Prompt Tuning Strikes Back: Customizing Foundation Models with Low-Rank Prompt Adaptation

Abhinav Jain
Department of Computer Science
Rice University
aj70@rice.edu

Swarat Chaudhuri
Department of Computer Science
UT Austin
swarat@cs.utexas.edu

Thomas Reps
Department of Computer Science
University of Wisconsin-Madison
reps@cs.wisc.edu

Chris Jermaine
Department of Computer Science
Rice University
cmj4@rice.edu

Abstract

Parameter-Efficient Fine-Tuning (PEFT) has become the standard for customising Foundation Models (FMs) to user-specific downstream tasks. However, typical PEFT methods require storing multiple task-specific adapters, creating scalability issues as these adapters must be housed and run at the FM server. Traditional prompt tuning offers a potential solution by customising them through task-specific input prefixes, but it under-performs compared to other PEFT methods like LoRA. To address this gap, we propose **Lo**w-Rank **P**rompt **A**daptation (LoPA), a prompt-tuning-based approach that performs on par with state-of-the-art PEFT methods and full fine-tuning while being more parameter-efficient and not requiring a server-based adapter. LoPA generates soft prompts by balancing between sharing task-specific information across instances and customization for each instance. It uses a low-rank decomposition of the soft-prompt component encoded for each instance to achieve parameter efficiency. We provide a comprehensive evaluation on multiple natural language understanding and code generation and understanding tasks across a wide range of foundation models with varying sizes.

1 Introduction

Language models exhibit remarkable few-shot learning capabilities, demonstrating strong performance across tasks, including those unseen during training [2, 32, 23]. Nevertheless, fine-tuning remains crucial for optimised performance on a given downstream task. However, it becomes increasingly challenging with larger models because updating all parameters is impractical. Parameter Efficient Fine-Tuning (PEFT) presents a promising solution, adjusting a limited subset of parameters while leaving the rest unchanged. This approach allows the personalisation of pre-trained Foundation Models (FMs) to multiple users simultaneously.

In recent years, numerous PEFT approaches have been proposed [11, 12, 16, 17, 18, 30, 36], each varying in how and what they modify within FMs [10, 9]. These variations can be categorized by the position in the FMs where modifications are applied, the functions used for modification, and the methods of integrating the modifications. While effective, these methods require maintaining multiple adapter-like modules for each user-specific task on the FM server and the need to select and assemble a subset of these modules every time a batch of user requests is processed for inference [35] (see Fig. 1).

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

Prompt tuning [15] is a simple approach that has certain advantages. First, it is parameter-efficient, requiring only a small set of vectors (soft prompts) that are prepended at the input layer and learned for a specific task. Second, prompt-based personalization requires no task-specific processing on the server. A task-specific prefix is added before processing, allowing the FM server to perform the same processing regardless of the task. However, despite its advantages, prompt tuning has been shown to underperform compared to other methods for PEFT [10]. This gap in performance raises concerns about the viability of prompt tuning as a solution. Recently, efforts have been made to enhance the performance of prompt tuning by strategically inserting soft prompts into different layers of the transformer [20, 38, 19, 41]. However, this improvement increases the number of parameters and again necessitates server-side modifications for serving multiple tasks. An interesting recent work explored a simple method to make soft prompts input-dependent by using a lightweight prompt generator for each sample [38]. This approach achieved notable improvements, raising the question: *Can we further improve the performance of prompt tuning while staying parameter-efficient?*

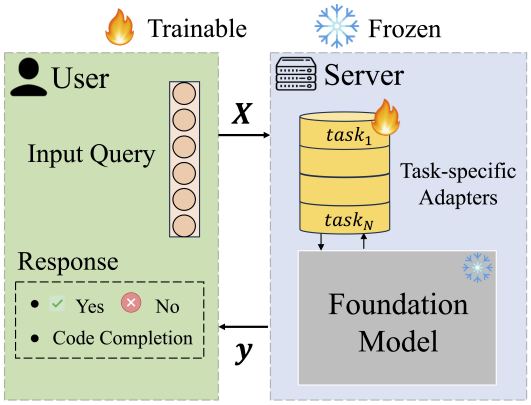


Figure 1: A schematic illustrating how typical PEFT methods like LoRA achieve personalization of a foundation model for multiple tasks, such as Yes/No text classification or code completion, during inference.

To this end, we propose Low-rank Prompt Adaptation (LoPA), a new instance-aware prompt tuning-based approach [1]. LoPA constructs the soft prompt from two components: a task-specific element that shares task information across samples and an instance-specific element that incorporates information from each individual instance. Our extensive analysis reveals that relying solely on either component, as done in previous works [15, 38], is insufficient for outperforming other PEFT baselines. LoPA achieves its high performance by taking a more balanced approach.

LoPA combines the task-specific and instance-specific components using a gating function, which activates task-specific information conditioned on each instance. Additionally, it employs a low-rank decomposition of the instance-level component to enhance parameter efficiency (see Fig. 2). Once trained, users can provide the learned soft prompts as a prefix with their input to the FM, incurring no additional computational cost at the server.

We conducted extensive experiments on various models. These included six benchmark NLU tasks from the GLUE dataset and three Code Understanding and Generation tasks. Our results show that LoPA outperforms existing prompt-tuning methods. It often matches the performance of full fine-tuning and LoRA. In 11 out of 24 test cases, we found LoPA outperformed LoRA.

To summarize, the contributions of this work are as follows:

- We propose LoPA, a parameter-efficient and high-performing prompt-tuning strategy.
- We verify its effectiveness by evaluating it against full fine-tuning and state-of-the-art PEFT methods in nine tasks using seven different transformer backbones.

2 Related Work

Adapter-based. Several recent approaches have emerged to support parameter-efficient fine-tuning. These methods typically involve incorporating trainable adapter layers or modules into the transformer network [11, 10, 17]. Training such layers has been demonstrated to be computationally more economical than full fine-tuning while maintaining performance. Notably, LoRA [12] has gained prominence for its low-rank approximation of model updates, effectively capturing task-specific knowledge. Beyond LoRA, Compacter [14] introduces adapters parameterised by low-rank hyper-complex multiplication (PHM) layers [39] to achieve a more optimal balance between task

[1]The code for LoPA can be found here

performance and the number of trainable parameters. DoRA [18] is another recent approach that decomposes weights into their magnitude and directional components and employs LoRA for directional updates to minimise the number of trainable parameters efficiently.

Soft Prompting. Another line of work advocates for strategically inserting soft prompts into hidden states of the transformer instead of using trainable adapter modules. For example, prefix-tuning [16] and P-tuning-v2 [20] prepend trainable prefix vectors to keys and values matrices at all transformer layers. Prompt tuning, P-tuning, and DePT [15, 21, 30] are special cases that operate by prepending prompt embeddings to the input at the first layer, with [30] being a hybrid-approach that further uses LoRA to learn updates for the input's embedding matrix. While these approaches are instance-agnostic and optimise a task-specific prompt, there are also methods to optimise an instance-aware soft prompt. For instance, IDPG [38] generates a soft prompt for each sample, prepended either at the initial word-embedding layer (version-S) or all layers (version-M). LPT [19] inserts a prompt into some intermediate layer (i) to eliminate gradient calculation below it for faster training and (ii) to retain more task-related information (which could be lost if it had to be propagated through lower layers). SPT [41] aims to be more intelligent by learning a gating function to determine whether the soft prompt from the previous layer should be propagated or if a new one should be learned.

With the exception of prompt tuning and S-IDPG, PEFT approaches mostly operate by injecting prefixes and new trainable modules into deeper layers or doing low-rank re-parameterization of existing ones, necessitating the storage of PEFT parameters at the server to update the foundation model. In contrast, LoPA provides the soft prompt as a prefix that is prepended to the input query, overcoming the need to store task-specific parameters on the server. Furthermore, we demonstrate that LoPA achieves a more balanced trade-off between specificity and generalization compared to existing approaches for enhancing learned models, such as prompt tuning [15], which solely focuses on a general task-specific soft prompt, and IDPG [38], which emphasizes an instance-specific prompt.

3 Proposed Methodology

In this section, we formally define the proposed approach, followed by an explanation of how it affects model learning as compared to existing soft-prompting approaches.

3.1 Preliminaries

Transformers. A transformer model consists of multiple stacked layers, where each layer has multiple heads ($= N_H$), each performing self-attention over the input [33]. Let us consider a single head, H parameterised by the query, key, and value weights as $W^Q, W^K, W^V \in \mathcal{R}^{d_H \times d}$, respectively, where d is the model dimension. In multi-headed attention, d_H is typically set to $\frac{d}{N_H}$. For a given sequence of n input vectors $\mathbf{X} = \{\mathbf{x}^1, \ldots, \mathbf{x}^n\}$ and a query vector $\mathbf{x}^i$ with each $\mathbf{x} \in \mathcal{R}^d$, the output of the head at position i is -

$$\mathbf{o}^i = \text{Attention}(W^Q\mathbf{x}^i, W^K\mathbf{X}, W^V\mathbf{X}) = \text{softmax}\big(W^Q\mathbf{x}^i(W^K\mathbf{X})^\top\big)W^V\mathbf{X} \tag{1}$$

where, we have ignored the constant $\sqrt{d_H}$ for notational convenience. For the first layer of the transformer network, the input $\mathbf{X}$ is the embedding matrix i.e. $\mathbf{X} = \mathbf{X}_e \in \mathcal{R}^{d \times n}$.

Full Fine-Tuning (FFT). With fine-tuning, the objective is to adapt the model to a new task with data $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$. Formally, adaptation is achieved by updating the weights (W) of the model to maximise the log-likelihood of the response $\mathbf{y}$, i.e., $\max_W \mathcal{L} = \mathbb{E}_{(\mathbf{X},\mathbf{y})\sim\mathcal{D}}[\log p_W(\mathbf{y}|\mathbf{X})]$.

Prompt-Tuning (PT). The objective of prompt-tuning is to achieve task-specific model adaptation by learning a "soft prompt" $\mathbf{Z}$. Such a soft prompt is prepended to the input word embeddings and trained via back-propagation: $\max_\mathbf{Z} \mathcal{L} = \mathbb{E}[\log p_W(\mathbf{y}|\text{concat}(\mathbf{Z}, \mathbf{X}_e))]$. In PT, the soft prompt $\mathbf{Z} \in \mathcal{R}^{d \times m}$ is parameterised by a set of m learnable vectors $\{\mathbf{z}^1, \ldots \mathbf{z}^m\}$, where m denotes its length [15]. Thus, it can be interpreted as an embedding of virtual tokens that enable the model to perform a downstream task without updating its parameters [28, 8].

With prompt tuning, the output of an attention head in the first layer is modified as follows:

$$\mathbf{o}^i_{PT} = \text{Attention}(W^Q\mathbf{x}^i, W^K\text{concat}(\mathbf{Z}, \mathbf{X}_e), W^V\text{concat}(\mathbf{Z}, \mathbf{X}_e))$$

By using the formulation of Attention from Eq. 1, this equation can be equivalently written as

$$\mathbf{o}^i_{PT} = \underbrace{\sum_k A_{ik} W^V \mathbf{z}^k}_{bias} + \underbrace{(1 - \sum_k A_{ik})}_{scale} \mathbf{o}^i \tag{2}$$

$$\text{with} \quad A_{ik} = \frac{\exp((W^K \mathbf{z}^k)^\top W^Q \mathbf{x}^i)}{\sum_k \exp((W^K \mathbf{z}^k)^\top W^Q \mathbf{x}^i)) + \sum_j \exp((W^K \mathbf{x}^j)^\top W^Q \mathbf{x}^i))} \tag{3}$$

where, A_{ik} is the attention transformer gives to the prefix vector $\mathbf{z}_k$ for a given query vector $\mathbf{x}^i$. In Eq. 2, we can observe that the soft prompt linearly interpolates the head's position-wise output; where the bias term can be considered in an offset subspace spanned by vectors $\{W^V \mathbf{z}^k\}_{k=1}^m$ with dimension m (or $\leq m$) [10].

3.2 Low-rank Prompt Adaptation (LoPA)

LoPA constructs the soft prompt as $\mathbf{Z} = \mathbf{Z}_S \circ g(\mathbf{Z}_I)$, where $\mathbf{Z}_S \in \mathbb{R}^{d \times m}$ is the task-specific component and $\mathbf{Z}_I \in \mathbb{R}^{d \times m}$ is the instance-specific component. These are combined using the gating function g, implemented using sigmoid, where $\circ$ denotes the Hadamard product. Intuitively, by sharing $\mathbf{Z}_S$ across instances, it captures general information, adapting the model to user-defined tasks, while $\mathbf{Z}_I$ fine-tunes the soft prompt for specific instances, acting as activations. Both $\mathbf{Z}_S$ and $\mathbf{Z}_I$ have their own dedicated parameters. Similar to prompt tuning, $\mathbf{Z}_S$ consists of m learnable vectors. Conversely, $\mathbf{Z}_I$ is obtained from input using the encoding function $f : \mathbb{R}^{d \times n} \rightarrow \mathbb{R}^{d \times m}$. However, encoding a matrix of size $d \times m$ can be expensive. For example, in [38], an MLP layer with hidden dimension h requires $\mathcal{O}(hdm)$ parameters. To improve parameter efficiency, we assume a low-rank decomposition of $\mathbf{Z}_I$ as $\mathbf{Z}_I = \mathbf{u} \times \mathbf{v}^\top$ and encode the two matrices with rank r, $\mathbf{u} \in \mathbb{R}^{d \times r}$ and $\mathbf{v} \in \mathbb{R}^{m \times r}$, using separate MLP layers. This design makes f computationally cheaper, with $\mathcal{O}(hdm(\frac{r}{d} + \frac{r}{m}))$ parameters, reduced by a factor of $(\frac{r}{d} + \frac{r}{m}) < 1$ when r is chosen to be $\ll \min(d, m)$. The overall composition of $\mathbf{Z}$ can be represented as:

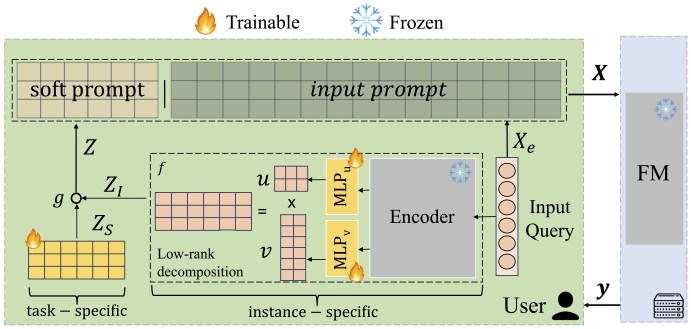


Figure 2: An illustration of LoPA. No task-specific adapters need to be stored on the server. | represents the concatenation of the soft prompt $\mathbf{Z}$ and the input prompt $\mathbf{X}_e$ i.e. $\mathbf{X} = \text{concat}(\mathbf{Z}|\mathbf{X}_e)$

$$\mathbf{Z} = \underline{\mathbf{Z}_S} \circ g(\underbrace{\underline{\texttt{MLP}_\mathbf{u}}(\mathbf{X}') \times \underline{\texttt{MLP}_\mathbf{v}}(\mathbf{X}')^\top}_{\mathbf{Z}_I = f(\mathbf{X}')}) \tag{4}$$

where _ marks the three trainable components. The MLP head consists of a down- and up-projection layer with a non-linear activation in between. It sits atop a smaller language model, referred to as the Encoder model, that gives the input representation $\mathbf{X}'$. A visual illustration of the framework can be found in Fig. 2.

Next, we provide an intuitive explanation why LoPA could be better than traditional prompt-tuning [15] and existing instance-specific approaches [38].

Offset subspace induced by LoPA. From Eq.2, we can observe that the bias vector in the offset subspace is a linear combination of $\{W^V \mathbf{z}^k\} \quad \forall k \in \{1, \ldots, m\}$, with A_{ik} (Eq. 3) representing the scalars [10, 26]. In LoPA, the input influences $\mathbf{Z}$, causing the vectors $\{W^V \mathbf{z}^k\}$ to vary accordingly. Consequently, distinct offset sub-spaces emerge for different inputs. In contrast, traditional prompt tuning maintains fixed vectors while allowing only the scalars to vary with input. As a result, instance-sensitive approaches can exert greater influence on the attention patterns in deeper layers of the transformer, thereby offering enhanced flexibility and responsiveness to varying inputs.

Tuning	Tunable Parameters	SST-2 (acc)	MNLI (acc)	MRPC (acc, F1)	QNLI (acc)	QQP (acc, F1)	RTE (acc)	Avg
FFT	355M	95.99	**90.40**	90.81	94.60	**90.39**	**85.92**	**91.35**
p-tuning-v2	0.49M	89.91	88.13	70.18	85.21	84.63	53.43	78.58
prefix-tuning	25.75M	93.80	89.51	90.86	**94.98**	86.87	82.67	89.78
LoRA	2.36M	**96.22**	90.30	90.77	94.69	89.91	85.66	91.26
None	0	59.97	39.60	73.52	50.16	42.34	53.43	53.17
PT	10.2K	84.40	54.67	72.38	58.74	48.20	53.07	61.91
DePT	10.2K	89.68	71.51	72.97	57.09	45.81	53.79	65.14
S-IDPG	2.89M	95.30	84.50	78.60	90.48	84.88	77.26	85.17
Ours	1.60M	95.99	89.22	**91.09**	93.74	89.72	83.39	90.53

Table 1: Performance on GLUE tasks. We report the average of accuracy and F1 for both MRPC and QQP. For all the other tasks, we report accuracy. Approaches below the dotted line do not require any modification to the model on the server side. **Bold** denotes the best-performing tuning method for the given model. Underline marks the best result among all prompt tuning methods.

Coupled learning of $\mathbf{Z}_S$ and $\mathbf{Z}_I$. Let's examine the partial derivatives of the objective $\mathcal{L}$ with respect to $\mathbf{Z}_S$ and $\mathbf{Z}_I$. Using the chain-rule on the formulation in Eq. 4 with $g(.) = \text{sigmoid}(.)$,

$$\nabla_{\mathbf{Z}_S}\mathcal{L} = \nabla_{\mathbf{Z}}\mathcal{L} \circ \sigma(\mathbf{Z}_I)$$
$$\nabla_{\mathbf{Z}_I}\mathcal{L} = (\nabla_{\mathbf{Z}}\mathcal{L} \circ \mathbf{Z}_S) \cdot \sigma(\mathbf{Z}_I)(1 - \sigma(\mathbf{Z}_I))$$

These expressions suggest a coupled learning process where changes in $\mathbf{Z}_I$ (through the sigmoid function) directly impact the updates to $\mathbf{Z}_S$, and the updates to $\mathbf{Z}_I$ are scaled by both $\mathbf{Z}_S$ and the derivative of the sigmoid function. In contrast, consider IDPG with $\mathbf{Z} = \texttt{MLP}(\mathbf{X}')$ [38]. Here, the bias term of the MLP layer can be viewed as a task-specific element, such that the composition is $\mathbf{Z} = \mathbf{Z}_S + \mathbf{Z}_I$. This approach results in updates to $\mathbf{Z}_S$ and $\mathbf{Z}_I$ being independent of each other. Such a linear-sum operation may fail to capture the complex relationships between the two components that LoPA can capture due to LoPA's non-linear factorization.

4 Experiments, Results, and Discussion

4.1 Experimental Setup

Tasks. We evaluate LoPA on (i) six Natural Language Understanding (NLU) tasks from the GLUE benchmark [34]—namely, SST-2 [31], MNLI [37], MRPC [3], QNLI [29], QQP, and RTE [5]; (ii) a code-generation task that requires the model to complete method bodies from MBPP benchmark [1], and (iii) two code-understanding tasks—namely, CruxEval-I (input prediction) and CruxEval-O (output prediction) from CruxEval benchmark [6]. These tasks assess the model's ability to reason about the execution of simple Python programs by asking it to predict the input given the output of a function and vice-versa.

Baselines. We evaluate LoPA against the following baselines that represent different customisation approaches. (1) FFT, (2) LoRA [12], (3) p-tuning-v2 [20] and (4) prefix-tuning (h=512) [16] are selected as representatives of methods that customise models on the server side, with FFT representing full fine-tuning and the remainder showcasing parameter-efficient techniques. (5) Standard prompt-tuning [15], (6) S-IDPG [38] and (7) DePT [30] are chosen because they customise models from the user side, focusing on soft prompt insertion with DePT also learning updates to the input embedding matrix. For IDPG, we use DNN layers in MLP-head to encode the soft prompt for the input embedding layer. We chose DNN over PHM layers [39] because we found them to exhibit better performance across our task set. Refer to Appendix 7.1 for the comparison. Additionally, the proposed LoPA also uses DNN layers, facilitating a fair comparison with S-IDPG-DNN [38]. Lastly, we include results for the model without any fine-tuning to report its (8) zero-shot performance.

Model	Tuning	#Params	Code Understanding		Code Generation
			CruxEval-I	CruxEval-O	MBPP
CodeGen-350M	FFT	350M	33.0	**19.5**	17.49
	LoRA	1.3M	31.0	18.2	**21.56**
	None	0	20.8	15.0	15.85
	PT	10.2K	32.8	15.8	15.20
	S-IDPG	8.5M	16.9	13.2	17.04
	Ours	4.4M	**34.5**	18.5	17.04
DeepseekCoder-1.3B	FFT	1.3B	**45.0**	34.8	**44.76**
	LoRA	4.7M	35.5	**36.0**	44.14
	None	0	26.8	29.8	34.08
	PT	20.5K	41.2	31.2	34.49
	S-IDPG	16.3M	26.0	28.5	42.50
	Ours	4.2M	43.0	34.5	44.66
Phi2-2.7B	FFT	2.7B	40.2	37.0	**55.03**
	LoRA	7.9M	41.5	**42.5**	51.54
	None	0	33.5	33.0	45.17
	PT	25.6K	35.0	34.0	49.69
	S-IDPG	20.3M	35.0	33.0	53.29
	Ours	4.76M	**43.0**	37.2	52.15
Phi3-3.8B	FFT	3.8B	39.2	39.5	**54.00**
	LoRA	6.3M	38.0	41.2	42.92
	None	0	33.8	39.5	8.82
	PT	30.7K	33.5	31.5	34.08
	S-IDPG	24.2M	31.0	39.5	42.29
	Ours	5.3M	**42.2**	**42.5**	44.35
DeepseekCoder-7B	FFT	7B	*OOM*	*OOM*	*OOM*
	LoRA	11.8M	47.5	**49.8**	53.38
	None	0	39.3	44.0	50.51
	PT	41.0K	45.8	44.8	37.47
	S-IDPG	32.1M	40.5	41.5	**53.59**
	Ours	6.35M	**50.0**	48.0	52.46
Llama3-8B	FFT	8B	*OOM*	*OOM*	*OOM*
	LoRA	9.4M	**45.5**	**40.5**	44.55
	None	0	27.0	31.5	**45.37**
	PT	41.0K	37.5	32.0	26.07
	S-IDPG	32.1M	29.2	35.2	33.05
	Ours	6.4M	41.2	39.8	43.94

Table 2: Performance comparison on CruxEval and MBPP tasks. We report average *pass*@1 scores. Approaches below the dotted line are prompt-tuning methods, which do not require any modification to the model on the server side. **Bold** denotes the best-performing tuning method for the given model. Underline marks the best result among all prompt-tuning methods. *OOM* indicates that the corresponding tuning approach exceeded the available GPU memory and ran out of memory.

$\mathbf{Z} =$	SST-2 (acc)	MNLI (acc)	MRPC (acc, F1)	QNLI (acc)	QQP (acc, F1)	RTE (acc)	Avg
$\text{concat}(\mathbf{Z}_S, \mathbf{Z}_I)$	94.50	78.45	76.00	91.43	76.11	84.12	83.44
$\max(\mathbf{Z}_S, \mathbf{Z}_I)$	**95.99**	**89.37**	88.62	**93.74**	78.44	**85.92**	88.68
$\mathbf{Z}_S \circ g(\mathbf{Z}_I)$	**95.99**	89.22	**91.09**	**93.74**	**89.72**	83.39	**90.53**

Table 3: Performance of LoPA on GLUE tasks with respect to function encoding $\mathbf{Z}$. $\text{concat}(.)$ represents the concatenation of $\mathbf{Z}_S$ and $\mathbf{Z}_I$. $\max(.)$ represents the element-wise max operation. We report the average of accuracy and F1 for both MRPC and QQP. For all the other tasks, we report accuracy. **Bold** denotes the best-performing encoding function for LoPA.

Backbone Architectures. For NLU tasks, we test all the tuning methods on 355M `RoBERTa` [22] similar to the setup opted by prior work [20, 38, 41]. For Code Generation and Understanding, we test a subset of the baselines (FFT, LoRA, PT, S-IDPG) on a range of FM backbones; starting from smaller FMs: 350M `CodeGen-mono` [25] and 1.3B `Deepseek-Coder` [7]; mid-sized FMs: 2.7B `phi-2` [13], 3.8B `phi-3` [24]; and larger FMs: 7B `Deepseek-Coder` [7] and 8B `Llama 3` [23].

Implementation Details. For the GLUE tasks, we use the train-test splits pre-defined in the benchmark, while for the MBPP and CruxEval tasks, we employ a 50-50 split. Across all tasks and backbone models, the soft prompting baselines are implemented with $m = 10$ virtual tokens representing the soft prompt. For NLU tasks, both IDPG and LoPA use `RoBERTa` + `MLP(h=256)` as the encoder, whereas in code tasks, `125M CodeSage` + `MLP(h=1024)` is used. Additionally, for LoPA, the best-performing rank (r) of the low-rank decomposition is chosen from $\{1, 2, 4\}$, and the corresponding number of tunable parameters is reported.

Training Configuration. For NLU tasks, training with FFT and LoRA was done for 10 epochs, while with prompt-tuning-based approaches it was done for 20 epochs. In MBPP, all foundation model (FM) backbones were trained for 10 epochs across all tuning methods. In CruxEval Tasks across all PEFT methods, FM backbones under 7B were trained for 20 epochs, while larger FMs ($\geq$7B) were trained for 10 epochs. Lastly, training with FFT on CruxEval tasks was done for 5 epochs. The learning rates for LoPA are set to 1×10^{-5} in NLU and 1×10^{-3} in Coding tasks. The baseline tuning methods use the following learning rates across all the tasks: FFT using 1×10^{-5}, LoRA and the remainder of soft-prompting approaches using 1×10^{-4}. All experiments are conducted on 40GB 2xA100 GPUs.

Evaluation. We report binary or multi-class classification accuracies and F1 scores for NLU tasks as provided in the GLUE benchmark. For coding tasks, we report the *pass*@1 scores computed using the best-performing temperatures: 0.6 for MBPP and 0.2 for CruxEval.

4.2 Baseline Comparison

Performance on Natural Language Understanding. In Table 1, we can observe that the LoPA consistently outperforms the traditional prompt-tuning method and DePT by an average margin of 28.62 points and 25.39 points respectively. This result demonstrates that conditioning the soft prompt on instances enables it to influence the model's output more significantly. Furthermore, LoPA shows an average improvement of 5.36 points over IDPG, highlighting that the proposed factorisation captures complex relationships between task-specific ($\mathbf{Z}_S$) and instance-specific ($\mathbf{Z}_I$) components. This improvement is particularly notable in limited-data settings, with a 12.5-point increase in MRPC and a 6.13-point increase in RTE.

Additionally, LoPA achieves performance close to FFT and LoRA, within 1 point, while using $760k$ fewer parameters than LoRA owing to LoPA's low-rank decomposition of the soft prompt. This performance profile suggests that the LoPA is a parameter-efficient and high-performing alternative for NLU tasks. It outperforms existing prompt-tuning approaches and performs on par with FFT and LoRA, making it a compelling choice for efficient and effective model tuning.

Performance on Code Understanding. In Table 2, LoPA consistently improves the *pass*@1 score of the baseline with no tuning across all FM backbones. It outperforms prompt-tuning on CruxEval tasks, with modest improvements of approximately 2 to 4 points on smaller FMs like CodeGen

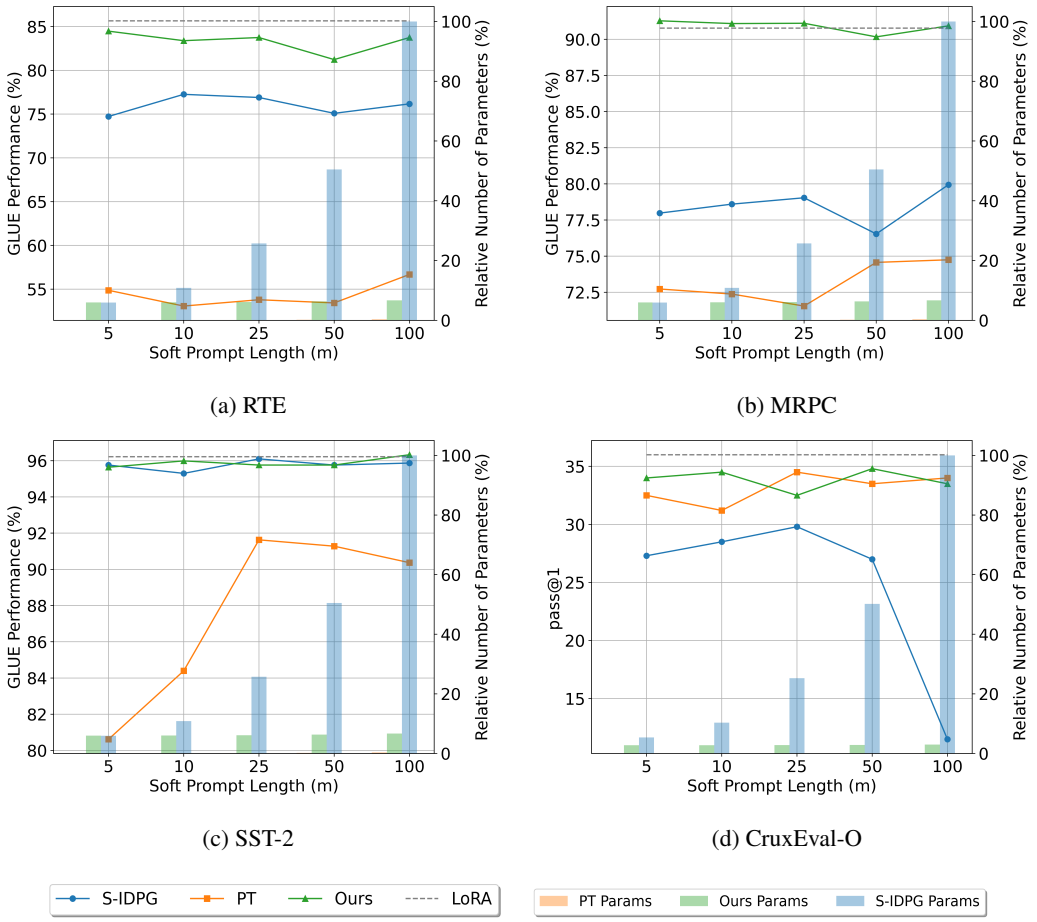
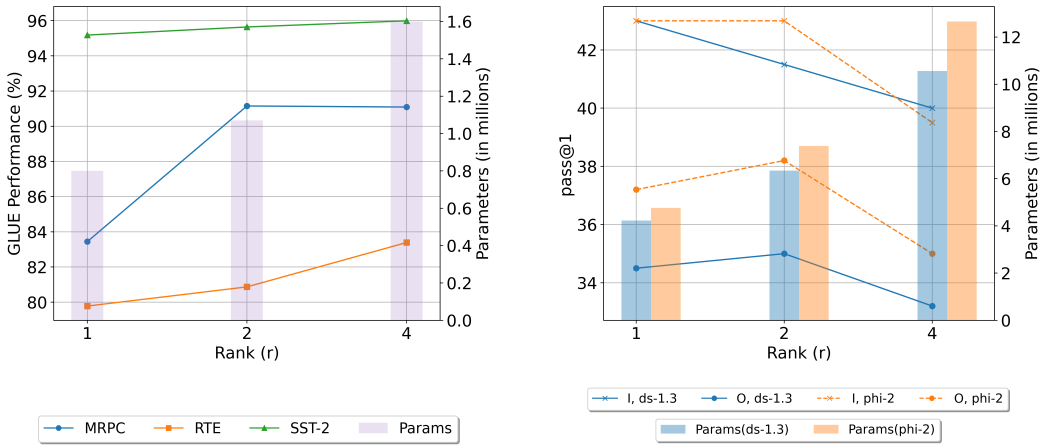


Figure 3: Performance comparison of baselines as a function of m on (a)-(c) GLUE benchmark and (d) CruxEval-O (with DeepseekCoder-1.3B as FM). Tunable parameters shown relative to the method with the most. Higher performance and fewer parameters indicate better results.

and DeepSeekCoder-1.3B and larger improvements ranging from 8 to 11 points on larger FMs like LLama-3 and Phi-3 in CruxEval-O. Furthermore, IDPG performs worse than PT for all models except Phi-3 in CruxEval-O. These results suggest that merely encoding an instance-sensitive soft prompt does not guarantee improvements and can even degrade performance (e.g., IDPG on CodeGen and DeepSeekCoder-1.3B in CruxEval tasks). The poor generalization of the learned soft prompt, possibly due to over-parameterization and resulting overfitting, might explain this behaviour. In contrast, LoPA, being more parameter-efficient, explicitly incorporates task and instance information in its design of the soft prompt, leading to better performance.

LoPA also performs on par with LoRA, often within a range of 1 to 4 points of *pass*@1, while roughly using two-thirds of the parameters. Notably, LoPA outperforms LoRA across all models in CruxEval-I, except for LLama-3, with improvements approximately ranging from 2 points in DeepSeekCoder-7B to 4 points in Phi-3. This result might be attributed to the fact that many of the FMs considered here are good knowledge approximators, well-trained on diverse datasets, and demonstrate strong zero-shot generalization. Directly updating a subset of their weights can still lead to catastrophic forgetting, where the models lose previously acquired knowledge [27]. In such cases, soft prompting, as employed by LoPA, can effectively elicit the necessary skills to solve new tasks without compromising existing knowledge [26].

Performance on Code Completion. On the code completion task of MBPP, both IDPG and LoPA improve the performance of the baseline model with nearly equal gains except for LLama-3. However, LoPA achieves this with significantly fewer tunable parameters—approximately half the number used



(a) NLU (b) CruxEval

Figure 4: Performance of LoPA as a function of rank shown for $m = 10$. (a) GLUE Benchmarks and (b) CruxEval tasks (I, O) where ds-1.3 denotes DeepseekCoder-1.3B and phi-2 denotes Phi2-2.7B models. Higher performance and fewer tunable parameters indicate better results.

in CodeGen and only one-fifth of those used by IDPG in DeepseekCoder-7b. This demonstrates that LoPA scales well with the size of the foundation model, maintaining both performance and parameter efficiency. This efficiency is attributed to the low-rank approximation of the instance-specific matrix employed by LoPA. For LLama-3, all tuning approaches led to a drop in performance, possibly due to over-fitting, suggesting that LLama-3 might already be trained on MBPP.

Overview of Results. Averaged over all tasks and foundation models, LoPA showed relative percentage improvements of 28.52% over PT and 20.16% over IDPG, while being outperformed by LoRA by only 0.54%. Notably, LoPA outperformed LoRA in 11 out of 24 cases. Thus, in the test cases we considered, there was no clear systematic advantage to LoRA in terms of accuracy. Given that LoPA requires no task-specific processing at the server—the prompt can be computed anywhere, even at the client, before being sent to the server for processing—we believe LoPA may be a useful alternative to LoRA for some tasks.

4.3 Ablation Study

Performance of LoPA as a function of soft-prompt length. In Figure 3, we study how the length of the soft prompt impacts the performance of LoPA compared to other prompting methods. Increasing the length of the soft prompt corresponds to adding more vectors to the set that represents the soft prompt, thus expanding the offset subspace (See Eq. 2). Whether the added vectors are mutually independent and provide additional useful information depends on the tuning approach to learning them and the offset subspace of the FM.

For instance, PT and IDPG initially see performance improvements with increased prompt length, but performance eventually plateaus or drops due to over-fitting (See PT on SST-2 Fig. 3c and IDPG on CruxEval-O Fig. 3d). In contrast, LoPA does not exhibit significant performance fluctuations with varying prompt lengths (See Fig. 3a-3d). This stability is likely due to the shared component $\mathbf{Z}_S$ acting as a regularizer, preventing over-fitting of the instance-specific soft prompts.

Moreover, LoPA with $m = 5$ outperforms PT and IDPG even when they use longer prompts ($m > 5$) (Refer Fig. 3a, 3b). This result suggests that the dimension of the offset subspace is much smaller, and LoPA can more accurately represent it with its learned vectors.

Performance of LoPA as a function of rank. In Figure 4, we examine how the rank of the proposed low-rank decomposition of the instance-specific component affects the performance. For NLU, we consider three tasks: MRPC, RTE, and SST-2, and observe the performance of `RoBERTa` as the rank increases from 1 to 4 . Performance consistently improves with increasing rank, showing gains of 1% to 2% for SST-2 and up to 8% for MRPC.

In contrast, for CruxEval tasks, increasing the rank does not proportionally improve the performance. We attribute this behaviour to the size of the datasets used to approximate the low-rank matrices. NLU tasks provide thousands of samples, allowing for better approximation of higher-order matrices. However, CruxEval has only a few hundred samples, and increasing the rank introduces more parameters, possibly leading to over-fitting.

Encoder	CruxEval-I	CruxEVal-O
CodeBert(125M)	41.2	32.8
CodeSage(130M)	**43.0**	**34.5**
CodeSage(365M)	42.2	**34.5**

Figure 5: Ablation for Encoder in LoPA with DeepseekCoder-1.3B as the foundation model.

Performance of LoPA as a function of encoder network. In Figure 5, we study the impact on the performance by choosing different transformer backbones for the Encoder network in LoPA. For this experiment, we use $125M$ `CodeBERT` [4], and $130M$ and $365M$ `CodeSAGE` [40] encoder models to generate input encodings, $\mathbf{X}'$ for the CruxEval tasks. We observe that `CodeSAGE` models achieve a 2-point improvement over CodeBERT in both tasks. This improvement can be attributed to `CodeSAGE`'s superior pre-training using Contrastive Learning, which allows for finer distinctions in code representations. Consequently, the soft-prompt vectors $\{\mathbf{z}^k\}$ in LoPA, as functions of the input $\mathbf{X}'$, capture instance-specific nuances more effectively and accordingly exert influence on the model's output. We also tried tuning the encoder model while learning soft prompts but did not find any significant improvements in the performance.

Performance of LoPA with respect to function encoding Z. In Table 3, we evaluate the effect of different functions encoding $\mathbf{Z}$ as a combination of $\mathbf{Z}_I$ and $\mathbf{Z}_S$ on RoBERTa's performance in NLU tasks. The results show that simply concatenating $\mathbf{Z}_S$ and $\mathbf{Z}_I$ performs the worst while the non-linear functions, such as max(.) and the proposed gating mechanism, yield the best performance. We leave the exploration of other non-linear functions for future work.

5 Conclusion

In this paper, we introduced Low-Rank Prompt Adaptation (LoPA), an instance-specific soft-prompting method that outperforms other methods in the prompt-tuning family, and performs on par with full fine-tuning and LoRA, while using fewer tunable parameters. LoPA first uses a low-rank approximation of the instance-specific soft prompt and combines it with a task-specific soft prompt via a gating function. With a more informed way of designing soft prompts, this work aims to position prompt tuning as a powerful alternative to adapter-based methods for user-specific customization of foundation models.

Limitations and Future Work. The main limitation of LoPA is that its effectiveness as an alternative to LoRA was demonstrated on a set of benchmark tasks, but this may not hold for obscure real-life user tasks where LoRA or even full fine-tuning might be necessary. Further investigation into its performance on real-world tasks is part of our future work. In this work, we assumed the learned soft prompt to be prepended as a prefix to the input. Future research could explore the effects of positioning it as a suffix or randomly within the input. Finally, the foundation model combined with LoPA can be viewed as a Conditional Auto-Encoder, where soft prompt vectors exist in a latent subspace rather than an offset subspace. This viewpoint raises intriguing questions, such as whether the observed performance improvements result from inferring and compressing task-specific knowledge and providing it as additional information. Investigating this alternate perspective could lead to further performance enhancements by developing more sophisticated auto-encoding systems.

Broader Impact. Our contribution to new knowledge is the development of a language-model customization method that delivers strong performance while being parameter-efficient. The significance of our work lies in its potential to reduce training and maintenance costs associated with hosting and customizing foundation models. Furthermore, our method enhances user privacy by enabling task-specific customization on the user end rather than the server end.

6 Acknowledgments

This research was supported by the NSF under grant numbers CCF1918651, 2008240, 2131294, and 2212557, NIH CTSA award No. UL1TR003167, and US DOT Tier-1 UTC CYBER-CARE grant #69A3552348332.

References

[1] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[3] Bill Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Third international workshop on paraphrasing (IWP2005)*, 2005.

[4] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.

[5] Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and William B Dolan. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9, 2007.

[6] Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I Wang. Cruxeval: A benchmark for code reasoning, understanding and execution. *arXiv preprint arXiv:2401.03065*, 2024.

[7] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

[8] Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. Warp: Word-level adversarial reprogramming. *arXiv preprint arXiv:2101.00121*, 2021.

[9] Zeyu Han, Chao Gao, Jinyang Liu, Sai Qian Zhang, et al. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*, 2024.

[10] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*, 2021.

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

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

[13] Mojan Javaheripi, Sébastien Bubeck, Marah Abdin, Jyoti Aneja, Sebastien Bubeck, Caio César Teodoro Mendes, Weizhu Chen, Allie Del Giorno, Ronen Eldan, Sivakanth Gopi, et al. Phi-2: The surprising power of small language models. *Microsoft Research Blog*, 2023.

[14] Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34:1022–1035, 2021.

[15] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

[16] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

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

[18] Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*, 2024.

[19] Xiangyang Liu, Tianxiang Sun, Xuanjing Huang, and Xipeng Qiu. Late prompt tuning: A late prompt could be better than many prompts. *arXiv preprint arXiv:2210.11292*, 2022.

[20] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*, 2021.

[21] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. Gpt understands, too. *AI Open*, 2023.

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

[23] Meta. https://ai.meta.com/blog/meta-llama-3/. 2024.

[24] Microsoft. https://azure.microsoft.com/en-us/blog/introducing-phi-3-redefining-whats-possible-with-slms/. 2024.

[25] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*, 2022.

[26] Aleksandar Petrov, Philip HS Torr, and Adel Bibi. When do prompting and prefix-tuning work? a theory of capabilities and limitations. *arXiv preprint arXiv:2310.19698*, 2023.

[27] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*, 2020.

[28] Guanghui Qin and Jason Eisner. Learning how to ask: Querying lms with mixtures of soft prompts. *arXiv preprint arXiv:2104.06599*, 2021.

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

[30] Zhengxiang Shi and Aldo Lipani. Dept: Decomposed prompt tuning for parameter-efficient fine-tuning. *arXiv preprint arXiv:2309.05173*, 2023.

[31] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 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, 2013.

[32] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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

[35] Yeming Wen and Swarat Chaudhuri. Batched low-rank adaptation of foundation models. *arXiv preprint arXiv:2312.05677*, 2023.

[36] Yeming Wen and Swarat Chaudhuri. Batched low-rank adaptation of foundation models. In *ICLR*, 2024.

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

[38] Zhuofeng Wu, Sinong Wang, Jiatao Gu, Rui Hou, Yuxiao Dong, VG Vydiswaran, and Hao Ma. Idpg: An instance-dependent prompt generation method. *arXiv preprint arXiv:2204.04497*, 2022.

[39] Aston Zhang, Yi Tay, Shuai Zhang, Alvin Chan, Anh Tuan Luu, Siu Cheung Hui, and Jie Fu. Beyond fully-connected layers with quaternions: Parameterization of hypercomplex multiplications with $1/n$ parameters. *arXiv preprint arXiv:2102.08597*, 2021.

[40] Dejiao Zhang, Wasi Ahmad, Ming Tan, Hantian Ding, Ramesh Nallapati, Dan Roth, Xiaofei Ma, and Bing Xiang. Code representation learning at scale. *arXiv preprint arXiv:2402.01935*, 2024.

[41] Wei Zhu and Ming Tan. Spt: Learning to selectively insert prompts for better prompt tuning. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

7 Appendix

7.1 Comparison with Parameterized Hypercomplex Multiplication (PHM) layers

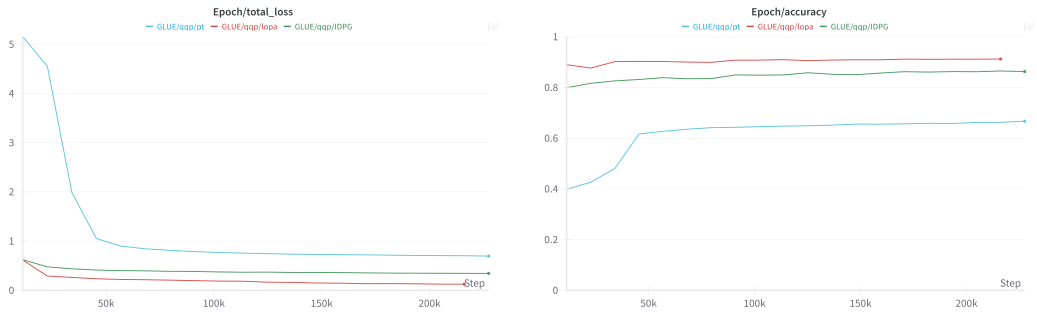
Tuning Method	Tunable Params	SST-2 (acc)	MNLI (acc)	MRPC (acc, F1)	QNLI (acc)	QQP (acc, F1)	RTE (acc)	Avg
LoPA	1.60M	**95.99**	**89.22**	**91.09**	**93.74**	**89.72**	**83.39**	**90.53**
S-IDPG								
+ FC	2.89M	95.30	84.50	78.60	90.48	84.88	77.26	85.17
+ PHM (n=8)	0.37M	95.07	83.46	76.12	85.45	77.35	67.14	80.77
+ PHM (n=16)	0.20M	94.61	83.66	76.68	81.16	80.39	65.34	80.31
+ PHM (n=32)	0.17M	94.72	81.45	74.99	84.59	81.90	68.23	80.98

Table 4: Performance on GLUE tasks. We report the average of accuracy and F1 for both MRPC and QQP. For all the other tasks, we report accuracy. **Bold** denotes the best-performing tuning method for the given model. FC represents the fully-connected layers and PHM represents the hypercomplex multiplication layers parameterised by n.

In this section, we consider PHM layers as an alternative to low-rank decomposition in LOPA to reduce parameter complexity. We carry out an ablation study on NLU tasks where we implement IDPG with PHM layers as $\mathbf{Z}_I = \texttt{PHM}_W(\mathbf{X}')$ with $W = \sum_{i=1}^{n} A_i \bigotimes B_i$, where $\bigotimes$ represents the Kroneckr product between learnable matrices A_i, B_i and n represents the hyper-parameter balancing the parameter complexity and extent of factorisation in the Kronecker product. We can observe in Table 4 that while PHM layers reduce parameters, they also lead to a significant performance drop of approximately 10 points on average compared to LOPA. This drop may be due to the structural bias in W imposed by Kronecker factorisation of PHM layers, which could limit expressiveness [39] in comparison to Fully-Connected layers of DNN. We want to point out that for a further reduction in trainable parameters, LoPA can also be used in conjunction with PHM layers.

7.2 Convergence Analysis of Soft-Prompting Approaches

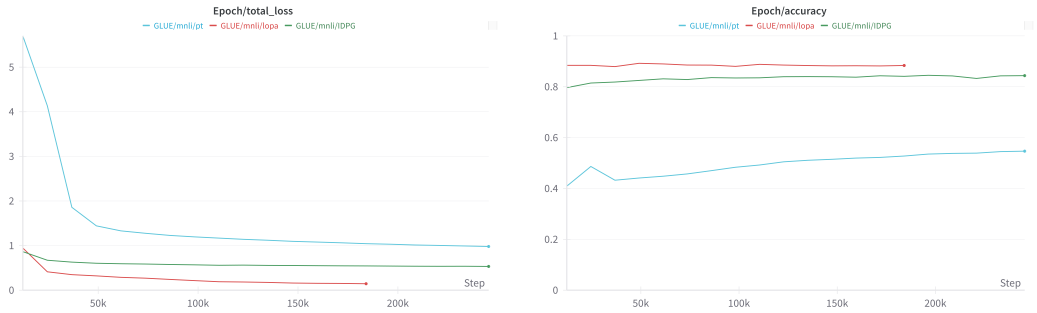
We present plots in Figures 6-8 comparing the training loss and performance on NLU tasks (QQP, QNLI, MNLI) for Prompt Tuning (PT), IDPG, and LOPA. The results show that instance-dependent methods like IDPG and LOPA converge faster than traditional prompt tuning. Moreover, LOPA converges faster and achieves higher accuracy or F1 scores compared to IDPG.



(a) Training Loss

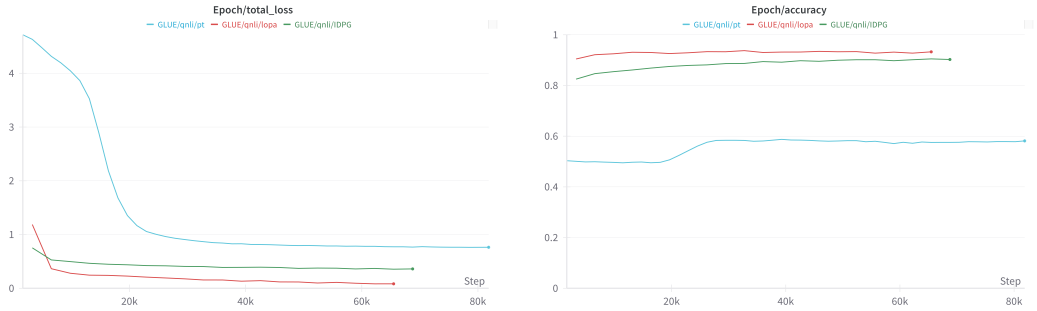
(b) Validation Accuracy

Figure 6: Convergence plots for the PEFT approaches Prompt-tuning (PT), IDPG and the proposed LOPA on the NLU task QQP.



(a) Training Loss (b) Validation Accuracy

Figure 7: Convergence plots for the PEFT approaches Prompt-tuning (PT), IDPG and the proposed LOPA on the NLU task MNLI.



(a) Training Loss (b) Validation Accuracy

Figure 8: Convergence plots for the PEFT approaches Prompt-tuning (PT), IDPG and the proposed LOPA on the NLU task QNLI.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Yes, the abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Yes, the limitations are discussed in the conclusion section.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: The paper does not present any theoretical results.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.

- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We include all the necessary set of hyper-parameters and training guidelines to replicate the results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code is publicly released on Github.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.

- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Yes, the hyper-parameters are discussed to replicate results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Running multiple experiments on many large language models across all tasks is expensive.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The amount of computer resources used is discussed in the paper.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.

- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Research adheres to NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss the potential societal impact of our work in the conclusion section.

    Guidelines: Broader impact is discussed in the paper.

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: No new data or model is released.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Models and datasets with their licenses used are respected and cited.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: Neither crowdsourcing nor research with human subjects was done.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer:[NA]

Justification: Neither crowdsourcing nor research with human subjects was done.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.