Punctuation-Steered Representation Fine-Tuning

Zheng Gong[1] **and Ying Sun**[2*] **and Ping Li**[3] **and Zheng Yi**[3] **and Zhefeng Wang**[3]
[1]Thrust of Artificial Intelligence, Hong Kong University of Science and Technology (Guangzhou)
[2]The 63rd Research Institute, National University of Defense Technology
[3]Huawei Cloud
Correspondence: sunying@nudt.edu.cn

Abstract

Representation Fine-tuning (ReFT), a recently proposed parameter-efficient fine-tuning (PeFT) method, significantly improves parameter efficiency by modifying the representation space alone. However, directly applying ReFT, which alters a fixed number of representations at the beginning and end positions of each layer, results in suboptimal performance for two reasons. (i) The impact of these fixed-position representations on the output is uncertain; (ii) As the sequence length increases, fine-tuning a fixed number of representations may have diminishing effects on the final results. Based on our observations that punctuation plays a crucial role in integrating representations from preceding layers and modulating those of subsequent layers, we introduce Punctuation-steered Representation Fine-tuning (PuReFT), a straightforward yet powerful approach that additionally fine-tunes punctuation representations to achieve performance improvements. Extensive evaluations on common-sense, arithmetic, and code datasets demonstrate the effectiveness and versatility of PuReFT. Furthermore, our analysis of its training speed and memory overhead confirms its greater ease of use and efficiency.

1 Introduction

Large Language Models (LLMs) (Yan et al., 2025; Wei et al., 2025; Yu et al., 2026; Xin et al., 2025; Chen et al., 2024; Gong and Sun, 2024; Zhang et al., 2025) have gained immense popularity and demonstrated remarkable performance across a wide range of tasks. However, adapting these colossal models to specific downstream tasks remains a significant challenge, primarily due to their prohibitive size and computational demands. To address this, Parameter-efficient Fine-Tuning (PeFT) methods have emerged as a dominant paradigm (Houlsby et al., 2019; Liu et al., 2022; Zhang et al., 2023). Among these, Low-Rank Adaptation (LoRA, Hu et al. (2022)) is arguably the most widely adopted approach. LoRA works by freezing the pre-trained model weights and injecting trainable rank-decomposition matrices into each layer, thereby significantly reducing the number of parameters that need to be updated.

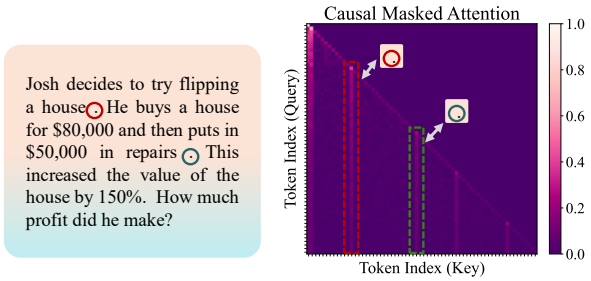


Figure 1: An example of attention heatmap for LLaMA-3-8B model. The punctuation tokens exhibit high self-attention scores and serve as focal points of attention for preceding tokens in the sequence.

While LoRA has proven effective, Representation Fine-tuning (ReFT, Wu et al. (2024)) offers an even more parameter-efficient alternative. Instead of modifying the model's weights, ReFT directly edits the hidden representations of a fixed number of tokens at specific, pre-determined positions (typically at the beginning and end) of the input sequence within each layer. This approach achieves comparable performance to LoRA with a much smaller parameter footprint. However, a key limitation of the original ReFT method is its reliance on fixed-position, fixed-quantity representations, which can lead to suboptimal performance. These pre-defined positions may not always be semantically relevant, and their influence on the final output can diminish as the input sequence grows, particularly in long-sequence tasks.

Our analysis of the attention mechanisms in LLMs, as depicted in Figure 1, reveals a critical insight: punctuation marks are disproportionately

*Corresponding author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 1–8
July 2-7, 2026

important in the flow of information. We observe that the token representations at punctuation positions often exhibit strong self-attention, suggesting they carry a wealth of rich semantic information. Furthermore, other tokens in the sequence frequently allocate relatively high attention scores to punctuation-related tokens, indicating that punctuation marks act as semantic anchors, facilitating the integration of preceding information and the modulation of representations for subsequent layers. This observation is further supported by recent studies on LLM generation quality (Seleznyov et al., 2025; Razzhigaev et al., 2025) and KV cache eviction (Chen et al., 2025), which highlight the crucial role of punctuation in summarizing segment content. Inspired by these findings, we introduce Punctuation-steered Representation Fine-tuning (PuReFT), a novel and straightforward yet powerful ReFT approach by fine-tuning the representations of punctuation tokens. By leveraging the semantic anchoring properties of punctuation, PuReFT ensures that the fine-tuned representations are more semantically meaningful and robustly impactful across varying sequence lengths.

We conducted extensive experiments on the LLaMA-3-8B and LLaMA-2-13B models across diverse datasets, including commonsense reasoning, arithmetic, and code generation. Our results demonstrate that PuReFT consistently outperforms the original ReFT method and achieves performance on par with or superior to LoRA, all while maintaining a remarkably low parameter overhead[1].

2 Method

In this section, we first introduce the representation fine-tuning framework in Section 2.1. We then extensively illustrate our method, PuReFT, for fine-tuning punctuation representations to achieve LLM downstream task adaptation in Section 2.2.

2.1 Representation Fine-Tuning Framework

A Transformer-based LLM, given an input prompt $x = (x_1, ..., x_n)$ consisting of n tokens, uses a stack of L transformer blocks to embed the input tokens into multi-layer hidden representations $\mathbf{h}^{(l)} = (\mathbf{h}_1^{(l)}, ..., \mathbf{h}_n^{(l)})$, where $\mathbf{h}_j^{(l)} \in \mathbb{R}^d$ is the representation of token x_j at layer l. The representation fine-tuning framework modifies these token hidden representations by defining a set of interventions $\mathcal{I}$.

[1]Our code is available at github.com/KellyGong/PuReFT.

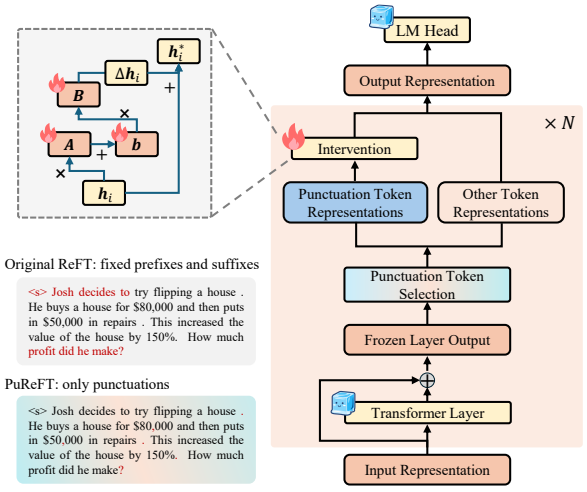


Figure 2: The overview of the procedural pipeline for our punctuation-steered representation fine-tuning.

Each intervention $I \in \mathcal{I}$ is represented as a triplet:

$$I := (\phi_I, \mathcal{P}_I, l), \tag{1}$$

which specifies that during inference, function ϕ_I with learnable parameters is applied to the token representations $\{\mathbf{h}_j^{(l)}\}_{j \in \mathcal{P}_I}$ at layer l and within the position set $\mathcal{P}_I$. This process modifies the token representations, thereby enabling the LLM to adapt to a specified task. The training objective is to maximize the likelihood of the target length-m generated sequence:

$$\max_{\mathcal{I}} \sum_{i=1}^{m} \log p\left(x_{n+i} \mid x\right). \tag{2}$$

2.2 Punctuation-Steered ReFT

A key design in ReFT is selecting the token positions $\mathcal{P}_I$ to be fine-tuned for each intervention I. Wu et al. (2024) directly select a fixed number of prefix and suffix tokens from the input prompt x, specifically $\mathcal{P}_I^{\text{fix}} = \{0, ..., k\} \cup \{n-k+1, ..., n\}$. However, this approach, relying on fixed-position and fixed-quantity representations, can lead to suboptimal performance. These predetermined positions may not always be semantically relevant, and their influence can diminish in longer sequences, particularly in tasks involving extensive context.

Beyond previous findings that removing punctuation can affect model generation performance (Seleznyov et al., 2025; Razzhigaev et al., 2025), we observe in Figure 1 that punctuation tokens exhibit strong self-attention and receive significant attention from other tokens. This highlights their critical role in integrating preceding information

Method (Param.) \ Dataset (Len.)		Arithmetic (Accuracy ↑)				Common Sense (Accuracy ↑)					
		GSM8K 60	SVAMP 40	MAWPS 37	Avg. 46	WG 55	HSwag 218	ARC-E 77	ARC-C 87	OQA 63	Avg. 100
LLaMA-3-8B	-	24.7	36.5	41.2	34.1	32.5	65.8	82.3	68.5	67.0	63.2
+ LoRA (r=32)	0.34%	71.0	79.5	90.3	80.3	**88.9**	94.5	93.4	**85.3**	89.0	90.0
+ LoRA (r=4)	0.04%	70.2	78.7	89.1	79.3	86.9	92.3	90.7	80.8	87.2	87.6
+ DoRA (r=4)	0.05%	**71.8**	79.0	90.3	80.4	88.6	93.5	**93.8**	84.8	88.8	89.9
+ ReFT (r=4)	0.02%	70.8	79.1	89.5	79.8	83.6	89.7	91.8	81.5	84.6	86.2
+ CRFT (r=4)	0.02%	69.7	77.6	89.6	79.0	84.5	90.2	91.4	80.9	84.7	86.2
+ **PuReFT-s** (r=4)	0.02%	71.3	**81.2**	90.3	80.9	86.2	93.6	93.4	82.7	86.2	88.4
+ **PuReFT-u** (r=4)	0.02%	71.7	81.1	**90.5**	**81.1**	87.8	**95.1**	**93.8**	85.2	**89.3**	**90.2**
LLaMA-2-13B	-	6.2	17.3	15.9	13.1	18.5	26.3	32.5	29.6	31.4	27.7
+ LoRA (r=32)	0.40%	49.7	58.3	80.7	62.9	**88.6**	**91.3**	**90.1**	80.7	85.4	**87.2**
+ LoRA (r=4)	0.05%	49.2	57.9	80.4	62.5	87.5	90.9	88.7	79.4	84.1	86.1
+ DoRA (r=4)	0.06%	49.3	58.1	80.7	62.7	88.4	91.0	88.5	80.3	84.6	86.6
+ ReFT (r=4)	0.04%	48.4	57.1	80.2	61.9	84.8	86.7	86.9	77.5	81.7	83.5
+ CRFT (r=4)	0.04%	47.5	56.7	79.8	61.3	85.3	87.0	87.3	78.2	81.5	83.9
+ **PuReFT-s** (r=4)	0.04%	49.3	58.5	**81.0**	62.9	87.6	89.3	88.2	78.4	82.7	85.2
+ **PuReFT-u** (r=4)	0.04%	**49.8**	**58.7**	80.6	**63.0**	88.4	90.5	**90.1**	**81.0**	**85.6**	87.1

Table 1: The performance comparison on arithmetic and common sense reasoning datasets with two LLM backbone: LLaMA-3-8B and LLaMA-2-13B. The `Param.` (%) is calculated by dividing the number of trainable parameters by the total number of parameters in the base LLM. The `Len.` represents the average number of tokens for input prompts in each dataset, with tokens generated by the LLaMA-3 tokenizer.

and modulating representations for subsequent layers. Consequently, we propose using punctuation tokens as key representations for the intervention. We define the positions of punctuation tokens as:

$$\mathcal{P}_{\text{punc}} = \{i | t_i \in \mathcal{D}, i \in \{1, ..., n\}\}, \quad (3)$$

where $t_i \in \mathcal{D}$ denotes token t_i belonging to the delimiter set [".", ",", "?", "!", ";", ":"].

Based on the selection of punctuation as the key representations, it is necessary to modulate them to ensure their influence on downstream tasks can be accurately calibrated. To achieve this, we model the adjustment of these key representations as $\Delta\mathbf{h}$ through a learnable function ϕ_I and the training objective from Formula (2). Following (Wu et al., 2024; Hu et al., 2022), we restrict the optimization space to a low-rank linear subspace. Specifically,

$$\phi_I(\mathbf{h}_i) = \begin{cases} \mathbf{h}_i + \mathbf{B}(\mathbf{A}\mathbf{h}_i + \mathbf{b}), & i \in \mathcal{P}_{\text{Punc}}, \\ \mathbf{h}_i, & i \notin \mathcal{P}_{\text{Punc}}, \end{cases} \quad (4)$$

where $\mathbf{A} \in \mathbb{R}^{r \times d}, \mathbf{B} \in \mathbb{R}^{d \times r}$ are two learnable low-rank matrices (rank $r \ll d$), and $\mathbf{b} \in \mathbb{R}^r$ is a learnable bias vector.

3 Experiments

In this section, we conduct experiments to prove the effectiveness of our PuReFT. We include **implementation details**, **inference time analysis** and **hyperparameter sensitivity** in Appendix A.

Benchmarks. We conduct experiments across three distinct task types. (i) For **arithmetic reasoning**, we fine-tune LLMs on the Math10K dataset following (Khaki et al., 2025) before evaluating them on three benchmarks: GSM8K (Cobbe et al., 2021), MAWPS (Koncel-Kedziorski et al., 2016), SVAMP (Patel et al., 2021). (ii) For **commonsense reasoning**, we fine-tune LLMs on the CSR170K dataset, consistent with prior work (Hu et al., 2023), and evaluate them on a suite of five downstream datasets: **W**ino**G**rande (Sakaguchi et al., 2021), **H**ella**S**wag (Zellers et al., 2019), **ARC-E**asy and **ARC-C**hallenge (Clark et al., 2018), **O**penbook**QA** (Mihaylov et al., 2018). (iii) To address **code generation**, LLMs are fine-tuned on the CodeFeedback (Chen et al., 2021) and evaluated using HumanEval/HumanEval+ (Liu et al., 2023; Zheng et al., 2024) benchmark. The results of code generation are illustrated in Appendix A.2.

Baselines and our variants. We use LLaMA-3-8B and LLaMA-2-13B as the base model for fine-tuning. We compare our method with two PeFT approaches, LoRA (Hu et al., 2022) and DoRA (Liu et al., 2024), as well as two representation fine-tuning approaches, ReFT (Wu et al., 2024) and CRFT (Huang et al., 2025a). Our PuReFT method has two variants: The "-s" variant fine-tunes the same number of token representations as both ReFT and CRFT, ensuring a fair comparison. The "-u" variant, on the other hand, fine-tunes all

punctuation tokens. All results are averaged over three runs with distinct random seeds.

Main Results. In Table 1, our proposed method, PuReFT, achieves superior or competitive performance across multiple benchmarks in the Arithmetic, Code Generation, and Common Sense domains while utilizing significantly fewer learnable parameters than LoRA (r=32). Furthermore, PuReFT consistently outperforms existing representation learning methods, *i.e.*, ReFT and CRFT, on the majority of tasks. These results validate our hypothesis that fine-tuning the representations of key punctuation tokens is an effective strategy for steering LLMs to specific downstream tasks. Our analysis also yielded several notable findings.

Observation (i): *Reducing the number of learnable parameters in LoRA leads to a decline in learning performance.* This effect is more pronounced on the longer-text Common Sense datasets compared to the shorter-text Arithmetic datasets (2.4% *v.s.* 1.0% with LLaMA-3-8B as backbone). The performance trends of LoRA under different ranks in Figure 4, further corroborate this finding. This phenomenon can be attributed to LoRA's need to fine-tune representations for all tokens. Therefore, a greater number of parameters is required to adapt a larger number of representations effectively.

Observation (ii): *The original representation learning fine-tuning methods are more effective on short texts.* Taking LLaMA-3-8B in Table 1 as an example, ReFT (r=4) achieves a performance of 79.8%, comparable to LoRA (r=32) at 80.3%, on the short-text Arithmetic task. However, on the longer-text Common Sense datasets, ReFT (r=4) attains only 86.6%, significantly underperforming LoRA (r=32), which reaches 90.0%. This suggests that by only fine-tuning the prefix and suffix representations, the original ReFT struggles to influence the global representations required for effective adaptation in long-text scenarios.

Observation (iii): *Intervening with punctuation tokens yields significant performance gains on both long- and short-text tasks.* For the Arithmetic dataset, our PuReFT method achieves an average accuracy that is 0.7% higher than LoRA (r=32) while using substantially fewer parameters for fine-tuning LLaMA-3-8B. Moreover, on the Common Sense datasets, PuReFT's average accuracy is 4.0% higher than that of ReFT (r=4) and slightly surpasses LoRA (r=32). This underscores that PuReFT, by injecting representational interventions at semantically meaningful intervals within long texts, can effectively adapt the model to downstream tasks with minimal parameter growth.

Method	Batch Time (s)	VRAM (GB)	TTFT (s)
LLaMA-3-8B	-	-	*0.184*
LoRA ($r = 32$)	2.29	50.89	**0.202**
LoRA ($r = 4$)	2.30	50.60	<u>0.203</u>
DoRA ($r = 4$)	2.92	60.66	0.247
ReFT ($r = 4$)	**2.13**	**49.55**	0.207
CRFT ($r = 4$)	3.16	<u>49.61</u>	0.314
PuReFT-s ($r = 4$)	<u>2.17</u>	**49.55**	0.207
PuReFT-u ($r = 4$)	<u>2.17</u>	49.83	0.224

Table 2: Training overhead is measured on CSR170K (Batch size 8, Max length 1024 with LLaMA-3-8B) in terms of batch training time and max CUDA memory (VRAM) usage. Inference overhead is measured on HellaSwag (Batch size 8 with LLaMA-3-8B) in terms of Time to First Token (TTFT).

Training Overhead Analysis. In Table 2, PuReFT and ReFT exhibit superior training efficiency, outperforming parameter-based approaches like LoRA and DoRA in both per-batch time and memory usage. This is due to their strategy of fine-tuning only a subset of representations, which reduces computational and memory overhead. DoRA performs poorly in both speed and memory, as it relies on weight decomposition and normalization. While CRFT's speed is significantly lower, since it relies on a token-sorting process that is inefficient for GPU devices.

Inference Time Analysis. The inference time required by a LLM when processing a new prompt primarily consists of two phases: prefill and generation. The prefill phase converts all tokens of the input prompt into a key-value (KV) cache, which represents the key and value states for each layer. The generation phase then iteratively produces new tokens based on this KV cache. Our method, PuReFT, fine-tunes the representations exclusively during the prefill stage, thus its impact is limited to the efficiency of this phase. To evaluate the effect of different fine-tuning methods on prefill efficiency, we measure the Time to First Token (TTFT), which is the time required to generate the first token. Figure 3 illustrates the TTFT for various fine-tuning approaches on the Llama-3-8B model using the HellaSwag dataset. The results show that our two PuReFT variants, PuReFT-s and PuReFT-u, have only a minor impact on TTFT compared to the original Llama-3-8B, which has a TTFT of 0.184s. PuReFT-s adds an overhead of only 0.023s, and

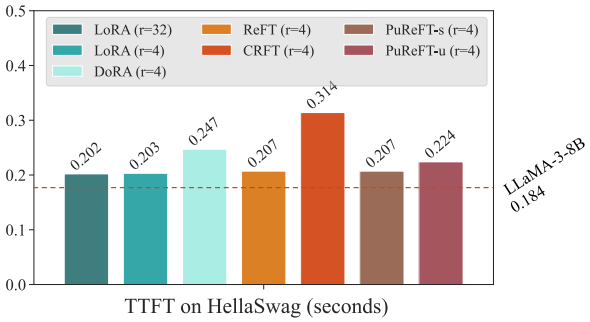


Figure 3: Time To First Token (TTFT) on HellaSwag (Batch size 8 with LLaMA-3-8B).

PuReFT-u adds 0.04s. This represents a significant efficiency advantage over CRFT, which requires sorting intermediate attention values and incurs a much larger overhead of 0.13s. Our method also compares favorably to LoRA, with PuReFT only adding a slightly larger overhead of 0.005s to 0.02s. In contrast, DoRA introduces efficiency impacts due to the additional weight normalization computations and extra parameters for direction and magnitude, which are part of its training process.

4 Related Work

To circumvent the prohibitive computational costs of full-parameter fine-tuning, Parameter-Efficient Fine-Tuning (PeFT) has emerged as a cornerstone for adapting LLMs to downstream tasks (Houlsby et al., 2019; Liu et al., 2022; Zhang et al., 2023; Huang et al., 2025b). A prominent PeFT technique is LoRA (Hu et al., 2022), which freezes the pre-trained model weights and solely trains injected, low-rank matrices. Building on this, Weight-Decomposed Low-Rank Adaptation (DoRA, Liu et al. (2024)) enhances this process by decoupling the weight update into magnitude and direction vectors, applying LoRA specifically to the directional component for more effective training. Departing from such weight modifications, the recently proposed Representation Fine-tuning (ReFT) (Wu et al., 2024) operates by directly manipulating a sparse subset of hidden representations within the LLM. CRFT (Huang et al., 2025a) fine-tunes the most important token representations, which are ranked layer-wise by attention values or gradients. However, to the best of our knowledge, no existing work has investigated the efficient adaptation of LLMs through fine-tuning punctuation.

5 Conclusion and Future Work

In this work, we introduced Punctuation-steered Representation Fine-tuning for adapting LLMs. PuReFT leveraged the critical role of punctuation marks as semantic anchors to guide the fine-tuning process. Our extensive evaluations demonstrated that PuReFT not only surpasses the performance of the original ReFT but also exhibits superior robustness, particularly on long-sequence tasks. In our future work, inspired by the recent advances of reinforcement learning (Zhang et al., 2026) and explainability (Ji et al., 2025) on LLMs, we plan to investigate the effectiveness of PuReFT on these aspects to enhance the LLMs.

Limitations

Our current methodology is primarily confined to linear interventions, which may constrain its ability to capture and manipulate more intricate, non-linear relationships within the representation space. Furthermore, while we demonstrate the effectiveness of our approach on models up to 13B parameters, its scalability and efficacy remain to be rigorously validated on larger-scale models, such as those exceeding 70B parameters. In future work, we plan to explore non-linear representation editing techniques and conduct comprehensive evaluations on these larger models to further establish the framework's robustness.

Acknowledgement

This work is partly supported by the National Natural Science Foundation of China (No. 62306255).

References

Guoxuan Chen, Han Shi, Jiawei Li, Yihang Gao, Xiaozhe Ren, Yimeng Chen, Xin Jiang, Zhenguo Li, Weiyang Liu, and Chao Huang. 2025. Sepllm: Accelerate large language models by compressing one segment into one separator. In *Forty-second International Conference on Machine Learning*.

Liyi Chen, Panrong Tong, Zhongming Jin, Ying Sun, Jieping Ye, and Hui Xiong. 2024. Plan-on-graph: Self-correcting adaptive planning of large language model on knowledge graphs. *Advances in Neural Information Processing Systems*, 37:37665–37691.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Zheng Gong and Ying Sun. 2024. Graph reasoning enhanced language models for text-to-sql. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2447–2451.

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. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Lee. 2023. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5254–5276.

Chenxi Huang, Shaotian Yan, Liang Xie, Binbin Lin, Sinan Fan, Yue Xin, Deng Cai, Chen Shen, and Jieping Ye. 2025a. Enhancing chain-of-thought reasoning with critical representation fine-tuning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23173–23195.

Weizhong Huang, Yuxin Zhang, Xiawu Zheng, Jing Lin, Yiwu Yao, Rongrong Ji, and 1 others. 2025b. Dynamic low-rank sparse adaptation for large language models. In *The Thirteenth International Conference on Learning Representations*.

Yang Ji, Ying Sun, Yuting Zhang, Zhigaoyuan Wang, Yuanxin Zhuang, Zheng Gong, Dazhong Shen, Chuan Qin, Hengshu Zhu, and Hui Xiong. 2025. A comprehensive survey on self-interpretable neural networks. *Proceedings of the IEEE*.

Samir Khaki, Xiuyu Li, Junxian Guo, Ligeng Zhu, Konstantinos N Plataniotis, Amir Yazdanbakhsh, Kurt Keutzer, Song Han, and Zhijian Liu. 2025. Sparselora: Accelerating llm fine-tuning with contextual sparsity. In *Forty-second International Conference on Machine Learning*.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. Mawps: A math word problem repository. In *Proceedings of the 2016 conference of the north american chapter of the association for computational linguistics: human language technologies*, pages 1152–1157.

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

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36:21558–21572.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094.

Anton Razzhigaev, Matvey Mikhalchuk, Temurbek Rahmatullaev, Elizaveta Goncharova, Polina Druzhinina, Ivan Oseledets, and Andrey Kuznetsov. 2025. Llm-microscope: Uncovering the hidden role of punctuation in context memory of transformers. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 7757–7764.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Mikhail Seleznyov, Mikhail Chaichuk, Gleb Ershov, Alexander Panchenko, Elena Tutubalina, and Oleg Somov. 2025. When punctuation matters: A large-scale comparison of prompt robustness methods for llms. *arXiv preprint arXiv:2508.11383*.

Shuyue Wei, Yongxin Tong, Zimu Zhou, Yi Xu, Jingkai Gao, Tongyu Wei, Tianran He, and Weifeng Lv. 2025. Federated reasoning llms: a survey. *Frontiers of Computer Science*, 19(12):1912613.

Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D Manning,

and Christopher Potts. 2024. Reft: Representation finetuning for language models. *Advances in Neural Information Processing Systems*, 37:63908–63962.

Haoran Xin, Ying Sun, Chao Wang, and Hui Xiong. 2025. Llmcdsr: Enhancing cross-domain sequential recommendation with large language models. *ACM Transactions on Information Systems*, 43(5):1–33.

Mengyi Yan, Yaoshu Wang, Xiaohan Jiang, Haoyi Zhou, and Jianxin Li. 2025. Towards uncertainty-calibrated structural data enrichment with large language model for few-shot entity resolution. *Frontiers of Computer Science*, 19(11):1911376.

Yanke Yu, Jin Li, Ying Sun, Ping Li, Zhefeng Wang, and Yi Zheng. 2026. Discovering decoupled functional modules in large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 34503–34511.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800.

Mingxu Zhang, Dazhong Shen, and Ying Sun. 2025. Atomdisc: An atom-level tokenizer that boosts molecular llms and reveals structure–property associations. *arXiv preprint arXiv:2512.03080*.

Mingxu Zhang, Huicheng Zhang, Jiaming Ji, Yaodong Yang, and Ying Sun. 2026. Enhance the safety in reinforcement learning by adrc lagrangian methods. *arXiv preprint arXiv:2601.18142*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*.

Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhu Chen, and Xiang Yue. 2024. Opencodeinterpreter: Integrating code generation with execution and refinement. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 12834–12859.

A Appendix

A.1 Implementation Details

We optimize all fine-tuning methods on the training sets of benchmarks, *i.e.*, Math10K, CodeFeedback and CSR170K, by selecting learning rate from $\{1 \times 10^{-4}, 5 \times 10^{-5}, 3 \times 10^{-5}, 1 \times 10^{-5}\}$. All PeFT methods fine-tune the projections of query, key, value, and output (`QKVO`) using a rank of either 32 or 4, with no dropout applied. The scaling factor, α, is set to double the value of the rank. All ReFT methods (including ours) use a rank of 4. As for ReFT, both the prefix and suffix token representations selected for intervention are set to 7. To ensure a fair comparison with ReFT, CRFT selects the 14 most critical representations in each layer for fine-tuning. We select the CRFT variant based on Multi-Referential Filtering attention values as our baseline model because the gradient-dependent variant of CRFT requires an additional forward pass to obtain gradients, which would severely impact efficiency. For our PuReFT variants, we utilize tokens from the prefix and suffix to complement the number of diverse punctuation tokens. This ensures that every sentence sample within the same batch has an identical number of fine-tuned representations, which allows us to leverage efficient matrix operations. All experiments are conducted on an Ubuntu 20.04.5 TLS Linux server, equipped with a 2.20GHz Intel Xeon Gold 5220R CPU and an NVIDIA A100 GPU with 80G memory. The maximum duration required for training across all experiments was 4 GPU hours, with each testing task completed within 1 GPU hour.

A.2 Code Generation

Table 4 presents the performance of our method and baselines on the Code Generation Datasets: HumanEval and HumanEval+. As the table shows, there's a significant performance gap between the representation fine-tuning methods, ReFT and CRFT, and the parameter-based fine-tuning methods, LoRA and DoRA. Our approach substantially improves the performance of representation-based fine-tuning, bringing it close to LoRA (r=32), which uses a significantly larger number of tunable parameters. We attribute this improvement to the higher frequency of punctuation marks in code datasets compared to common text datasets. These punctuation marks carry stronger code-specific information. Therefore, fine-tuning the representa-

Dataset	Seq.	Gen.	Bs.	Ep.	Scheduler	Warmup
Math10K	512	256	16	3	Cosine	0.04
CodeFeedback	1,024	1,024	8	1	Cosine	0.04
CSR170K	1,024	256	8	1	Cosine	0.04

Table 3: Training and evaluation hyperparameters for each benchmark. The learning rates are scheduled with a cosine scheduler with warm up ratio 0.04 and optimized using a grid search, with the search space described in Section A.1. The `Seq.` and `Gen.` denote the truncation lengths of input prompts and generation responses, respectively. The `Bs.` and `Ep.` represent the batch size and epochs for training, respectively.

Method \ Dataset	Code Generation (Pass@1 ↑)		
	HumanEval	HumanEval+	Avg.
LLaMA-3-8B	37.5	33.8	35.7
+ LoRA (r=32)	**45.6**	**38.4**	**42.0**
+ LoRA (r=4)	44.1	37.5	40.8
+ DoRA (r=4)	44.6	38.0	41.3
+ ReFT (r=4)	41.7	36.8	39.3
+ CRFT (r=4)	39.4	36.5	38.0
+ **PuReFT-s** (r=4)	44.9	38.1	41.5
+ **PuReFT-u** (r=4)	45.1	38.3	41.7

Table 4: The performance comparison on code generation datasets with LLaMA-3-8B.

tions of these punctuation marks helps to boost the final code generation results.

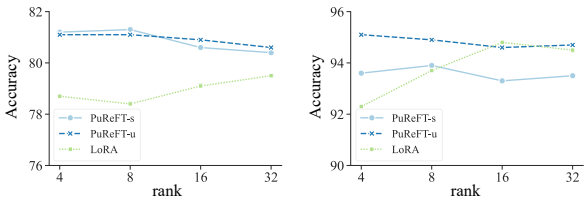
Note that while both LoRA and DoRA can be merged with the original model weights to achieve the same inference speed as the base model, this is often not a practical approach in real-world deployment. In production environments, it is frequently necessary to serve multiple fine-tuned adapters from a single base model instance, for example, to handle different tasks or A/B test various fine-tuning configurations. In such cases, the ability to dynamically load and unload the fine-tuned weights without merging them is crucial. Therefore, our testing of the unmerged LoRA and DoRA holds practical significance as it directly measures the efficiency impact of these methods in a dynamic, multi-task serving scenario.

A.3 Hyperparameter Sensitivity

Existing widely-used parameter-efficient fine-tuning methods, such as LoRA and DoRA, as well as representation tuning approaches including our PuReFT, optimize models for downstream tasks by operating in a low-rank space. A key hyperparameter influencing final performance is the rank r. Figure 4 illustrates the performance of our method across different ranks. As shown in the figure, both of our variants, PuReFT-s and PuReFT-u, achieve their best performance at r=4. Performance degrades when r is excessively large, primarily because we only fine-tune the representations of a small subset of tokens, which inherently limits the number of input features. Using too many parameters under this constraint may increase the risk of overfitting, thereby impairing overall performance.

B LLM Usage

The LLM was employed solely for the purpose of linguistic polishing and enhancing the readability of the manuscript. All intellectual contributions



(a) SVAMP (b) HellaSwag

Figure 4: Effects of different rank on SVAMP dataset and HellaSwag dataset (with LLaMA-3-8B).

pertaining to the central ideas, methodological design, experimental work, and analytical insights originate entirely from the authors.