LoRETTA: Low-Rank Economic Tensor-Train Adaptation for Ultra-Low-Parameter Fine-Tuning of Large Language Models

Yifan Yang[1*], **Jiajun Zhou**[2*†], **Ngai Wong**[2] **and Zheng Zhang**[1]
[1]University of California, Santa Barbara
[2]The University of Hong Kong
yifanyang@cs.ucsb.edu, {jjzhou,nwong}@eee.hku.hk, zhengzhang@ece.ucsb.edu

Abstract

Various parameter-efficient fine-tuning (PEFT) techniques have been proposed to enable computationally efficient fine-tuning while maintaining model performance. However, existing PEFT methods are still limited by the growing number of trainable parameters with the rapid deployment of Large Language Models (LLMs). To address this challenge, we present LoRETTA, an ultra-parameter-efficient framework that significantly reduces trainable parameters through tensor-train decomposition. Specifically, we propose two methods, named LoRETTA$_{adp}$ and LoRETTA$_{rep}$. The former employs tensorized adapters, offering a high-performance yet lightweight approach for the fine-tuning of LLMs. The latter emphasizes fine-tuning via weight reparameterization with a set of small tensor factors. LoRETTA achieves comparable or better performance than most widely used PEFT methods with up to $100\times$ fewer parameters on the LLaMA-2-7B models. Furthermore, empirical results demonstrate that the proposed methods exhibit remarkable anti-overfitting capability, effectively improve training efficiency, and enjoy better multi-task learning performance. Plug-and-play loretta library built upon the Huggingface framework and PEFT library are provided.[‡]

1 Introduction

The BERT and LLaMA families (Devlin et al., 2018; Touvron et al., 2023; Floridi and Chiriatti, 2020), representing the prevailing paradigm of Large Language Models (LLMs), showcase remarkable task generalization capabilities in diverse applications, from dialogue systems to question-answering, summarization and translation. While LLMs exhibit proficiency in following instructions and learning task solutions with minimal contextual input, their accuracy can be further enhanced through fine-tuning techniques.

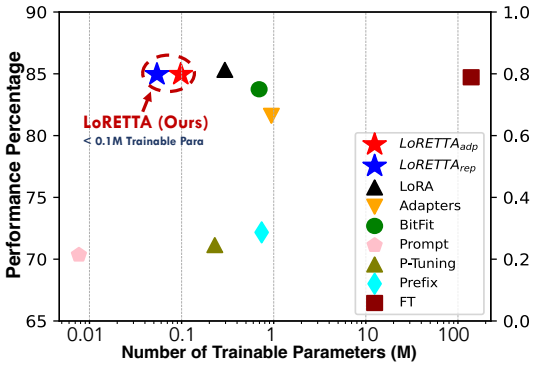

Figure 1: The performance vs. trainable parameters on the DeBERTa-Base, showcasing the relationship between parameter efficiency and performance across various GLUE tasks.

Since full-model fine-tuning becomes infeasible as the model size of LLMs grows rapidly, there has been increased interest in parameter-efficient fine-tuning (PEFT) (Hu et al., 2023). PEFT methods fine-tune LLMs by modifying only a subset of parameters. The concept was initially explored in (Houlsby et al., 2019), which proposes the Adapters method to inject trainable modules into the transformer encoders. Based on this concept, the LoRA approach (Hu et al., 2021) adds low-rank updating matrices on the weights of linear projection layers in the self-attention blocks. These two types of methods achieve similar or even better performance than full-model fine-tuning, but still incur a large number of trainable parameters. Taking the LLaMA-2-70B model as an example, LoRA needs to update over 16 million parameters, which is even more than the total parameters of some BERT models. Moreover, we observe that both the Adapters and LoRA approaches experience significant overfitting problems, detrimentally affecting their overall performance.

[*]Equal contributions
[†]Work undertaken during the visit at UC Santa Barbara
[‡]Code available at: https://github.com/yifanycc/loretta

In contrast, other methods like prefix tuning (Li and Liang, 2021) and prompt tuning (Lester et al., 2021) introduce trainable tokens to the input or hidden layers of the base model, significantly reducing trainable parameters but potentially sacrificing accuracy, especially in few-shot learning scenarios (Mao et al., 2022). Furthermore, (Aghajanyan et al., 2020) achieves approximately 90% of the full fine-tuning performance with only 200∼800 parameters on a RoBERTa model by exploring the intrinsic dimension, which is far less than the 0.3 million parameters needed in the LoRA method (Hu et al., 2021). Despite LoRA's ability to outperform full-model fine-tuning, its number of trainable parameters is still too high, motivating our exploration of more economic and efficient high-performance PEFT approaches. This raises the question: *Is there a PEFT approach with ultra-low trainable parameters that still performs on-par or better than full-model fine-tuning?*

In this paper, we present **Lo**w-**R**ank **E**conomic **T**ensor-**T**rain **A**daptation (LoRETTA), which is tailored for efficient fine-tuning of variously scaled LLMs with minimal trainable parameters. Our approach leverages the tensor-train (TT) format to represent large weight matrices. LoRETTA encompasses two variants: LoRETTA$_{adp}$ and LoRETTA$_{rep}$. The LoRETTA$_{adp}$ variant embeds tensorized adapters in encoder/decoder layers and performs better than *all* PEFT methods under equivalent trainable parameter sizes. The LoRETTA$_{rep}$ variant, our ultra-efficient innovation, requires substantially fewer trainable parameters, occupies less than 1MB of storage, and maintains comparable performance. Our contributions are threefold:

- LoRETTA is proposed that utilizes tensor-train format to effectively fine-tune LLMs with up to $100\times$ fewer trainable parameters than widely used PEFT methods like Adapters and LoRA on the LLaMA-2 model.
- Our proposed framework demonstrates better performance to other widely used PEFT methods across various scales of models, tasks, and setups, particularly excelling in generation tasks with large-scale models.
- Comprehensive studies are conducted against other PEFT methods regarding storage/computation efficiency, anti-overfitting ability, forgetting risks for multi-task learning, and performance under different setups.

2 Background

2.1 Parameter-Efficient Fine-Tuning

Except for the aforementioned Adapters, LoRA, and prompt-based approach, there exist various PEFT-related works (Li and Liang, 2021; Lester et al., 2021; Hyeon-Woo et al., 2021; Liu et al., 2023; Tian et al., 2023), including the BitFit method (Zaken et al., 2022) that tries to further reduce trainable parameters by only fine-tuning the bias term. However, it is observed that BitFit suffers from a considerable performance drop, which is also shown in our experiments. Furthermore, there are large-scale models like LLaMA that do not employ any bias terms in the model structure, which makes the utilization of the BitFit method restricted. Compared with these previous methods, the proposed LoRETTA is efficient and versatile, making it applicable to any kind of language model, offering a seamless and lightweight plug-and-play solution for fine-tuning.

2.2 Tensor-based Model Compression

Over the past decade, tensor compression has emerged as a promising technique for reducing model size and both inference and training times (Lebedev et al., 2015; Kim et al., 2015). For example, (Novikov et al., 2015) proposed the idea of the TT format by representing the weight matrix with a series of tensor factors. (Hawkins et al., 2022; Hawkins and Zhang, 2021) presented an end-to-end compressed training approach with automatic rank determination for various tensor formats. Despite these advancements, the application of the tensorized approach to the fine-tuning of LLMs is limited, primarily due to the complex, high-rank structure of pretrained weights.

An exception to this trend is the work of (Liu et al., 2021), which proposed a tensorized fine-tuning approach by only updating parts of the tensor factors. Nevertheless, it still requires over 10% of the model parameters for effective fine-tuning. Researchers in (Jie and Deng, 2023), instead, tried to stack all weight matrices of the Vision Transformer (ViT) into a single weight tensor and create a tensorized updating tensor following the idea of LoRA. However, its applicability to LLMs is hindered by the extremely large stacked tensor,

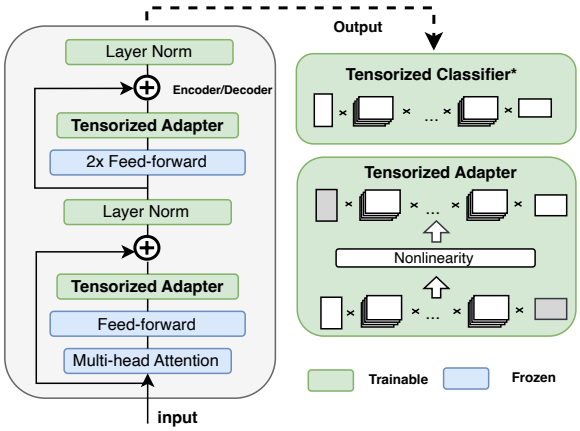

Figure 2: Architecture of LoRETTA$_{adp}$ for the transformer encoders or decoders. $*$ the tensorized classifier is optional for different tasks. For classification tasks, we set this part to be trainable and we freeze this part during language modeling tasks.

which, for the LLaMA-2-7B model, reaches 7 billion parameters for this single variable.

3 LoRETTA Method

PEFT methods can be broadly categorized into three types, the adapters, the reparameterization method, and the prompt-based method (Hu et al., 2023). Among them, the reparameterization-based and adapter-based methods are notable for incorporating new structures within the model architecture, thereby introducing a large number of additional trainable parameters. To reduce the size of the injected modules, we introduce our LoRETTA framework, which contains the adapter-based approach LoRETTA$_{adp}$ and the reparameterization-based approach LoRETTA$_{rep}$. Subsequent sections will delve into the intricacies of the tensorized layer, followed by an in-depth exploration of the LoRETTA$_{adp}$ and LoRETTA$_{rep}$ structures.

3.1 Tensorized TT Layer

We devise the modules in LoRETTA$_{adp}$ and LoRETTA$_{rep}$ based on tensorized layers, where we first reshape the weight matrix in the linear layer into a tensor and then employ the TT format to reduce the number of model parameters. Specifically, TT (Oseledets, 2011) decomposes a large tensor into a set of small tensor factors. Unlike traditional linear layers that involve training large weight matrices, we only store and train the small TT factors during the fine-tuning process. Consequently, considering a fully connected layer with an input vector of $\boldsymbol{x} \in \mathbb{R}^N$, the forward pass can be expressed as $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$, where $\boldsymbol{W} \in \mathbb{R}^{M\times N}$ is the weight matrix, and $\boldsymbol{b}$ is the bias vector.

In a tensorized layer, the matrix $\boldsymbol{W}$ is first reshaped into a tensor $\mathcal{W} \in \mathbb{R}^{k_1\times\cdots\times k_d}$, where $\prod_{i=1}^{d} k_i = M \times N$. Then, the reshaped weight tensor $\mathcal{W}$ can be effectively represented by TT-format using a set of tensor factors $\mathcal{G}_1, \cdots, \mathcal{G}_i, \cdots, \mathcal{G}_d$ with the shape of $\mathcal{G}_i \in \mathbb{R}^{r_{i-1}\times k_i\times r_i}, i \in [1, d]$. Then, for the d dimension tensor $\mathcal{W}$ and a sequence of value $(a_1, \cdots, a_d)$ for each dimension, the element $\mathcal{W}(a_1, \cdots, a_d)$ can be calculated with a given set of TT rank $[r_0, \cdots, r_d]$:

$$\mathcal{W}(a_1, \cdots, a_d) = \boldsymbol{G}_1^{a_1} \cdots \boldsymbol{G}_i^{a_i} \cdots \boldsymbol{G}_{\boldsymbol{d}}^{\boldsymbol{a_d}} \quad (1)$$

where $\boldsymbol{G}_i^{a_i} := \mathcal{G}_i(:, a_i, :) \in \mathbb{R}^{r_{i-1}\times r_i}$ is a slice of each tensor factors with the same shape of $r_{i-1} \times r_i$. By setting the first and last TT-ranks as $r_0 = r_d = 1$, we can obtain the value for an element in $\mathcal{W}$ by doing the matrix multiplication among the slice of each tensor factor.

Since the matrices $\boldsymbol{G}_i^{a_i}$ are stacked into the tensor factor $\mathcal{G}_i$, the original weight matrix $\boldsymbol{W}$ can also be written by the TT representation, which reshapes the product of all the tensor factors:

$$\text{TT}(\boldsymbol{W}) := \prod_{i=1}^{d} \mathcal{G}_i[r_{i-1}, k_i, r_i], \quad (2)$$

where $\mathcal{G}_i[r_{i-1}, k_i, r_i]$ means for the i-th tensor factor $\mathcal{G}_i$ with the size of $r_{i-1} \times k_i \times r_i$.

As we can see, the tensorized layer substantially reduces the parameter count for the weight matrix $\boldsymbol{W}$ from the original $M \times N$ to $\sum_{i=1}^{d} r_{i-1}k_i r_i$. Thus, the compression ratio is closely linked to the choice of TT ranks. For simplicity, we fix all ranks $r_i, \forall i \in [1, d-1]$ to be the constant. However, adaptive rank adjustments during training, as discussed in (Hawkins et al., 2022), may further enhance the performance of the LoRETTA framework. In the following, we elaborate on how to utilize this tensorized layer in the LoRETTA$_{adp}$ and LoRETTA$_{rep}$ methods.

3.2 Lightweight Tensorized Adapters

LoRETTA$_{adp}$ is inspired by the ultra-low "intrinsic dimension" of the language models (Aghajanyan et al., 2020). This idea has been utilized in the

previous Adapters and LoRA methods by using the bottleneck approach. However, there still exists a large gap between trainable parameters of the current PEFT methods and the "intrinsic dimension" explored in (Aghajanyan et al., 2020). This motivates us to push this idea further. In our method, we first fine-tune the LLMs by injecting tensorized adapters, demonstrating superior performance with ultra-low trainable parameters.

The general workflow of LoRETTA$_{adp}$ is illustrated in Fig. 3. Different from the traditional Adapters method that utilizes the bottleneck structure to reduce the trainable parameters, our tensorized adapters achieve a much larger compression ratio by including two tensorized linear layers and an activation function. For example, set the hidden size of the models as 768, and the bottleneck size as 64, compared to the Adapters method with the number of trainable parameters of $2 \cdot 768 \cdot 64 \approx 98K$ for weight matrices, LoRETTA$_{adp}$ adds only $\sum_{i=1}^{6}(5^2 \cdot 8) = 1.2K$ parameters, assuming tensor shapes of $[8, 8, 8, 8, 8, 8]$ and a constant TT rank of 5. Inspired by the idea presented in (Houlsby et al., 2019), we incorporate trainable tensorized adapters following each attention and feed-forward sub-layer within the self-attention blocks.

Optimizable modules: Further to fine-tuning the tensorized adapters modules, we also investigate making the layer normalization and the last layer of networks trainable. From our observations in the Appendix B, it is obvious that fine-tuning the last layer of the models is crucial for classification tasks. However, it is a common challenge to fine-tune the last layer due to its large number of parameters in models like RoBERTa and DeBERTa. To tackle this, we employ the tensorized last layer for classification tasks in our methods, thereby achieving a significant reduction in trainable parameters while maintaining effectiveness, as evidenced in our experiments. Note that we choose to freeze the last layer for language model tasks since the parameters of the language model head are inherited from the pre-trained weight.

3.3 TT Reparameterization

Next, we propose a more compact PEFT approach by reparameterizing the weight matrix with tensor factors, dubbed LoRETTA$_{rep}$. The idea of the reparameterization also appeared in LoRA (Hu et al., 2021), which updates the weight with two low-rank matrices in a linear layer as follows:

$$y = W_0 x + \Delta W x = W_0 x + BAx \tag{3}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ denote the input and output of a linear layer. Setting h as the hidden size of the model, $W_0 \in \mathbb{R}^{h \times l}$ is a pre-trained weight matrix, $B \in \mathbb{R}^{h \times r}$ and $A \in \mathbb{R}^{r \times l}$ are low-rank matrices representing the update matrix ΔW, with $r \ll \min(h, l)$ as the LoRA rank parameter. In the original LoRA, A is initialized from a Gaussian distribution whereas B is zero, ensuring that the update part $BA = 0$ at the beginning.

However, as mentioned in the introduction, the reparameterization of weights through matrix factorization may not fully exploit the intrinsic dimension. Here, we propose a more compact way to represent the updating matrix with two tensorized layers (without bias terms) introduced in Section 3.1, whose general idea is depicted in Fig. 3. In our method, we also employ the bottleneck structure to first reduce the large updating matrix into two small matrices. Then, we reshape the two updating matrices $\Delta \boldsymbol{W}_{up}$ and $\Delta \boldsymbol{W}_{down}$ into tensors $\Delta \mathcal{W}_{up}$ and $\Delta \mathcal{W}_{down}$ with the shape of $k_1 \times \cdots \times k_d$ and $j_1 \times \cdots \times j_d$. Here, both $\Delta \mathcal{W}_{up}$ and $\Delta \mathcal{W}_{down}$ are cast into TT factors. The tensorized update process of a full-connected layer with linear transformation to an input $\boldsymbol{x}$ can be expressed as:

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{W}_0 \boldsymbol{x} + \mathrm{TT}(\Delta \boldsymbol{W}_{up}) \cdot \mathrm{TT}(\Delta \boldsymbol{W}_{down}) \boldsymbol{x} \\ &= \boldsymbol{W}_0 \boldsymbol{x} + \prod_{i=1}^{d} \mathcal{G}_i \prod_{i=1}^{d} \mathcal{Q}_i \boldsymbol{x} \end{aligned} \tag{4}$$

where $\boldsymbol{W}_0$ represent the pre-trained weight, $\Delta \boldsymbol{W}_{up}$ and $\Delta \boldsymbol{W}_{down}$ are represented as the TT layers following the TT representation in eq. (2) with tensor factors $(\mathcal{G}_1, \cdots, \mathcal{G}_d)$ and $(\mathcal{Q}_1, \cdots, \mathcal{Q}_d)$ in the TT layers. In our implementation, we use the tensorized layer mentioned ahead, but without the bias term to perform the tensorized linear transformation in the second term of eq. (4). In this manner, our approach reduces the parameters from 12K to 1K for a single reparameterization adapter compared with the LoRA method with the LoRA rank of 8, when the hidden size is 768 and the tensor rank is 5 for the LoRETTA$_{rep}$ method.

Initialization: As noted before, LoRA starts

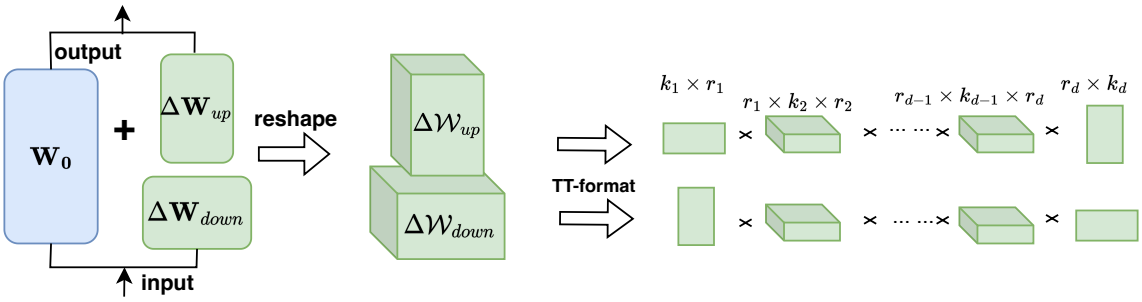

Figure 3: Architecture of the LoRETTA$_{rep}$ method for a single transformer encoder.

with $B = 0$, making the initial model outputs identical to pre-reparameterization. However, our proposed method requires optimizing each tensor factor. Compared with the LoRA method, which only contains two factors for each weight matrix, assigning a value of zero to one of the numerous tensor factors can more readily lead to optimization challenges due to zero gradient issues. To overcome this issue, we initiate the process with a tensor reconstruction (Kolda and Bader, 2009) step at the beginning of the training process. This step involves converting the list of tensor factors back into a matrix form. Following this, we compute the mean of the reconstructed matrix to evaluate the noise introduced by Gaussian initialization, and subsequently mitigate the noise from the pre-trained weight.

4 Experiment

We conduct comprehensive experiments for the performance of LoRETTA on the downstream task for the LLMs with different scales. Specifically, we present the results on both BERT-family (RoBERTa-base (Liu et al., 2019) and DeBERTa-base (He et al., 2020)) models and the large-scale LLaMA-2 models (Touvron et al., 2023). We first show that LoRETTA frameworks perform on par or better than other PEFT methods (like BitFit, LoRA, Adapters, and Prefix tuning, etc.) with fewer trainable parameters across different model types, sizes, and tasks, especially on the LLaMA-2 models. Then, we discuss some observations of the strong ability of LoRETTA in multi-task learning and addressing overfitting issues. Further experiments demonstrate that the LoRETTA method can help to reduce the memory storage, training FLOPs, and improve the memory copy efficiency. Finally, we also carry out the tensor rank analysis of our approach to show the applicability of LoRETTA with even fewer trainable parameters. All experiments utilize the AdamW optimizer (Loshchilov and Hutter, 2018), and similar learning rate and batch size set up for different methods (See Appendix A for details). We use NVIDIA Tesla V100-16GB and A100-40GB for experiments.

Compared Methods. Our exploration covers both full-model fine-tuning (FT) and PEFT methods like Adapters (Ding et al., 2023), BitFit (Zaken et al., 2022), LoRA (Hu et al., 2021), Prefix-tuning(Li and Liang, 2021), Prompt-tuning (Lester et al., 2021), P-tuning (Liu et al., 2022b) and IA3 (Liu et al., 2022a). To ensure a fair and easier comparison, we implemented most PEFT methods with the Huggingface PEFT library (Mangrulkar et al., 2022) and evaluated most methods with the same learning rate, batch size, and training epochs. Furthermore, we primarily adhered to the default settings for other hyperparameters of the baseline methods, upholding consistency across all tasks for generalizability.

4.1 GLUE Experiments on the BERT Family

We initially conducted experiments on the Generalized Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018), encompassing various natural language understanding tasks. Table 1 summarizes the downstream task performance comparison between LoRETTA framework and other baseline methods. We utilize the whole training dataset for each task, collect the best validation results in every 200 training steps, and reach the following conclusions.

LoRETTA performs on-par or better than other PEFT methods. Both LoRETTA$_{adp}$ and LoRETTA$_{rep}$ consistently achieve higher average

Table 1: Comparative analysis of various PEFT methods on the BERT family models (including RoBERTa-base and DeBERTa-base models). We specifically bold the PEFT method that achieves the best results among methods with similar parameter sizes. $*$ represents results shown in previous works (Valipour et al., 2022; Zaken et al., 2022). Different from the LoRA paper (Hu et al., 2021), we use the F1 score for the MRPC and QQP tasks.

Model & Method	# Train. Param.	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
DeBERTa-Base (FT)	139.19M	88.67	94.61	91.98	59.32	93.04	91.42	68.23	91.10	84.79
DeBERTa-Base (Adapters$_{r=8}$)	0.94M	87.69	94.72	88.88	54.19	92.95	85.52	59.20	89.68	81.60
DeBERTa-Base (LoRA$_{r=8}$)	0.30M	87.30	94.95	92.84	60.56	93.35	85.19	80.14	90.13	85.56
DeBERTa-Base (P-Tuning)	0.23M	56.25	91.39	79.93	43.31	86.30	78.43	55.95	78.38	71.24
DeBERTa-Base (LoRA$_{r=4}$)	0.15M	**87.69**	94.49	91.10	62.57	92.60	87.30	69.67	91.12	84.54
DeBERTa-Base (Prompt)	0.01M	77.63	92.43	81.90	32.99	80.30	78.15	62.81	56.71	70.36
DeBERTa-Base (Prefix)	0.15M	60.32	88.87	81.22	45.82	83.28	82.22	59.57	84.99	73.28
DeBERTa-Base (BitFit)	0.10M	84.63	95.41	91.42	**64.06**	93.20	84.15	66.79	90.23	83.75
DeBERTa-Base (LoRETTA$_{adp}$)	0.10M	85.93	95.30	**93.53**	60.84	92.99	84.08	75.50	**91.32**	**84.96**
DeBERTa-Base (LoRETTA$_{rep}$)	0.05M	86.80	**95.53**	88.73	59.69	**93.25**	**89.20**	**75.81**	90.66	84.95
RoBERTa-Base (BitFit)$*$	0.10M	85.30	**94.80**	**92.33**	62.70	91.30	68.10	73.60	88.50	82.08
RoBERTa-Base (LoRA$_{r=8}$)$*$	0.63M	**86.82**	94.01	91.48	62.08	92.39	85.71	74.51	**90.48**	84.69
RoBERTa-Base (LoRETTA$_{adp}$)	0.10M	85.61	94.38	91.08	**62.70**	92.12	**87.22**	**78.70**	90.26	**85.26**
RoBERTa-Base (LoRETTA$_{rep}$)	0.07M	84.40	94.28	90.63	61.72	**92.40**	85.23	74.42	89.24	84.04

scores on the GLUE tasks versus PEFT methods with lower than 0.2M trainable parameters, like LoRA, Prefix/Prompt tuning, P-tuning, and BitFit methods. Compared to LoRA with $3\times$ more trainable parameters, LoRETTA$_{adp}$ outperforms across 4 of 8 tasks and attains a similar average performance (with nearly 0.5% difference). Similarly, LoRETTA$_{rep}$ reduces parameters by $6\times$ with just an average score gap within 0.6%.

LoRETTA performs well across different BERT models. For fair comparison, we also include LoRA and BitFit results on the RoBERTa-base model reported in (Valipour et al., 2022; Zaken et al., 2022), which sets the last layer to be trainable. We observe that LoRETTA$_{adp}$ outperforms LoRA, with a substantial $7\times$ reduction in trainable parameters. The results also highlight LoRETTA performs much better than the BitFit on the RoBERTa-base model, showing our advantages over other PEFT methods across various models, alongside its robust generalization capabilities.

4.2 Large-Scale Language Models

Building upon the encouraging results achieved with DeBERTa/RoBERTa models, we expanded the application of LoRETTA to the LLaMA-2 models. The results are summarized in Table 2 and Table 3. To raise the difficulty of experiments, we use low data resource settings for both SuperGLUE tasks (Wang et al., 2019) and generation tasks about question answering (SQuAD (Rajpurkar et al., 2016), DROP (Dua et al., 2019)). For each

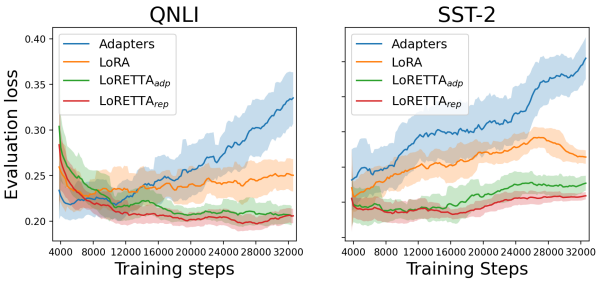

Figure 4: Evaluation loss comparison across various PEFT methods on the DeBERTa-base model. The loss is smoothed with a window size of 20 and the shallow means the standard deviation boundaries.

task, we randomly selected 1000, 500, and 1000 examples for training, validation, and testing. All classification tasks in the SuperGLUE benchmark have been transferred to language modeling tasks following the prompt-based fine-tuning strategy used in (Malladi et al., 2023). Our observations are summarized as follows.

LoRETTA performs better or on-par compared with other widely used PEFT methods with up to $100\times$ trainable parameters reduction. LoRETTA$_{adp}$ shows superior performance across most tasks compared to *all* parameter-efficient fine-tuning methods. Compared with LoRA or the Adapter methods, LoRETTA$_{adp}$ achieves better performance in up to 7 tasks with nearly $5\times$ and $56\times$ reduction of trainable parameters. Even compared with full model fine-tuning, our method still outperforms in 5 of 7 tasks. Furthermore,

Table 2: Performance Comparison on LLaMA-2-7B with low data resource setting (1000 examples). LoRETTA$_{adp}$ outperforms other widely used PEFT methods among most tasks.

Model & Method	Train. Param.	Classfication			Multiple Choice		Generation	
		CB	BoolQ	WSC	COPA	ReCoRD	SQuAD	DROP
LLaMA2-7B (FT)	6738.42M	66.07	84.6	63.46	86	81.1	**90.71**	51.38
LLaMA2-7B (Adapter)	50.33M	66.07	71.8	62.50	84	78.8	88.45	49.14
LLaMA2-7B (LoRA$_{r=8}$)	4.19M	**67.86**	84.8	62.50	81	79.4	90.56	45.96
LLaMA2-7B (Prefix)	1.31M	51.78	78.6	61.53	83	81.0	90.56	45.95
LLaMA2-7B (IA3)	0.60M	64.29	72.3	36.53	80	**81.5**	89.41	39.37
LLaMA2-7B (LoRETTA$_{rep}$)	0.51M	55.35	78.1	57.61	86	80.3	88.47	42.71
LLaMA2-7B (LoRETTA$_{adp}$)	0.88M	66.07	**87.0**	**63.46**	**87**	80.0	90.17	**51.60**

Table 3: Performance Comparison on LLaMA-2-13B and LLaMA-2-70B. We compare our proposed method with LoRA, which is one of the most widely used high-performance PEFT methods.

Model & Method	LLaMA-2-13B					LLaMA-2-70B		
	Param.	COPA	ReCoRD	SQuAD	DROP	Param.	SQuAD	DROP
Adapters	79.05M	90	83.8	**93.37**	57.41	252.97M	93.37	68.12
LoRA$_{r=8}$	6.55M	90	83.4	92.71	59.13	16.38M	93.78	72.99
IA3	0.96M	85	84.2	91.81	51.48	2.45M	92.85	71.48
LoRETTA$_{rep}$	0.77M	86	**84.4**	90.87	53.19	1.99M	90.18	68.83
LoRETTA$_{adp}$	1.67M	**90**	83.9	92.67	**59.41**	4.79M	**94.33**	**74.50**

LoRETTA$_{rep}$ achieves comparable performance with up to $100\times$ fewer trainable parameters compared to the Adapters.

LoRETTA is working even better on 13B and 70B models. We compare the performance of our proposed method with the most widely used LoRA method over the LLaMA-2 13B and 70B models. Due to the limited computation resources, we only give the results on the more important reasoning (COPA and ReCoRD) and generation tasks (SQuAD and DROP). The results are summarized in Table 3. We can observe that our LoRETTA$_{adp}$ method outperforms the LoRA method across 5 of 6 tasks on both 13B and 70B models. In particular, the LoRETTA$_{adp}$ method achieves a reduction of nearly 12 million trainable parameters on the 70B model with over 1% accuracy improvement.

The tensorized method shows robust performance across various tasks. Beyond the classification and multi-choice tasks, we also included language generation tasks such as SQuAD and DROP, which are more intricate. It can be seen that LoRETTA$_{adp}$ continues to yield excellent results with much lower trainable parameters, especially on the large-scale LLaMA-2 13B and 70B models.

Table 4: Performance of anti-forgetting in MTL tests. The three training sets are fed sequentially during the training process and we test the validation loss for each task after the training is finished.

Model & Method	SST-2	MRPC	QNLI	Average
DeBERTa-Base(Adapters)	51.83	27.21	90.21	56.42
DeBERTa-Base(LoRA)	49.20	20.15	87.74	55.70
DeBERTa-Base(LoRETTA$_{adp}$)	**52.29**	39.22	91.52	61.01
DeBERTa-Base(LoRETTA$_{rep}$)	51.26	**52.94**	**92.15**	**65.45**

4.3 Over-fitting and Multi-Task Learning

LoRETTA method uniquely addresses overfitting and promotes multi-task learning (MTL) by reducing trainable parameters. We further explore its anti-overfitting and MTL capabilities.

Adapters and LoRA exhibit overfitting during training. We follow the experiments of SST-2 and QNLI tasks in Section 4.1 and record the curve of evaluation loss by testing the validation dataset every 200 steps. The corresponding results are in Fig. 4. It is evident from the figure that the evaluation loss for both LoRA and Adapters escalates rapidly beyond a certain point, indicating a significant over-fitting. In contrast, LoRETTA$_{adp}$ and LoRETTA$_{rep}$ show markedly improved handling of overfitting and a much more stable learning curve with less variance. That is attributed to their much fewer trainable parameters, which better retain the information captured by the pre-trained weights.

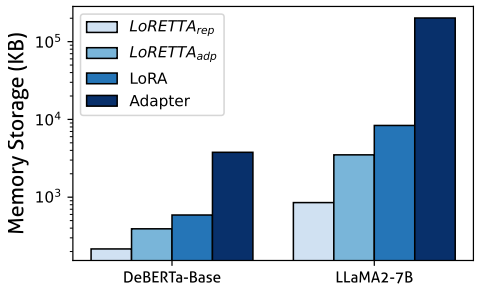

Figure 5: Comparison of memory storage for trainable parameters across different models and methods.

LoRETTA excels in MTL tasks. MTL optimizes multiple tasks using shared model parameters (Ruder, 2017). We utilize the DeBERTa-Base model and train our model with SST-2, MRPC, and QNLI training set in the GLUE benchmark sequentially. We test the accuracy with the validation set after the training of all three datasets, which can show the degree of forgetting.

The results, presented in Table 4, demonstrate that LoRETTA$_{adp}$ and LoRETTA$_{rep}$ achieve higher average test accuracy. This shows our method performs better in retaining the information in the previous training, highlighting our method as a potentially better foundational approach for fine-tuning in MTL setup. Future work could include integrating more comprehensive MTL strategies with LoRETTA , such as task clustering or task relation learning (Zhang and Yang, 2021) to achieve better performance.

4.4 Memory Performance

In Figure 5, we compare LoRETTA with prominent fine-tuning approaches, including LoRA and adapters on two types of LLMs to show that our proposed method enjoys the following key features.

Ultra-low memory storage for trainable parameters. LoRETTA$_{rep}$, our most compact PEFT method, requires only around 1MB storage for its trainable parameters, outperforming its counterparts. On DeBERTa-Base, both LoRETTA$_{rep}$ and LoRETTA$_{adp}$ (0.852MB vs 3.5MB) outperform classical baselines, reducing the trainable parameter storage by a factor of $9.6\times$ and $2.7\times$, respectively, compared to LoRA and Adapters. Ditto for LLaMA-2, where LoRETTA$_{rep}$ and LoRETTA$_{adp}$ similarly reduce the trainable parameter storage by a factor of $57.4\times$ and $9.8\times$, respectively. Such an economic storage space makes our proposed method suitable for resource-limited hardware (Wu et al., 2023), suggesting potential applications in quantized tensor models for future research.

Table 5: Memory profiling and FLOPs analysis.

Model & Method	Memcpy (us)	FLOPS (Reduction)
LLaMA2-7B(Adapter)	10590	6.18E+15(Baseline)
LLaMA2-7B(LoRA)	45674	6.145E+15(-4.2+E13)
LLaMA2-7B(LoRETTA$_{adp}$)	**9879**	**6.141E+15(-4.6+E13)**

Table 6: Tensor rank analysis on SST-2 and QNLI.

LoRETTA$_{adp}$	r=2	r=5	r=10	r=20
Train. Param.	0.067	0.098	0.206	0.627
DeBERTa-Base(SST-2)	95.41	95.30	94.84	95.41
DeBERTa-Base(QNLI)	92.04	92.99	93.50	93.34
LoRETTA$_{rep}$	r=2	r=5	r=10	r=20
Train. Param.	0.042	0.054	0.094	0.250
DeBERTa-Base(SST-2)	94.61	94.4	94.95	95.07
DeBERTa-Base(QNLI)	92.71	93.25	93.47	93.32

LoRETTA minimizes data movement overhead and reduces end-to-end training FLOPs. Considering data movement overhead during training, our method minimizes memory handling time, surpassing other PEFT methods. Overall, with $57.4\times$ less storage consumption, LoRETTA achieves comparable or superior results in memory copying time, as shown in Table 5, outperforming LoRA and Adapters. Additionally, it decreases the total floating-point operations (FLOPs) required for the fine-tuning of LLaMA2 on the SST-2 (Stanford Sentiment Treebank) task. This reduction in computational cost is accompanied by enhanced accuracy, demonstrating superior computational efficiency. Note that using some automatic CUDA optimization techniques (like *torch.compile*) can speed up the training of LoRETTA methods to a great extent due to the existence of a large number of small tensor multiplications during the training process.

4.5 Tensor Rank Analysis and Ablation Study

We first investigate the influence of different tensor ranks on our model's performance. The results are summarized in Table 6. We see that the performance for different ranks of LoRETTA approach varies across tasks. For the SST-2 task, the performance is not sensitive to the rank setting for both LoRETTA$_{adp}$ and LoRETTA$_{rep}$. However, the test accuracy drops when dealing with the QNLI task with an extra small rank.

Generally, our method performs well even under smaller ranks in some tasks, which shows the possible ability to reduce the trainable parameters under tight hardware constraints.

We also test the influence of activating the final layer and layernorm on our method. The tensorized classifier demonstrates comparable results to the regular one with a notable parameter reduction and the layernorm is shown to play a crucial role in some specific tasks. Detailed analyses are in the Appendix B.

4.6 Configuration of Tensor Shape

In this paper, we use the TT-format to represent the weight matrices in the tensorized layer. To represent a weight matrix into a list of tensor factors with shape $\mathbb{R}^{r_{i-1} \times k_i \times r_i}$ for the $i-$th factor (refer to section 3.1), we design the specific shapes for models with different hidden sizes and bottleneck setups. Presently, a standardized approach to ascertain the precise shapes of tensor factors ideal for Tensor-Train decomposition remains elusive. This process generally necessitates experimental efforts to discover the most effective configuration for the shapes of tensor factors. Here, we present an illustrative example of fine-tuning the DeBERTa-Base model with LoRETTA$_{adp}$, highlighting the procedure for selecting tensor shapes. This process involves the incorporation of tensorized layers that are characterized by an input of 768 hidden dimensions and an output size of 64.

We explored three setups $[k_1, .., k_i, ..k_d]$ for tensor factor shapes. The results are presented in Table 7, and it's evident that the shape of $[8, 8, 12, 8, 8]$ yields the best performance across the three selected tasks. Notably, the results with a shape of $[4, 4, 4, 12, 4, 4, 4]$ indicate that an excessively small dimension size for the tensor shape may lead to a significant performance drop. Hence, we chose the shape of $[8, 8, 12, 8, 8]$ in our paper. We provide detailed tensor shape setup for most widely used models, refer to Appendix A.4 and the provided code for more detail.

Table 7: Experiment results for determining the shape of the TT-format

Tensor Shape	Param.	SST-2	MRPC	QNLI
[8, 8, 12, 8, 8]	0.10M	95.30	93.53	93.25
[64, 12, 64]	0.11M	95.07	93.05	92.92
[4,4,4,12,4,4,4]	0.10M	94.72	90.72	92.86

5 Conclusion

We propose an ultra-parameter-efficient fine-tuning method, named LoRETTA , which outperforms other PEFT methods with fewer trainable parameters on LLaMA-2 models. Extensive experiments have verified that having low trainable parameters can facilitate computation and memory demands, reduce storage requirements, and enhance the ability to deal with multi-task learning/overfitting. Our proposed methods exhibit strong capabilities in both natural language understanding and generation tasks. In future work, the computation efficiency of the LoRETTA method can be further improved with other memory-efficient methods, such as FlashAttention (Dao et al., 2022) and quantization (Frantar et al., 2022).

6 Acknowledgement

This project is supported by Amazon. The authors would like to thank Siegfried Kunzmann, Athanasios Mouchtaris, Hieu Nguyen, Samridhi Choudhary, Ershad Banijamali and Clement Chung for their regular technical discussions.

Limitations

Given the extensive array of Parameter-Efficient Fine-Tuning (PEFT) methods discussed in this paper, as well as the wide range of models and tasks, the training process can become quite lengthy. Our experiments are therefore confined to testing our methods on DeBERTa, RoBERTa, and LLaMA-2 models. Notably, in the case of LLaMA-2, we adopt a low data resource setting to expedite our experiments. Future research could extend the application of our proposed method across a broader spectrum of models and tasks, leveraging the library we have made available on GitHub. Some related topics like the robustness (Yuan et al., 2024), and fairness (Li et al., 2023b) issues of the LoRETTA method can also be studied.

Another area of limitation involves the optimization of the training time and memory cost for our proposed method. At present, we utilize automatic CUDA optimization via the torch.compile function. However, a fully customized CUDA graph could potentially reduce the training duration of our meth-

ods even further. Additionally, there's scope for an extension aimed at enhancing the training efficiency and scalability of the Tensor Train (TT) format, particularly following its adaptation to low-bit quantization (Zhou et al., 2023; Ran et al., 2023).

Ethics Statement

LoRETTA provides a cost-effective solution that operates with a minimal memory footprint. This alleviates the burden on data centers and reduces CO_2 emissions. However, we acknowledge that prolonged training times, especially with multiple GPUs, can pose environmental challenges. Consequently, our ongoing research endeavors are focused on developing more efficient training methods and preserving computational power with ecological considerations in mind.

References

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. 2020. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *Machine learning challenges workshop*, pages 177–190. Springer.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. 2023. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. *arXiv preprint arXiv:1903.00161*.

Luciano Floridi and Massimo Chiriatti. 2020. Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30:681–694.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

Cole Hawkins, Xing Liu, and Zheng Zhang. 2022. Towards compact neural networks via end-to-end training: A bayesian tensor approach with automatic rank determination. *SIAM Journal on Mathematics of Data Science*, 4(1):46–71.

Cole Hawkins and Zheng Zhang. 2021. Bayesian tensorized neural networks with automatic rank selection. *Neurocomputing*, 453:172–180.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations*.

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

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Zhiqiang Hu, Yihuai Lan, Lei Wang, Wanyu Xu, Ee-Peng Lim, Roy Ka-Wei Lee, Lidong Bing, and Soujanya Poria. 2023. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*.

Nam Hyeon-Woo, Moon Ye-Bin, and Tae-Hyun Oh. 2021. Fedpara: Low-rank hadamard product for communication-efficient federated learning. *arXiv preprint arXiv:2108.06098*.

Shibo Jie and Zhi-Hong Deng. 2023. Fact: Factor-tuning for lightweight adaptation on vision transformer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 1060–1068.

Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. 2015. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*.

Tamara G Kolda and Brett W Bader. 2009. Tensor decompositions and applications. *SIAM review*, 51(3):455–500.

V Lebedev, Y Ganin, M Rakhuba, I Oseledets, and V Lempitsky. 2015. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *3rd International Conference on Learning Representations, ICLR 2015-Conference Track Proceedings*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on*

Empirical Methods in Natural Language Processing, pages 3045–3059.

Jonathan Li, Will Aitken, Rohan Bhambhoria, and Xiaodan Zhu. 2023a. Prefix propagation: Parameter-efficient tuning for long sequences. *arXiv preprint arXiv:2305.12086*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. 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)*, pages 4582–4597.

Yingji Li, Mengnan Du, Rui Song, Xin Wang, and Ying Wang. 2023b. A survey on fairness in large language models. *arXiv preprint arXiv:2308.10149*.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022a. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965.

Peiyu Liu, Ze-Feng Gao, Wayne Xin Zhao, Zhi-Yuan Xie, Zhong-Yi Lu, and Ji-Rong Wen. 2021. Enabling lightweight fine-tuning for pre-trained language model compression based on matrix product operators. 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)*, pages 5388–5398.

Weiyang Liu, Zeju Qiu, Yao Feng, Yuliang Xiu, Yuxuan Xue, Longhui Yu, Haiwen Feng, Zhen Liu, Juyeon Heo, Songyou Peng, et al. 2023. Parameter-efficient orthogonal finetuning via butterfly factorization. *arXiv preprint arXiv:2311.06243*.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022b. 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)*, pages 61–68.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2018. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D Lee, Danqi Chen, and Sanjeev Arora. 2023. Fine-tuning language models with just forward passes. *arXiv preprint arXiv:2305.17333*.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. Peft: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft.

Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Scott Yih, and Madian Khabsa. 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)*, pages 6253–6264.

Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. 2015. Tensorizing neural networks. *Advances in neural information processing systems*, 28.

Ivan V Oseledets. 2011. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Jie Ran, Rui Lin, Jason Chun Lok Li, Jiajun Zhou, and Ngai Wong. 2023. Pecan: A product-quantized content addressable memory network. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE.

Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Jiayi Tian, Chao Fang, Haonan Wang, and Zhongfeng Wang. 2023. Bebert: Efficient and robust binary ensemble bert. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. 2022. Dylora: Parameter efficient tuning of pre-trained models using dynamic

search-free low-rank adaptation. *arXiv preprint arXiv:2210.07558*.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2018. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2017. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Jiajun Wu, Jiajun Zhou, Yizhao Gao, Yuhao Ding, Ngai Wong, and Hayden Kwok-Hay So. 2023. Msd: Mixing signed digit representations for hardware-efficient dnn acceleration on fpga with heterogeneous resources. In *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 94–104. IEEE.

Lifan Yuan, Yangyi Chen, Ganqu Cui, Hongcheng Gao, Fangyuan Zou, Xingyi Cheng, Heng Ji, Zhiyuan Liu, and Maosong Sun. 2024. Revisiting out-of-distribution robustness in nlp: Benchmarks, analysis, and llms evaluations. *Advances in Neural Information Processing Systems*, 36.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 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)*, pages 1–9.

Yu Zhang and Qiang Yang. 2021. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 34(12):5586–5609.

Jiajun Zhou, Jiajun Wu, Yizhao Gao, Yuhao Ding, Chaofan Tao, Boyu Li, Fengbin Tu, Kwang-Ting Cheng, Hayden Kwok-Hay So, and Ngai Wong. 2023. Dybit: Dynamic bit-precision numbers for efficient quantized neural network inference. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1.

A Experiment setup

A.1 Dataset Setup

We initially conducted experiments on the Generalized Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018), encompassing various natural language understanding tasks. These tasks include perceptual analysis (SST-2 (Socher et al., 2013)), language acceptability (CoLA (Warstadt et al., 2018)), similarity and paraphrase tasks (MRPC, STS-B, QQP (Dagan et al., 2005)), and natural language reasoning (MNLI, QNLI, RTE (Williams et al., 2017; Rajpurkar et al., 2018)). The metrics we used for the GLUE benchmark are summarized in Table 8.

Table 8: Metrics that we use to evaluate GLUE Benchmark for BERT-based Model.

Task Name	Metric
QNLI	Accuracy
SST-2	Accuracy
MNLI	Matched Acc.
CoLA	Matthews corr.
MRPC	F1
STS-B	Spearman corr.
RTE	Accuracy
QQP	F1

Subsequently, we selected both SuperGLUE tasks (Wang et al., 2019), involving classification (CB, BoolQ, WSC) and multiple-choice (COPA and ReCoRD), as well as two additional generation tasks about question answering (SQuAD (Rajpurkar et al., 2016), DROP (Dua et al., 2019)). For the test with the SuperGLUE and generation datasets, we increase the difficulty by employing a low data resource setting. We randomly sample 1,000 examples for training, 500 examples for validation, and 1,000 examples for testing. We follow the prompt settings in Appendix D of (Malladi et al., 2023) to transfer the classification into the language model tasks and the metrics we used are summarized in Table 9. All experiments are finished with the AdamW optimizer (Loshchilov and Hutter, 2018).

A.2 Baselines

Fine-tuning (FT) is a common approach for adaptation. In this process, the model is initialized with pre-trained weights and biases, and all model parameters undergo gradient updates.

Table 9: Metrics that we use to evaluate SuperGLUE and generations tasks.

Task Name	Metric
CB	F1
BoolQ	Accuracy
WSC	F1
COPA	Accuracy
ReCoRD	F1
SQuAD	F1
DROP	F1

Adapters, as proposed by (Houlsby et al., 2019), insert adapter layers between the self-attention module (and the MLP module) and the subsequent residual connection. An adapter layer consists of two fully connected layers with biases, separated by a nonlinearity. We conducted the adapter experiment using various adapter bottleneck sizes, such as 8 and 64.

LoRA introduces trainable pairs of rank decomposition matrices in parallel to existing weight matrices. As mentioned in Sections 3 and 4 (Hu et al., 2021), we primarily apply LoRA to the query and value layers in most experiments for simplicity. The number of trainable parameters is determined by the LoRA rank and the shape of the original weights, as shown in Table 12.

Prefix Tuning adds a prefix of m tunable representations at each layer and freezes the remaining parts of the model. These representations serve as new keys and values, providing additional context during the attention operation. The tunable representations are initialized by randomly sampling tokens from the vocabulary and passing them through the language model to obtain their keys and values at various attention layers. In our experiments, we observe that $m = 8$ can achieve satisfactory performance across most tasks.

BitFit is a baseline where only the bias vectors are trained while keeping all other parameters frozen. We only test the BitFit methods with the BERT-based models since the bias term is not enabled in the linear layer of the LLaMA models.

Prompt Tuning tuning technique can guide the behavior of language models by adding text prompts to the input, wherein we only need to train a small part of prompt parameters.

IA3 rescales inner activations with learned vectors on the attention and feed-forward layers. The method is ultra-parameter efficient, with a similar parameter size as the proposed LoRETTA methods. However, the LoRETTA methods shows better performance in almost all tasks on the Llama-2 models compared with the IA3 method.

A.3 Hyperparameters

We outline the configuration details for each comparative experiment. Specifically, for the DeBERTa/RoBERTa-Base models, the learning rates and batch sizes of individual methods are presented in Table 12. For a fair comparison, we use almost the same learning rate, batch size, and learning rate setting for different methods in the same tasks, except for the full model fine-tuning, which cannot converge under the large learning rate. In the case of P-tuning, we extended the prompt length to 768, with a virtual token count of 20 during fine-tuning. Regarding the prompt method, we increased the virtual token to 20. For prefix, we used Prefix-Propagation (Li et al., 2023a) to experiment. We implement the LoRA, Adapters, prefix/prompt tuning, and P-tuning methods with the PEFT library (Mangrulkar et al., 2022). All GLUE tasks underwent training for 10 to 20 epochs.

Except for the experiments on BERT-based models, we also compare our proposed method with the Adapters, LoRA, and prefix tuning methods. We use the hyperparameters in Table 13 for the experiment on LLaMA-2 models. Note that even though we run all experiments for 3 epochs, further learning steps may help to improve the performance of our proposed methods further.

A.4 Additional Detail of TT-format

The design of the tensor shape $[k_1, \cdots, k_d]$ for models with other hidden sizes are summarized in Table 10. Here we only show the tensor shape used in the DeBERTa/RoBERTa-base and LLaMA-2-7b models. The hidden sizes used are 768 and 4096 respectively. For other models with different hidden sizes, the tensor shape needs to be defined specifically before the training. More detail can be found in the code we provided, which has included the most widely used hidden sizes (like 768, 1024, 1536, 4096, 5120, and 8192) in the implementations, which work for nearly all kinds of widely used models. For a more detailed setup, please refer to the source code of the loretta library provided.

Table 10: The shape settings of the TT-format

Modules	Matrix Shape	Tensor Shape
Tensorized Adapters	768×64	[8, 8, 12, 8, 8]
	4096×64	[16, 16, 16, 4, 4, 4]
	64×768	[8, 8, 12, 8, 8]
	64×4096	[4, 4, 4, 16, 16, 16]
Tenosrized updating matrix	768×8	[8, 8, 12, 8]
	768×16	[8, 8, 12, 4, 4]
	768×32	[8, 8, 12, 8, 4]
	8×768	[8, 12, 8, 8]
	16×768	[4, 4, 12, 8, 8]
	32×768	[4, 8, 12, 8, 8]
	4096×8	[8, 8, 8, 8, 8]
	4096×16	[8, 8, 8, 8, 4, 4]
	4096×32	[8, 8, 8, 8, 8, 4]
	8×4096	[8, 8, 8, 8, 8]
	16×4096	[4, 4, 8, 8, 8, 8]
	32×4096	[4, 8, 8, 8, 8, 8]
Tenosrized Classifier(Optional)	768×768	[12, 8, 8, 8, 8, 12]
	768×768	[8, 8, 8, 8, 8, 8, 8, 8]

B Ablation Study on Classifier and Layernorm

Here, we examined six scenarios for three tasks for both LoRETTA$_{adp}$ and LoRETTA$_{rep}$ methods, considering the trainable status of layernorm and classifiers. The results are shown in Table 11. Our findings highlight that the tensorized classifier demonstrates comparable results to the regular classifier with a notable reduction in parameters. Furthermore, the layernorm plays a significant role in our framework.

First, we set the tensorized classifier/adapters to be trainable and observed the influence of layernorm. We find that layernorm plays an important role in our framework. Then, we fix the layernorm to

Table 11: LoRETTA fine-tuning with/without layernorm and classifier layers.

Method	Train Param	SST-2	MRPC	QNLI	Classfier & Pooler	Layernorm
LoRETTA$_{adp}$	0.061M	94.38	92.01	92.98	Tensorized	No
LoRETTA$_{adp}$	0.1M	95.3	92.53	92.99	Tensorized	Yes
LoRETTA$_{adp}$	0.650M	93	91.9	93.15	Regular	No
LoRETTA$_{adp}$	0.688M	94.26	91.09	93.06	Regular	Yes
LoRETTA$_{adp}$	0.058M	93.92	92.11	92.71	No	No
LoRETTA$_{adp}$	0.096M	94.03	91.31	93.46	No	Yes
LoRETTA$_{rep}$	0.054M	95.53	88.73	93.25	Tensorized	Yes
LoRETTA$_{rep}$	0.016M	93.81	90.78	90.15	Tensorized	No
LoRETTA$_{rep}$	0.645M	95.18	91.88	92.99	Regular	Yes
LoRETTA$_{rep}$	0.606M	95.41	91.00	92.57	Regular	No
LoRETTA$_{rep}$	0.052M	95.41	91.19	92.69	No	Yes
LoRETTA$_{rep}$	0.014M	94.83	87.5	91.87	No	No

be trainable and observe the tensorized classifier demonstrates comparable results to the regular classifier and reduces about 92% of trainable parameters in the last layer. Furthermore, the tensorized classifier still helps a lot in improving the performance of our approach, even if we freeze the layernorm.

We also test the influence of the tensorized classifier layer for our LoRETTA$_{rep}$ method. As we can see from the table, optimizing the classifier layer for the sequence classification task is important. Our tensorized classifier successfully reduces the trainable parameters led by the traditional classifier layer and still maintains high performance.

Table 12: The hyperparameter grids used for GLUE experiments. We fine-tune each task for 10 to 20 epochs, evaluating the validation loss every 500 steps. We record the best model checkpoint based on the validation loss.

Experiment	Hyperparameters	Values
FT	Batch size	$[16, 32]$
	Learning rate	$1e-6$
LoRA	Batch size	$[16, 32]$
	Learning rate	$[1e-4, 5e-4]$
	Rank	$4, 8$
Adapters	Batch size	$[16, 32]$
	Learning rate	$[1e-4, 5e-4]$
	Bottleneck dimension	$[8, 64]$
Prefix	Batch size	$8, 64$
	Learning rate	$[1e-4, 5e-4]$
	Prefix Tokens	8
Bitfit	Batch size	$[16, 32]$
	Learning rate	$[1e-4, 5e-4]$
	Bias Terms	All
Prompt	Batch size	$[16, 32]$
	Learning rate	$[1e-4, 5e-4]$
	Tokens	10
P-tuning	Batch size	$[16, 32]$
	Learning rate	$[1e-4, 5e-4]$
	Tokens	20
	Prompt Length	$[128, 768]$
LoRETTA$_{adp}$	Batch size	$[16, 32]$
	Learning rate	$[1e-4, 5e-4]$
	Bottleneck dimension	64
	Tensor Rank	$[2, 5, 10, 20]$
LoRETTA$_{rep}$	Batch size	$[16, 32]$
	Learning rate	$[1e-4, 5e-4]$
	Tensor Rank	$[2, 5, 10, 20]$

Table 13: The hyperparameter grids used for LLaMA-2 experiments. We evaluate the validation loss every 1000 steps and record the best model checkpoint according to the validation loss.

Experiment	Hyperparameters	Values
FT	Batch size	$[1, 2]$
	Learning rate	$5e-6$
LoRA	Batch size	$[1, 2]$
	Learning rate	$1e-4$
	Rank	8
Adapters	Batch size	$[1, 2]$
	Learning rate	$1e-4$
	Bottleneck r	$[8, 64]$
Prefix	Batch size	$[1, 2]$
	Learning rate	$1e-4$
	Prefix Tokens	8
LoRETTA$_{adp}$	Batch size	$[1, 2]$
	Learning rate	$1e-4$
	Bottleneck dimension	64
	Tensor Rank	$[2, 4, 8, 16, 32]$
LoRETTA$_{rep}$	Batch size	$[1, 2]$
	Learning rate	$1e-4$
	Tensor Rank	$[2, 4, 8, 16, 32]$