be slightly larger than that of other methods.



(a) Attention scores of MT-M (prefix).
(b) Attention scores of MT-F.
(c) Attention scores of MHA's memory in MT-MF.
(d) Attention scores of FFN's memory in MT-MF.

Fig. 3. Visualization of attention scores on SST-2.



(a) Attention scores of MT-M (prefix).
(b) Attention scores of MT-F.
(c) Attention scores of MHA's memory in MT-MF.
(d) Attention scores of FFN's memory in MT-MF.

Fig. 4. Visualization of attention scores on MSRA (NER task).

external memory, i.e., the Softmax result in (8) for FFN and the second Softmax result in (9) for MHA. Two datasets namely SST-2 and MSRA are adopted to cover both the sentence- and token-level tasks. To obtain attention scores, we first train MT-M (i.e., prefix), MT-F and MT-MF on the datasets. Then, during model inference, we store attention scores over the external memory for MHA or FFN in the last block of the PLM. To visualize the attention scores, we first apply PCA [29] to reduce the dimension from 32 (MHA) or 50 (FFN) to 10, and then use t-SNE [30] to further reduce the dimension from 10 to 2. Four sets of attention scores are illustrated for each dataset, which are obtained from the last MHA's external memory in MT-M, the last FFN's external memory in MT-F, and the external memory of the last MHA and FFN in MT-MF.

Results: The visualization results on SST-2 are illustrated in Fig. 3, where the attention scores from positive (negative) sentences are colored in red (blue) as SST-2 is a sentiment classification dataset. The visualizations of MT-M, MT-F and MT-MF consistently reveal that the distribution of attention scores is closely related to the sentiment polarities of sentences. This promising result indicates that the representations obtained from external memory are attached to sentence polarities and thus are able to identify task-specific knowledge in sentence-level tasks.

MSRA is a named entity recognition dataset, where each token has one of seven labels named “B-ORG”, “I-ORG”, “B-PER”, “I-PER”, “B-LOC”, “I-LOC” and “O”. Except for label “O” that indicates non-entity tokens, the other six labels indicate which type of entity the token belongs to, e.g., the suffix “ORG” represents organization, and also indicates the position of a token in the entity, e.g., the prefix“B” represents the starting of an entity. Fig. 4 illustrates the attention scores of the tokens in MSRA, where each token is colored according to its label. As can be seen, apart from the results of MT-F, the distributions of attention scores of the tokens are highly correlated with their corresponding labels. Specifically, the attention scores of the tokens with a particular type of named entity label (i.e., the labels with a same suffix) are gathered into a cluster, and each cluster is further divided into two sub-clusters with two different prefix labels. The attention scores of tokens with label “O” form an isolated cluster. The above results indicate that MT-MF and MT-M can identify fine-grained task-specific knowledge in token-level tasks. The results of MT-F do not show obvious patterns, which indeed coincides with the observation in Section IV-B; that is, for token-level tasks especially those that rely on modeling semantic relations between tokens, memory-tuning of MHA rather than FFN is essential to task-specific knowledge learning. But with the help of the MT-M module, the MT-F module in MT-MF also successfully learns the fine-grained task-specific knowledge according to Fig. 4(d), which again justifies the advantage of truly unified tuning of MT-MF.

E. Data Efficiency

Setup: Here, we explore performance of our proposed Memory-Tuning under low-data settings. Specifically, we compare our method with all baselines described in Section IV-A when training data is limited. For low-data settings, we use $\text{BERT}_{\text{Large}}$ as the base PLM and sample a small subset of the training data for each task with size $N \in \{128, 256, 512\}$, which covers cases from extreme low-data to normal low-data. To reduce variance, we sample training data with 3 random seeds and report the average performance.

Results: The results under low-data settings are presented in Table V, where we can find that although Full-Tuning (FT) achieves the best performance when available training data is quite limited, MT-MF performs the best among existing parameter-efficient tuning methods. Above promising results indicate Memory-Tuning has better data efficient against other parameter-efficient tuning methods.

TABLE V
RESULTS WITH VARYING TRAINING SIZE N

Methods	MRPC	RTE	CB	QNLI	SST2	CN3	CN4	MSRA	Avg	Order
					N=128					
FT	80.2	**53.6**	**79.4**	**65**	79.5	**60.1**	**67.2**	**76.2**	**70.2**	**2.6**
Prefix	80.5	51.7	78.6	54.6	60.8	37.8	54.6	75.2	61.7	5.0
Adapter	80.3	52	69.6	58.3	80.5	37.1	53.6	74.9	63.3	6.0
LoRA	80.3	52.1	72.1	58.3	80.8	45.7	53.9	75.6	64.9	4.2
Bitfit	75.4	47.3	67.4	53.9	62.8	39.2	53.8	67.6	58.4	7.4
MAM	80.3	52.2	78.8	57.8	81	57.4	53.9	75.3	67.1	3.6
MT-F	80.3	51.8	77.4	54.7	80.3	38.3	54.1	75.1	64.0	5.1
MT-MF	**80.6**	52.2	79.2	57.5	**81.2**	57.8	54.2	75.7	67.3	2.3
					N=256					
FT	82.1	**53.2**	**83.6**	71.4	**84.2**	**66.4**	**78.7**	**84.2**	**75.5**	**2.0**
Prefix	**82.3**	53	82.1	**75.6**	81.7	43	69.6	83.5	71.4	3.7
Adapter	80.8	52.1	71.4	69	83.9	45.7	67.8	84.1	69.4	5.7
LoRA	81.6	52.2	74.9	69.7	84.1	59.3	69.4	84.0	71.9	4.5
Bitfit	76.6	48.6	68.2	67.3	80.3	40.7	62.5	80.1	65.5	7.9
MAM	**82.3**	52.8	76.8	71.8	83.8	60.7	69.7	83.7	72.7	3.3
MT-F	81.9	51.9	70.1	68.1	82.4	44.9	67.9	82.2	68.7	6.1
MT-MF	82.2	52.5	78.9	70.7	84.1	62.5	70.2	83.7	73.1	2.9
					N=512					
FT	82.2	**56.6**	**84.4**	**77.3**	**85.2**	**81.8**	**81**	**86.8**	**79.4**	**1.6**
Prefix	81.9	53.5	75	76.9	83.3	67.8	75.1	85.4	74.9	4.9
Adapter	81.8	53.6	71.4	73.9	84.3	73.6	74	85.7	74.8	6.0
LoRA	81.9	54.0	79.5	74.3	84.5	76.6	78.7	85.6	76.9	4.3
Bitfit	77.9	49.2	70.1	71.3	83.1	58.7	72.1	83.1	70.7	7.9
MAM	81.9	54.1	83.9	75.8	84.1	77.4	80	86.2	77.9	3.4
MT-F	82	52.9	72.2	74.1	83.7	68.1	74.2	85.3	74.1	5.7
MT-MF	**82.4**	54.2	83.8	75.8	84.6	78.9	80.3	86.1	78.3	2.3

The best and 2nd best methods on each dataset are in bold and underlined, respectively.

TABLE VI
INFERENCE LATENCY OF DIFFERENT METHODS

	FT	Prefix	Adapter	LoRA	BitFit	MAM	MT-F	MT-MF
BERT$_{\text{large}}$	100	112	106	105	99	108	107	108
BERT$_{\text{base}}$	100	110	106	104	101	106	107	107

F. Inference Latency

Setup: In order to compare the inference lantency between our method and earlier ones, we calculate from saved logs of running time during inference. Then, in comparison to Full-Tuning (FT), we report the relative inference times. This includes the average time costs for eight NLU datasets for BERT in Tables IV and III. The time cost of FT is normalized to 100.

Results: As shown in Table VI, FT and BitFit share the least inference time over all methods since there are no extra tuned parameters added into original PLM architecture. The inference latency is similar between Adapter and our MT-F, slightly inferior to LoRA. As unified frameworks, MAM and our MT-MF performs comparable and cost more inference time than the aforementioned non-unified methods. Finally, prefix-tuning takes the longest inference time over all methods since the computational complexity is $O(n^2)$ in MHA, where n refers to the length of prefix vectors.

G. Performance on Generation Tasks

Setup: To validate the general effectiveness of our proposed method across wide range NLP tasks, we conduct experiment on generation tasks by comparing our MT-MF with prefix-tuning[5] and the performance of Full-tuning is also added as a reference. Specifically, we evaluate methods on two benchmarks namely E2E [31] and WebNLG [32], which are Table-to-Text generation datasets and are also used in the paper of prefix-tuning. The E2E dataset contains about 50K examples from 8 distinct fields, with multiple test references for each source table, and the average output length is 22.9. We use the official evaluation script to calculate BLEU [33], NIST [34], METEOR [35], ROUGE-L [36], and CIDEr [37]. The WebNLG dataset consists of 22K examples, and its input is a sequence of (subject, property, object) triples. The average output length is 22.5. In the training and validation data, each input describes entities from 9 distinct DBpedia[6] categories (e.g., Monument). The test data consists of two parts: the first half contains DB categories observed in training data, and the second half contains 5 unobserved categories. These unobserved categories are used to evaluate extrapolation [9]. We use the official evaluation script to calculate BLEU, METEOR and TER [38]. We use both GPT-2$_{\text{Medium}}$ and GPT-2$_{\text{Large}}$ [17] as base PLMs, where the input token length

[5]Prefix-tuning is proposed for downstream text generation tasks.
[6]https://www.dbpedia.org

TABLE VII
RESULTS ON GENERATION TASKS

Method	E2E					WebNLG								
	BLEU$\uparrow$	NIST$\uparrow$	METEOR$\uparrow$	ROUGE-L$\uparrow$	CIDEr$\uparrow$	BLEU-S$\uparrow$	BLEU-U$\uparrow$	BLEU-A$\uparrow$	MET-S$\uparrow$	MET-U$\uparrow$	MET-A$\uparrow$	TER-S$\downarrow$	TER-U$\downarrow$	TER-A$\downarrow$
GPT-2 Medium														
FT	70.3	8.78	46.1	71.9	2.46	61.8	45.1	54.3	0.44	0.36	0.41	0.35	0.48	0.41
Prefix	69.6	8.75	45.5	71.7	2.44	61.2	44.7	53.8	0.41	0.36	0.4	0.37	0.52	0.43
MT-MF	70.1	8.77	46.1	71.9	2.46	61.6	44.9	54.2	0.44	0.36	0.41	0.35	0.49	0.41
GPT-2 Large														
FT	71.2	8.84	46.3	72.0	2.47	65.2	46.4	55.9	0.46	0.38	0.42	0.33	0.47	0.40
Prefix	70.5	8.79	45.8	71.8	2.45	63.8	45.3	55.2	0.43	0.37	0.41	0.34	0.50	0.42
MT-MF	71.1	8.82	46.2	71.9	2.46	64.6	46.1	55.7	0.45	0.38	0.42	0.33	0.48	0.41

is fixed to 12 and the max generation length is fixed to 30 for both E2E and WebNLG, according to above described average output length.

Results: The results of text generation are reported in Table VII. Our proposed MT-MF performs comparable with Full-Tuning in experimental generation tasks and MT-MF consistently outperforms prefix-tuning on two benchmarks according to all evaluation measures in both GPT-2_{Medium} and GPT2_{Large}. Moreover, the extent to which our proposed MT-MF outperforms prefix-tuning is more pronounced on GPT2_{Large} compared to GPT-2_{Medium}. This aligns with the conclusions drawn from results comparing Table IV with Table III, collectively indicating that our method holds greater promise for application in large-scale LLMs. The above promising results indicate simultaneously tuning FFN and MHA with memory-tuning can help PLMs achieve better performance on text generation tasks.

V. CONCLUSION

In this paper, we propose *memory-tuning*, a parameter-efficient tuning method that introduces external memory in a unified way to both the MHA and FFN components of Transformer blocks, for improving transfer learning of PLMs for downstream tasks. We further prove that the well-known prefix-tuning is a kind of memory-tuning. We conduct experiments on eight data sets under cross-task and cross-scale configurations, and our proposed method consistently outperforms the state-of-the-art baselines including full fine-tuning and the existing unified method. The subsequent parameter alignment study validates the robustness of memory-tuning, and the visualization study of attention scores justifies the ability of memory-tuning in identifying and storing task-related knowledge.

REFERENCES

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

[2] W. Han, B. Pang, and Y. N. Wu, “Robust transfer learning with pretrained language models through adapters,” in *Proc. 59th Annu. Meeting Assoc. Comput. Linguistics 11th Int. Joint Conf. Natural Lang. Process. (Vol. 2: Short Papers)*, 2021, pp. 854–861, doi: 10.18653/v1/2021.acl-short.108.

[3] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang, “Pre-trained models for natural language processing: A survey,” *Sci. China Technological Sci.*, Springer, vol. 63, no. 10, pp. 1872–1897, 2020.

[4] M. E. Peterset al., “Deep contextualized word representations,” in *Proc. North Amer. Chapter Assoc. Comput. Linguistics: Hum. Lang. Technol.*, 2018, pp. 2227–2237.

[5] Z. M. Ziegler, L. Melas-Kyriazi, S. Gehrmann, and A. M. Rush, “Encoder-agnostic adaptation for conditional language generation,” *CoRR*, vol. abs/1908.06938, 2019. [Online]. Available: http://arxiv.org/abs/1908.06938

[6] N. Houlsby et al., “Parameter-efficient transfer learning for NLP,” in *Proc. Int. Conf. Mach. Learn.*, Long Beach, California, USA, 2019, vol. 97, pp. 2790–2799. [Online]. Available: http://proceedings.mlr.press/v97/houlsby19a.html

[7] J. Pfeiffer, A. Kamath, A. Rücklé, K. Cho, and I. Gurevych, “Adapter-fusion: Non-destructive task composition for transfer learning,” in *Proc. 16th Conf. Eur. Chapter Assoc. Comput. Linguistics: Main Vol.*, 2021, pp. 487–503. [Online]. Available: https://aclanthology.org/2021.eacl-main.39/

[8] R. Heet al., “On the effectiveness of adapter-based tuning for pretrained language model adaptation,” in *Proc. 59th Annu. Meeting Assoc. Comput. Linguistics 11th Int. Joint Conf. on Natural Lang. Process. (Vol. 1: Long Papers)*, Aug. 2021, pp. 2208–2222, doi: 10.18653/v1/2021.acl-long.172.

[9] X. L. Li and P. Liang, “Prefix-tuning: Optimizing continuous prompts for generation,” in *Proc. 59th Annu. Meeting Assoc. Comput. Linguistics 11th Int. Joint Conf. on Natural Lang. Process. (Vol. 1: Long Papers)*, 2021, pp. 4582–4597, doi: 10.18653/v1/2021.acl-long.353.

[10] X. Liuet al., “GPT understands, too,” *AI Open*, Elsevier, vol. 5, pp. 208–215, 2024.

[11] X. Liu et al., “P-tuning V2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks,” *ACL*, pp. 61–68, 2022.

[12] B. Lester, R. Al-Rfou, and N. Constant, “The power of scale for parameter-efficient prompt tuning,” in *Proc. Conf. Empirical Methods Natural Lang. Process., Virtual Event*, Punta Cana, Nov., 2021, pp. 3045–3059. [Online]. Available: https://aclanthology.org/2021.emnlp-main.243

[13] E. J. Hu et al., “Lora: Low-rank adaptation of large language models,” in *Proc. Int. Conf. Learn. Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=nZeVKeeFYf9

[14] A. Vaswaniet al., “Attention is all you need,” in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6000–6010.

[15] 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. [Online]. Available: https://openreview.net/forum?id=0RDcd5Axok

[16] Y. Maoet al., “UniPELT: A unified framework for parameter-efficient language model tuning,” *ACL*, Dublin, Ireland: Association for Computational Linguistics, pp. 6253–6264, May 2022.

[17] A. Radford et al., “Language models are unsupervised multitask learners,” *OpenAI blog*, vol. 1, no. 8, 2019.

[18] C. Raffelet al., “Exploring the limits of transfer learning with a unified text-to-text transformer,” *J. Mach. Learn. Res.*, vol. 21, pp. 1–67, 2020.

[19] M. Lewiset al., “BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension,” in *Proc. Assoc. Comput. Linguistics*, 2020, pp. 7871–7880.

[20] L. Donget al., “Unified language model pre-training for natural language understanding and generation,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 13042–13054.

[21] E. B. Zaken, Y. Goldberg, and S. Ravfogel, “Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models,” in *Proc. 60th Annu. Meet. Assoc. Comput. Linguistics*, Dublin, Ireland: Association for Computational Linguistics: Short Papers, vol. 2, May 2022, pp. 1–9.

[22] D. Guo, A. M. Rush, and Y. Kim, “Parameter-efficient transfer learning with diff pruning,” in *Proc. Assoc. Comput. Linguistics 2021*, 2021, pp. 4884–4896.

[23] J. Pfeifferet al., “Adapterhub: A framework for adapting transformers,” in *Proc. Empirical Methods Natural Lang. Process.*, 2020, pp. 46–54, doi: 10.18653/v1/2020.emnlp-demos.7.

[24] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, “GLUE: A multi-task benchmark and analysis platform for natural language understanding,” in *Proc. Int. Conf. Learn. Representations*, New Orleans, LA, USA, May 2019. [Online]. Available: https://openreview.net/forum?id=rJ4km2R5t7

[25] A. Wanget al., "Superglue: A stickier benchmark for general-purpose language understanding systems," in *Proc. Adv. Neural Inf. Process. Syst.*, Vancouver, BC, Canada, 2019, pp. 3261–3275.
[26] E. F. T. K. Sang and F. D. Meulder, "Introduction to the conll-2003 shared task: Language-independent named entity recognition," in *Proc. 7th Conf. Natural Lang. Learn.,* Edmonton, Canada, 2003, pp. 142–147. [Online]. Available: https://aclanthology.org/W03-0419/
[27] X. Carreras and L. Màrquez, "Introduction to the conll-2004 shared task: Semantic role labeling," in *Proc. 8th Conf. Comput. Natural Lang. Learn., Boston, MA, USA,* 2004, pp. 89–97. [Online]. Available: https://aclanthology.org/W04-2412/
[28] G. Levow, "The third international chinese language processing bakeoff: Word segmentation and named entity recognition," in *Proc. 5th Workshop Chin. Lang. Process.,* Sydney, Australia, 2006, pp. 108–117. [Online]. Available: https://aclanthology.org/W06-0115/
[29] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley Interdiscipl. Rev.: Comput. Statist.*, vol. 2, no. 4, pp. 433–459, 2010.
[30] L. V. der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. 11, pp. 2579–2605, 2008.
[31] J. Novikova, O. Dušek, and V. Rieser, "The E2E dataset: New challenges for end-to-end generation," in *Proc. 18th Annu. SIGdial Meet. Discourse Dialogue*, Saarbrücken, Germany: Association for Computational Linguistics, Aug. 2017, pp. 201–206.
[32] C. Gardent, A. Shimorina, S. Narayan, and L. Perez-Beltrachini, "The WebNLG challenge: Generating text from RDF data," in *Proc. 10th Int. Conf. Natural Lang. Gener.*, 2017, pp. 124–133.
[33] K. Papineni, S. Roukos, T. Ward, and W. Zhu, "Bleu: A method for automatic evaluation of machine translation," in *Proc. Assoc. Comput. Linguistics*, 2002, pp. 311–318.
[34] A. Belz and E. Reiter, "Comparing automatic and human evaluation of NLG systems," in *Proc. 11th Conf. Eur. Chapter Assoc. Comput. Linguistics*, 2006, pp. 313–320.
[35] A. Lavie and A. Agarwal, "METEOR: An automatic metric for MT evaluation with high levels of correlation with human judgments," in *Proc. Workshop Stat. Mach. Transl.,* 2007, pp. 228–231.
[36] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Proc. Workshop Text Summarization Branches Out, Post-Conf.*, Barcelona, Spain, 2004, pp. 74–81.
[37] R. Vedantam, C. L. Zitnick, and D. Parikh, "Cider: Consensus-based image description evaluation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 4566–4575.
[38] M. Snover, B. Dorr, R. Schwartz, L. Micciulla, and J. Makhoul, "A study of translation edit rate with targeted human annotation," in *Proc. 7th Conf. Assoc. Mach. Transl. Amer.: Tech. Papers*, Cambridge, Massachusetts, USA, Aug. 2006, pp. 223–231.

Wang Qi received the master's degree from Beihang University, Beijing, China, in 2022. He is currently a research Engineer with Zhejiang Lab, Hangzhou, China. His research interests include natural language processing, topic-to-essay generation and parameter-efficient tuning of pre-trained language models.

Rui Liu received the Ph.D. degree from Beihang University, Beijing, China, in 2011. He is currently a Professor with the State Key Laboratory of Software Development Environment, School of Computer Science and Engineering, Beihang University. He is currently with the National Engineering Research Center for Science and Technology Resource Sharing Service of China Portal as a Chief Engineer. His work has been published in conference proceedings and refereed journals, including ICDM, CIKM, IJCAI and IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING. His research interests include database management, data mining and natural language processing.

Yuan Zuo received the Ph.D. degree from Beihang University, Beijing, China, in 2017. He is currently an Associate Professor with the Information Systems Department, Beihang University, and a member of the MIIT Key Laboratory of Data Intelligence and Management. His research has been published in refereed journals and conferences, including IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, NeurIPS, and ACM SIGKDD. His research interests include data mining and machine learning, with a particular focus on representation learning, graph neural networks, and large language models.

Fengzhi Li received the bachelor's degree from Beihang University, Beijing, China, in 2018. He is currently working toward the Ph.D. degree with the School of Economics and Management, Beihang University. His research interests include dynamic graph learning, text classification, and large language model.

Yong Chen received the Ph.D. degree in computer science and engineering from Beihang University, Beijing, China, in 2019. From 2019 to 2021, he was a âBoyaâ Postdoc with the Key Lab of Machine Perception, School of EECS, Peking University, Beijing, China. He is currently an Associate Professor with the School of Computer Science, Beijing University of Posts and Telecommunications, Beijing, China. From 2018 to 2019, he was a visiting Ph.D. Student with UCL and Birkbeck. His research interests include machine learning, data mining, and numerical optimization.

Junjie Wu received the B.E. degree in civil engineering and the Ph.D. degree in management science and engineering from Tsinghua University, Beijing, China. He is currently a Full Professor with the School of Economics and Management, Beihang University, Beijing, China. He is also the Director of Research Center for Data Intelligence and Management, and the Director of the Institute of Artificial Intelligence for Management. He has published prolifically on UTD journals, IEEE Transaction journals and ACM Conferences. His research interests include data science with applications in social, urban and financial areas.