An Efficient Row-Based Sparse Fine-Tuning with Low Quantization Error

Cen-Jhih Li [1] **Aditya Bhaskara** [1]

Abstract

Fine-tuning is essential for adapting large language models to downstream tasks, but can be costly for users with limited resources. To address this, Sparse Fine-tuning (SpFT) and Low-rank adaptation (LoRA) have been widely adopted for efficient fine-tuning. In this work, we propose a new SpFT framework inspired by neural network pruning: we identify important neurons using structural pruning and fine-tune only the associated weights. Experiments on common language tasks show our method improves SpFT's memory efficiency by 20–50% while matching the accuracy of state-of-the-art methods like LoRA's variants.

1. Introduction

The paradigm of *pre-training followed by fine-tuning* has seen tremendous success in the last few years. Very large models (often referred to as foundation models) are first trained, typically using very large amounts of data and computational resources, using self-supervised learning approaches (Dosovitskiy, 2020; Achiam et al., 2023; Dubey et al., 2024; Zhou et al., 2024). When building a model for a new task (which could be a supervised learning task), the idea is to start with the foundation model and then tune its parameters, possibly after adding additional classification layers, by training using task-specific data. The pre-train then fine-tune paradigm has been shown to have significant advantages over training a new model from scratch for the new task. Often, high accuracy can be obtained using much smaller datasets for the new task.

Despite the success, fine-tuning a model with billions of parameters requires access to heavy computational resources, even when the task datasets are fairly small. Fortunately, studies (e.g., (Panigrahi et al., 2023) and references therein) show that fine-tuning only a small fraction of parameters can be effective. Parameter-efficient fine-tuning (PEFT) methods have thus been proposed to carry out this idea and address the challenge of making fine-tuning more accessible (Lialin et al., 2023). A leading PEFT approach, Low-Rank Adaptation (LoRA, Hu et al. 2022), achieves memory efficiency by simply making low-rank updates to the weight matrices in the different layers. Another class of PEFT methods is *sparse* fine-tuning (SpFT, Sung et al. 2021; Guo et al. 2021; Ansell et al. 2022; Nikdan et al. 2024), which learns a sparse matrix, typically an unstructured one, for updating the pre-trained weights. However, SpFT typically incurs higher memory costs than LoRA during the fine-tuning process, because of the unstructured sparsity. Several works aim to mitigate the memory complexity of SpFT (Mofrad et al., 2019; Holmes et al., 2021; Nikdan et al., 2023; 2024), often at the cost of increased running time and more complex implementations of sparse kernels. Besides PEFTs, techniques like Zeroth-Order optimization (Malladi et al., 2023; Guo et al., 2024b) and quantization (Gholami et al., 2022; Dettmers et al., 2022; 2024) can further enhance memory and training efficiency for fine-tuning, including LoRA and SpFT.

As LLMs increase in scale, advancing efficient sparse matrix computation, PEFT, and efficient training remains a crucial problem. Towards this goal, we study the question: *Can sparse fine-tuning be improved to create a memory- and parameter-efficient framework, while avoiding additional implementations of sparse operations and without increasing the training time complexity?* We answer this question in the affirmative, by proposing a new SpFT framework for fine-tuning LLMs that achieves memory- and parameter-efficiency while maintaining or even improving performance on downstream tasks. Our approach utilizes NN pruning techniques to identify a subset of fine-tuning parameters and employs a matrix decomposition-based computation for efficient fine-tuning. This design enables the integration of ideas from model compression, SpFT, and matrix decomposition methods.

1.1. Our Contributions

At a high level, our contributions are as follows:

[1]Kahlert School of Computing, University of Utah, Salt Lake City, UT 84112, USA. Correspondence to: Cen-Jhih Li <cenjhih.li@gmail.com>, Aditya Bhaskara <bhaskaraaditya@gmail.com >.

Proceedings of the 3^{rd} Efficient Systems for Foundation Models (ES-FoMo) Workshop at the 41^{st} International Conference on Machine Learning (ICML), Vancouver, Canada. PMLR 267, 2025.

- We leverage ideas from *network pruning* to improve SpFT, achieving significant memory efficiency considerably lower than the popular LoRA. Our method uses only standard tensor operations, eliminating the need for custom sparse tensor libraries. Our approach supports fine-tuning quantized base models to further reduce the memory footprints.
- We analyze the memory assignment of several PEFT methods and suggest that *computation graphs can affect memory more significantly* than the number of trainable parameters. We validate our methods across diverse fine-tuning tasks and provide practical guidance on training strategies to maximize efficiency and accuracy.

The rest of the paper is organized as follows. We describe our approach in detail in Section 2 and analyze the advantage of memory footprint in Section 3. Section 4 describes the settings of our experiments. We then present and discuss our results in Section 5.

2. Our Method

To address the challenges mentioned in Section 1, we propose **S**tructured-**Pru**ning-based Sparse **F**ine-**T**uning (SPruFT), as illustrated in Figure 1. This is a row-based SpFT approach designed to streamline computation graphs. This method ensures memory efficiency while maintaining competitive performance.

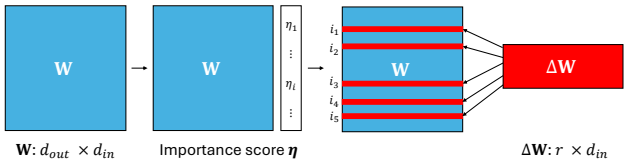

Figure 1. The illustration of SPruFT: we evaluate the importance score for each neuron to select the fine-tuning indices. Then we construct the lower-dimensional fine-tuning parameter matrix $\Delta\mathbf{W}$.

2.1. Proposed Method

SPruFT leverages structured neural network pruning to select a subset of parameters for fine-tuning, building upon the fine-tuning framework of SpFT (Guo et al., 2021; Sung et al., 2021; Ansell et al., 2022; Nikdan et al., 2024), which models fine-tuning as learning an additive weight matrix $\boldsymbol{\delta}$:

$$\hat{\mathbf{W}} = \mathbf{W} + \boldsymbol{\delta}, \tag{1}$$

$$\mathbf{h} = f(\hat{\mathbf{W}}, \mathbf{x}) = f(\mathbf{W} + \boldsymbol{\delta}, \mathbf{x}), \tag{2}$$

where $\mathbf{h} \in \mathbb{R}^{d_{out}}$ and $\mathbf{x} \in \mathbb{R}^{d_{in}}$ represent the output and input of the hidden layer, respectively.The function $f(\cdot)$ denotes the layer's forward pass. $\mathbf{W} \in \mathbb{R}^{d_{out} \times d_{in}}$ is the frozen pre-trained weight matrix, and $\hat{\mathbf{W}}$ is the effective weight matrix used at inference. In SpFT, $\boldsymbol{\delta}$ is typically unstructured, which often requires memory-saving techniques such as sparse formats (e.g., CSC/CSR (Mofrad et al., 2019; Lu et al., 2024) and semi-structured sparsity (Holmes et al., 2021)) and efficient computation (e.g., Sparse Matrix Multiplication (SpMM) and sparse backpropagation (Zhang et al., 2020; Gale et al., 2020; Peste et al., 2021; Schwarz et al., 2021; Hoefler et al., 2021; Jiang et al., 2022; Nikdan et al., 2023; Xu et al., 2024))[1]. However, these come with added complexity in implementation and runtime.

In contrast, SPruFT uses a structured $\boldsymbol{\delta}$, eliminating the need for sparse computation techniques. Specifically, we select the top-r most important neurons using an importance score vector $\boldsymbol{\eta}$, where r is determined by the target parameter budget. The selected neuron indices are denoted $i_1, i_2, \ldots, i_r$. Identifying task-relevant neurons has been widely studied in the pruning literature (LeCun et al., 1989; Han et al., 2015; Han, 2017; Hoefler et al., 2021; Liu et al., 2021; Fang et al., 2023; Ma et al., 2023; Frantar & Alistarh, 2023; Sun et al., 2024), typically for reducing model size while preserving accuracy. In this work, we use the ℓ_2 norm of each neuron's weight vector as the importance score, though we do not explore other metrics here. Notably, our definition of importance is task-specific for the fine-tuning objective, which may differ significantly from the original pretraining task.

After determining $\boldsymbol{\eta}$, we construct a smaller trainable matrix $\Delta\mathbf{W} \in \mathbb{R}^{r \times d_{in}}$ corresponding to the selected rows. Let $\mathbf{M} \in 0, 1^{d_{out} \times r}$ be a binary row selection matrix with $\mathbf{M}_{i_j j} = 1$ for $j \in [r]$ and zeros elsewhere. We adopt LoRA's computation graph to enable efficient gradient computation[2]:

$$\begin{aligned} f(\hat{\mathbf{W}}, \mathbf{x}) &= f(\mathbf{W} + \mathbf{M}\Delta\mathbf{W}, \mathbf{x}) \\ &= f(\mathbf{W}, \mathbf{x}) + f(\mathbf{M}\Delta\mathbf{W}, \mathbf{x}), \end{aligned} \tag{3}$$

For comparison, LoRA approximates $\boldsymbol{\delta}$ as the product of two low-rank matrices $\mathbf{B} \in \mathbb{R}^{d_{out} \times r}$ and $\mathbf{A} \in \mathbb{R}^{r \times d_{in}}$:

$$\hat{\mathbf{W}} = \mathbf{W} + \frac{\alpha}{r}\mathbf{B}\mathbf{A}, \tag{4}$$

$$\mathbf{h} = f(\hat{\mathbf{W}}, \mathbf{x}) = f(\mathbf{W}, \mathbf{x}) + f\left(\frac{\alpha}{r}\mathbf{B}\mathbf{A}, \mathbf{x}\right), \tag{5}$$

where α is a scaling hyperparameter.

In both SPruFT and LoRA, due to the additive structure of transformer components (e.g., self-attention and feed

[1]See also NVIDIA's memory optimization techniques: https://pytorch.org/torchtune/stable/tutorials/memory_optimizations.html

[2]Directly updating selected rows of $\mathbf{W}$ would be more efficient, but learning $\Delta\mathbf{W}$ separately provides better task modularity. During inference, $\Delta\mathbf{W}$ can be merged into $\mathbf{W}$, preserving runtime efficiency.

Model, ft setting	mem	#param	BoolQ	PIQA	SIQA	HS	WG	ARC-c	ARC-e	OBQA	Avg	GSM8k
Llama2(7B), pretrained	-	-	58.00	40.00	29.00	15.40	4.80	14.00	16.20	25.40	25.35	0.00
LoRA, $r = 64$	23.46GB	159.9M(2.37%)	77.00	**76.20**	67.80	84.20	62.60	70.00	82.00	**74.00**	74.23	18.42
VeRA, $r = 64$	22.97GB	1.374M(0.02%)	47.80	51.80	41.80	37.60	50.40	36.80	43.20	32.80	41.53	0.00
DoRA, $r = 64$	44.85GB	161.3M(2.39%)	75.20	75.40	64.60	78.60	63.00	65.20	82.20	70.60	71.85	21.46
RoSA, $r = 32, d = 1.2\%$	44.69GB	157.7M(2.34%)	79.80	73.40	**70.20**	76.00	57.00	68.80	80.80	71.60	72.20	21.99
SPruFT, $r = 128$	**17.62GB**	145.8M(2.16%)	**80.00**	75.20	67.60	**85.00**	**63.40**	**70.80**	**82.40**	71.80	**74.53**	**22.90**
Llama3(8B), pretrained	-	-	58.80	41.60	38.00	10.20	11.20	55.20	63.00	27.40	38.18	0.00
LoRA, $r = 64$	30.37GB	167.8M(2.09%)	84.20	77.00	63.20	84.20	67.20	76.40	88.80	71.00	76.50	41.77
VeRA, $r = 64$	29.49GB	1.391M(0.02%)	61.00	62.40	55.60	41.80	49.60	59.60	77.60	60.00	58.45	0.00
DoRA, $r = 64$	51.45GB	169.1M(2.11%)	83.20	**82.80**	69.00	**89.40**	**70.80**	77.20	89.00	80.40	80.23	46.02
RoSA, $r = 32, d = 1.2\%$	48.40GB	167.6M(2.09%)	79.00	81.00	69.20	84.80	68.60	79.00	90.40	78.40	78.80	45.72
SPruFT, $r = 128$	**24.49GB**	159.4M(1.98%)	**87.60**	77.40	**71.40**	85.40	70.20	**79.80**	**90.80**	**81.80**	**80.55**	**46.10**

Table 1. Main results of fine-tuning full precision Llama2 and Llama3. "mem" represents the memory cost in training excluding the model itself, with further details provided in Appendix C.3. #param is the number of trainable parameters. HS, OBQA, and WG represent HellaSwag, OpenBookQA, and WinoGrande datasets. All reported results for commonsense reasoning tasks are accuracies, while the results for GSM8k are the exact match score. **Bold** indicates the best result on the same task.

forwarding network (FFN), see (Vaswani, 2017)), gradients can be computed efficiently while keeping $\mathbf{W}$ frozen.

Due to LoRA's simplicity and effectiveness, numerous variants have been proposed to enhance the performance, e.g., QLoRA (Dettmers et al., 2022; Guo et al., 2024a; Li et al., 2024; Dettmers et al., 2024), DoRA (Liu et al., 2024), RoSA (Nikdan et al., 2024), and VeRA (Kopiczko et al., 2024). These methods have achieved exceptional performance, often comparable to full fine-tuning across a range of tasks.

2.2. QSPruFT: Extension Approach with Low Quantization Error

Our approach is model-agnostic, allowing users to integrate recent advances in PEFT such as QLoRA (Dettmers et al., 2024), LoftQ (Li et al., 2024), and QPiSSA (Meng et al., 2024). QLoRA quantizes the base model to Normal Float 4-bit (NF4) and fine-tunes full-precision LoRA matrices $\mathbf{A}$ and $\mathbf{B}$. Based on QLoRA, LoftQ and QPiSSA propose alternative initialization strategies for $\mathbf{A}$ and $\mathbf{B}$ to reduce quantization error. This error is defined as:

$$\mathbf{W}^{res} = \mathbf{W} - \mathbf{W}^{\text{NF4}}, \tag{6}$$

and both LoftQ and QPiSSA use singular value decomposition (SVD) to initialize $\mathbf{A}$ and $\mathbf{B}$ such that $\mathbf{B}^{init}\mathbf{A}^{init} \approx \mathbf{W}^{res}$.

In our method, when applied to quantized base models, we do not require decomposition to approximate $\mathbf{W}^{res}$. Since we fine-tune only selected rows, we can directly initialize $\Delta\mathbf{W}$ using the corresponding rows from $\mathbf{W}^{res}$, where *the quantization error of the fine-tuning rows is **zero***. This offers a potential advantage over QLoRA, QDoRA, LoftQ, and QPiSSA in terms of accuracy.

3. Advantage of Our Approach in Memory Footprints

In this section, we want to emphasize that *in PEFT research, reducing the number of trainable parameters is not the most critical factor for minimizing memory consumption.* While certain PEFT methods explicitly aim to lower the number of trainable parameters to reduce memory usage, the impact of this reduction diminishes once the parameter count is sufficiently small. To investigate this further, we compare several representative methods of SpFT and low-rank methods, focusing on their memory footprints during training, as shown in Figure 2. For precisely, we assume full-precision training to exclude memory costs introduced by quantization or other compression techniques and implementations. Notably, quantizing the model to 4-bit precision can yield an additional memory savings of approximately 75%.

During neural network training, backpropagation requires caching a large number of intermediate activations to compute gradients efficiently. The memory cost of these intermediate values is largely influenced by the structure of the computation graph. When the number of trainable parameters is small, the memory consumed by intermediate activations (see green bars in Figure 2) often dominates memory usage apart from the model weights.

Among the LoRA-based methods, VeRA attempts to reduce memory by sharing a pair of low-rank matrices across layers, thereby reducing the number of trainable parameters. However, as shown in Figure 2, this results in only marginal memory savings—around 0.5GB to 2GB depending on the maximum token length, which is almost negligible. In contrast, DoRA and RoSA incur significantly higher memory usage due to their more complex computation graphs and reliance on unstructured sparse matrices. For instance, DoRA decomposes LoRA's matrices into separate magnitude and direction components (see Figure 4 in Appendix C.4), which substantially increases memory requirements for activation

Model, ft setting	BoolQ	PIQA	SIQA	HS	WG	ARC-c	ARC-e	OBQA	Avg	GSM8k
Llama2(7B), pretrained	58.00	40.00	29.00	15.40	4.80	14.00	16.20	25.40	25.35	0.00
QLoRA, $r = 64$	75.20	74.80	66.80	73.80	63.60	63.40	77.60	65.40	70.08	18.88
QDoRA, $r = 64$	77.40	72.00	68.40	81.60	63.20	64.60	80.20	70.60	72.25	**25.63**
LoftQ, $r = 64$	73.80	**73.20**	**72.60**	81.60	64.20	**67.60**	**82.00**	71.60	73.33	20.55
QPiSSA, $r = 64$	79.40	72.40	68.60	77.40	65.00	67.40	78.60	69.80	72.33	21.08
QSPruFT*, $r = 128$	78.00	72.60	69.00	79.40	63.20	67.20	79.60	**74.80**	72.98	20.32
QSPruFT†, $r = 128$	**82.00**	71.80	69.20	**83.20**	**65.40**	67.20	79.20	72.40	**73.80**	22.14
Llama3(8B), pretrained	58.80	41.60	38.00	10.20	11.20	55.20	63.00	27.40	38.18	0.00
QLoRA, $r = 64$	77.40	81.60	**72.80**	87.60	69.60	75.80	90.20	78.60	79.20	41.17
QDoRA, $r = 64$	79.00	79.20	68.20	82.20	66.40	75.80	87.40	77.20	76.93	46.17
LoftQ, $r = 64$	87.00	**82.80**	69.40	**90.20**	62.40	74.80	91.00	**79.60**	79.65	44.96
QPiSSA, $r = 64$	87.60	82.00	71.00	82.60	**71.80**	**77.20**	89.60	77.20	79.88	45.87
QSPruFT*, $r = 128$	89.80	79.40	65.40	89.80	68.40	76.20	**91.80**	78.00	79.85	47.76
QSPruFT†, $r = 128$	**90.40**	82.40	68.60	89.40	69.20	76.80	90.80	76.40	**80.50**	**49.13**

Table 2. Main results of fine-tuning NF4-quantized LLaMA-2 and LLaMA-3. QSPruFT* denotes our approach with $\Delta\mathbf{W}$ initialized randomly, while QSPruFT† uses $\Delta\mathbf{W}$ initialized with the corresponding rows from the quantization residual $\mathbf{W}^{res}$.

caching. While DoRA's trainable parameter cost is similar to that of LoRA, its overall memory consumption is considerably higher. RoSA also consumes much more memory than LoRA despite incorporating efficiency-oriented design choices. These findings suggest that a simple computation graph can be a far more significant contributor to memory usage than reducing trainable parameters.

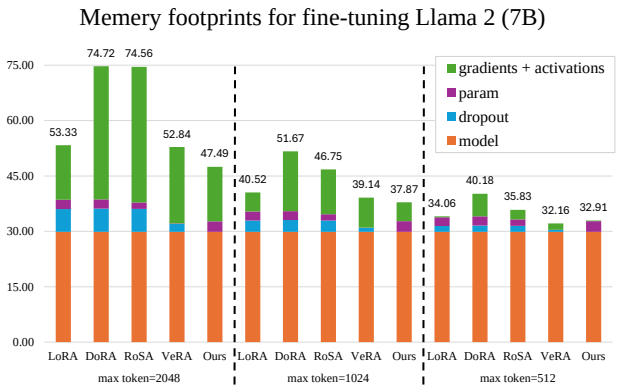

Figure 2. Memory footprints of fine-tuning full precision Llama2 using LoRA (Hu et al., 2022), RoSA (Nikdan et al., 2024), DoRA (Liu et al., 2024), VeRA (Kopiczko et al., 2024), and ours. We set $r = 32$, $d = 1.2\%$ for RoSA, $r = 128$ for ours, and $r = 64$ for the others. Note that RoSA has its own official implementation, which may influence memory consumption, whereas LoRA, DoRA, and VeRA are integrated into the PEFT library provided by Hugging Face. More details please see Appendix C.3.

4. Experimental Setup

We fine-tune Llama-2-7B and Llama-3-8B in full-precision (float32) and NF4 (Dettmers et al., 2024) on the training split of 8 commensense reasoning tasks and evaluate the result on the test splits. Then we fine-tune the LLMs on the training split of GSM8k (Cobbe et al., 2021) and evaluate the performance on the test split via EleutherAI LM Harness tasks (Gao et al., 2021). See Appendix D for details.

We fine-tune the models using our SPruFT, QSPruFT, LoRA (Hu et al., 2022), QLoRA (Dettmers et al., 2024), VeRA (Kopiczko et al., 2024), DoRA (Liu et al., 2024), QDoRA, RoSA (Nikdan et al., 2024), LoftQ (Li et al., 2024), and QPiSSA (Meng et al., 2024). RoSA is chosen as the representative SpFT method and is the only SpFT due to the high memory demands of other SpFT approaches, while full fine-tuning is excluded for the same reason. We freeze Llama's classification layers and fine-tune only the linear layers in attention and FFN blocks.

We use a learning rate of $2 \cdot 10^{-5}$ with linear decay (rate 0.01) for our method, and 10^{-4} for other PEFT methods, with $\alpha = 16$ and dropout rate 0.1. All methods apply linear decay after a 3% warmup. For commonsense reasoning, we train on 2048 samples (256 per dataset) and evaluate on 500 test examples per dataset. For GSM8K, we fine-tune on 2048 random training samples and evaluate on the full test set. Instruction-style prompts are used for commonsense datasets,[3] while GSM8K uses question-answering prompts. We fine-tune all models for 3 epochs.

Our framework is built on torch-pruning (Fang et al., 2023), PyTorch (Paszke et al., 2019), and HuggingFace Transformers (Wolf et al., 2020). Most experiments are conducted on a single A100-80GB GPU, except DoRA and RoSA (at 2048 max tokens) which run on an H100-96GB. We use the Adam optimizer (Kingma & Ba, 2015) and train with a fixed epoch budget without early stopping.

5. Results and Discussion

5.1. Main Results

We now present the results of fine-tuning LLaMA models using our SPruFT framework and LoRA's variants. As

[3]LLaMA-3 performs well with question-answering prompts, and fine-tuning yields limited gains, suggesting possible pre-training on these datasets with question-answering prompts.

shown in Table 1, our method achieves superior memory efficiency while maintaining strong accuracy. Notably, although VeRA uses significantly fewer trainable parameters, it suffers from a marked drop in accuracy. In addition, Table 2 demonstrates that our method remains effective when fine-tuning NF4-quantized base models. Additionally, the results show that leveraging initialization strategies to reduce quantization error can further enhance the performance of our approach.

5.2. Conclusions and Future Work

We propose a parameter-efficient fine-tuning (PEFT) framework that integrates various techniques and importance metrics from model compression research to enhance sparse fine-tuning (SpFT). Using our method, we can fine-tune LLMs using significantly less computation resources than the popular LoRA (Low-Rank Adaptation) technique, while achieving similar accuracy. There are several future directions: (1) For importance metrics, we may wish to explore better metrics for LLMs. (2) Our results show that fine-tuning a small number of neurons can significantly improve model performance on downstream tasks. This observation naturally raises the question: do the selected neurons play a distinctive role in specific tasks? This question is related to the explainability of neural networks, which is an extensive area of research. It will be interesting to understand if (and how) individual neurons chosen for fine-tuning contribute to the new task.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Ansell, A., Ponti, E., Korhonen, A., and Vulić, I. Composable sparse fine-tuning for cross-lingual transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1778–1796, 2022.

Bar-Haim, R., Dagan, I., Dolan, B., Ferro, L., Giampiccolo, D., Magnini, B., and Szpektor, I. The second PASCAL recognising textual entailment challenge. 2006.

Bentivogli, L., Dagan, I., Dang, H. T., Giampiccolo, D., and Magnini, B. The fifth PASCAL recognizing textual entailment challenge. 2009.

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I., and Specia, L. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In Bethard, S., Carpuat, M., Apidianaki, M., Mohammad, S. M., Cer, D., and Jurgens, D. (eds.), *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pp. 1–14, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/S17-2001. URL https://aclanthology.org/S17-2001.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, 2019.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dagan, I., Glickman, O., and Magnini, B. The PASCAL recognising textual entailment challenge. In *Machine learning challenges. evaluating predictive uncertainty, visual object classification, and recognising tectual entailment*, pp. 177–190. Springer, 2006.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332, 2022.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.

Dolan, W. B. and Brockett, C. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the International Workshop on Paraphrasing*, 2005.

Dosovitskiy, A. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Fang, G., Ma, X., Song, M., Mi, M. B., and Wang, X. Depgraph: Towards any structural pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 16091–16101, 2023.

Frantar, E. and Alistarh, D. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023.

Gale, T., Zaharia, M., Young, C., and Elsen, E. Sparse gpu kernels for deep learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14. IEEE, 2020.

Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., et al. A framework for few-shot language model evaluation. *Version v0. 0.1. Sept*, 10:8–9, 2021.

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., and Keutzer, K. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pp. 291–326. Chapman and Hall/CRC, 2022.

Giampiccolo, D., Magnini, B., Dagan, I., and Dolan, B. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pp. 1–9. Association for Computational Linguistics, 2007.

Guo, D., Rush, A., and Kim, Y. Parameter-efficient transfer learning with diff pruning. In *Annual Meeting of the Association for Computational Linguistics*, 2021.

Guo, H., Greengard, P., Xing, E., and Kim, Y. LQ-loRA: Low-rank plus quantized matrix decomposition for efficient language model finetuning. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=xw29VvOMmU.

Guo, W., Long, J., Zeng, Y., Liu, Z., Yang, X., Ran, Y., Gardner, J. R., Bastani, O., De Sa, C., Yu, X., et al. Zeroth-order fine-tuning of llms with extreme sparsity. *arXiv preprint arXiv:2406.02913*, 2024b.

Han, S. *Efficient methods and hardware for deep learning*. PhD thesis, Stanford University, 2017.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, P., Gao, J., and Chen, W. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing. In *The Eleventh International Conference on Learning Representations*, 2023.

Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., and Peste, A. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021.

Holmes, C., Zhang, M., He, Y., and Wu, B. Nxmtransformer: semi-structured sparsification for natural language understanding via admm. *Advances in neural information processing systems*, 34:1818–1830, 2021.

Hu, E. J., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Jiang, P., Hu, L., and Song, S. Exposing and exploiting fine-grained block structures for fast and accurate sparse training. *Advances in Neural Information Processing Systems*, 35:38345–38357, 2022.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.

Kopiczko, D. J., Blankevoort, T., and Asano, Y. M. VeRA: Vector-based random matrix adaptation. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NjNfLdxr3A.

Krizhevsky, A. et al. Learning multiple layers of features from tiny images. 2009.

LeCun, Y., Denker, J., and Solla, S. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

Li, F.-F., Andreeto, M., Ranzato, M., and Perona, P. Caltech 101, Apr 2022.

Li, Y., Yu, Y., Liang, C., Karampatziakis, N., He, P., Chen, W., and Zhao, T. Loftq: LoRA-fine-tuning-aware quantization for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=LzPWWPAdY4.

Lialin, V., Deshpande, V., and Rumshisky, A. Scaling down to scale up: A guide to parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.15647*, 2023.

Liu, L., Zhang, S., Kuang, Z., Zhou, A., Xue, J.-H., Wang, X., Chen, Y., Yang, W., Liao, Q., and Zhang, W. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, pp. 7021–7032. PMLR, 2021.

Liu, S.-Y., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., Cheng, K.-T., and Chen, M.-H. DoRA: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=3d5CIRG1n2.

Loshchilov, I. and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

Lu, X., Zhou, A., Xu, Y., Zhang, R., Gao, P., and Li, H. SPP: Sparsity-preserved parameter-efficient fine-tuning for large language models. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=9Rroj9GIOQ.

Ma, X., Fang, G., and Wang, X. LLM-pruner: On the structural pruning of large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=J8Ajf9WfXP.

Malladi, S., Gao, T., Nichani, E., Damian, A., Lee, J. D., Chen, D., and Arora, S. Fine-tuning language models with just forward passes. *Advances in Neural Information Processing Systems*, 36:53038–53075, 2023.

Meng, F., Wang, Z., and Zhang, M. Pissa: Principal singular values and singular vectors adaptation of large language models. In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 121038–121072. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/db36f4d603cc9e3a2a5e10b93e6428f2-Paper-Conference.pdf.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., and Wu, H. Mixed precision training. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=r1gs9JgRZ.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. 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*, pp. 2381–2391, 2018.

Mofrad, M. H., Melhem, R., Ahmad, Y., and Hammoud, M. Multithreaded layer-wise training of sparse deep neural networks using compressed sparse column. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–6. IEEE, 2019.

Nikdan, M., Pegolotti, T., Iofinova, E., Kurtic, E., and Alistarh, D. SparseProp: Efficient sparse backpropagation for faster training of neural networks at the edge. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 26215–26227. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/nikdan23a.html.

Nikdan, M., Tabesh, S., Crnčević, E., and Alistarh, D. RoSA: Accurate parameter-efficient fine-tuning via robust adaptation. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=FYvpxyS43U.

Panigrahi, A., Saunshi, N., Zhao, H., and Arora, S. Task-specific skill localization in fine-tuned language models. In *International Conference on Machine Learning*, pp. 27011–27033. PMLR, 2023.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Peste, A., Iofinova, E., Vladu, A., and Alistarh, D. Ac/dc: Alternating compressed/decompressed training of deep neural networks. *Advances in neural information processing systems*, 34:8557–8570, 2021.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of EMNLP*, pp. 2383–2392. Association for Computational Linguistics, 2016.

Rissanen, J. J. Fisher information and stochastic complexity. *IEEE transactions on information theory*, 42(1):40–47, 1996.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Sap, M., Rashkin, H., Chen, D., Le Bras, R., and Choi, Y. Social IQa: Commonsense reasoning about social interactions. In Inui, K., Jiang, J., Ng, V., and Wan, X. (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4463–4473, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1454. URL https://aclanthology.org/D19-1454/.

Schwarz, J., Jayakumar, S., Pascanu, R., Latham, P. E., and Teh, Y. Powerpropagation: A sparsity inducing weight reparameterisation. *Advances in neural information processing systems*, 34:28889–28903, 2021.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*, pp. 1631–1642, 2013.

Sun, M., Liu, Z., Bair, A., and Kolter, J. Z. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

Sung, Y.-L., Nair, V., and Raffel, C. A. Training neural networks with fixed sparse masks. *Advances in Neural Information Processing Systems*, 34:24193–24205, 2021.

Tavanaei, A. Embedded encoder-decoder in convolutional networks towards explainable ai. *arXiv preprint arXiv:2007.06712*, 2020.

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jégou, H. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pp. 10347–10357. PMLR, 2021.

Vaswani, A. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Warstadt, A., Singh, A., and Bowman, S. R. Neural network acceptability judgments. *arXiv preprint 1805.12471*, 2018.

Williams, A., Nangia, N., and Bowman, S. R. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of NAACL-HLT*, 2018.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.

Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

Xu, M., Yin, W., Cai, D., Yi, R., Xu, D., Wang, Q., Wu, B., Zhao, Y., Yang, C., Wang, S., et al. A survey of resource-efficient llm and multimodal foundation models. *arXiv preprint arXiv:2401.08092*, 2024.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019.

Zhang, Z., Wang, H., Han, S., and Dally, W. J. Sparch: Efficient architecture for sparse matrix multiplication. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 261–274. IEEE, 2020.

Zhou, A., Wang, K., Lu, Z., Shi, W., Luo, S., Qin, Z., Lu, S., Jia, A., Song, L., Zhan, M., et al. Solving challenging math word problems using gpt-4 code interpreter with code-based self-verification. In *The Twelfth International Conference on Learning Representations*, 2024.

A. Importance Metrics

Taylor importance is the Taylor expansion of the difference between the loss of the model with and without the target neuron:

$$\begin{aligned}\boldsymbol{\eta}_i &= L(\mathcal{D}, \acute{F}_{\boldsymbol{c}_i}) - L(\mathcal{D}, F) \\ &\approx -\mathbf{w}^\top \nabla_{\mathbf{w}} L(\mathcal{D}, F) + \frac{1}{2}\mathbf{w}^\top \nabla^2_{\mathbf{w}} L(\mathcal{D}, F)\mathbf{w} + \mathcal{O}(\nabla^3_{\mathbf{w}} L(\mathcal{D}, F)) \\ &\overset{(*)}{\approx} \frac{1}{2}\mathbf{w}^\top \nabla^2_{\mathbf{w}} L(\mathcal{D}, F)\mathbf{w} + \mathcal{O}(\nabla^3_{\mathbf{w}} L(\mathcal{D}, F)) \\ &\overset{(**)}{\approx} \frac{1}{2}(G\mathbf{w})^\top (G\mathbf{w}) + \mathcal{O}(\nabla^3_{\mathbf{w}} L(\mathcal{D}, F)),\end{aligned}$$

where $G = \nabla_{\mathbf{w}} L(\mathcal{D}, F)$. (**) is from the result of Fisher information (Rissanen, 1996):

$$\nabla^2_{\mathbf{w}} L(\mathcal{D}, F) \approx \nabla_{\mathbf{w}} L(\mathcal{D}, F)^\top \nabla_{\mathbf{w}} L(\mathcal{D}, F).$$

Note that (*) is from $\nabla_{\mathbf{w}} L(\mathcal{D}, F) \approx 0$, as removing one channel/neuron from a large neural network typically results in only a negligible reduction in loss. To efficiently compute $\boldsymbol{\eta}_i$, the equation can be further derived as:

$$\boldsymbol{\eta}_i \approx (G\mathbf{w})^\top (G\mathbf{w}) = \sum_j (\frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}} \frac{\partial L(\mathbf{x}, F)}{\partial w_j} w_j)^2 \approx \sum_j |\frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}} \frac{\partial L(\mathbf{x}, F)}{\partial w_j} w_j|,$$

where $\mathbf{w} = (w_1, \ldots, w_j, \ldots)$.

Magnitude importance is the ℓ_2-norm of the neuron vector computed as $\sqrt{\sum_j w_j^2}$.

B. Parameter Dependency

Dependencies of parameters between neurons or channels across different layers exist in NNs. These include basic layer connections, residual connections, tensor concatenations, summations, and more, as shown in Figure 3. The black neurons connected by real lines represent the dependent parameters that are in the same group. Pruning any black neurons results in removing the parameters connected by the real lines. (Liu et al., 2021) introduced a group pruning method for CNN models that treats residual connections as grouped dependencies, evaluating and pruning related channels within the same group simultaneously. Similarly, (Fang et al., 2023) proposed a novel group pruning technique named Torch-Pruning, which considers various types of dependencies and achieves state-of-the-art results. (Ma et al., 2023) further applied this procedure to pruning LLMs. Torch-Pruning can be applied to prune a wide range of neural networks, including image transformers, LLMs, CNNs, and more, making it a popular toolkit for neural network pruning.

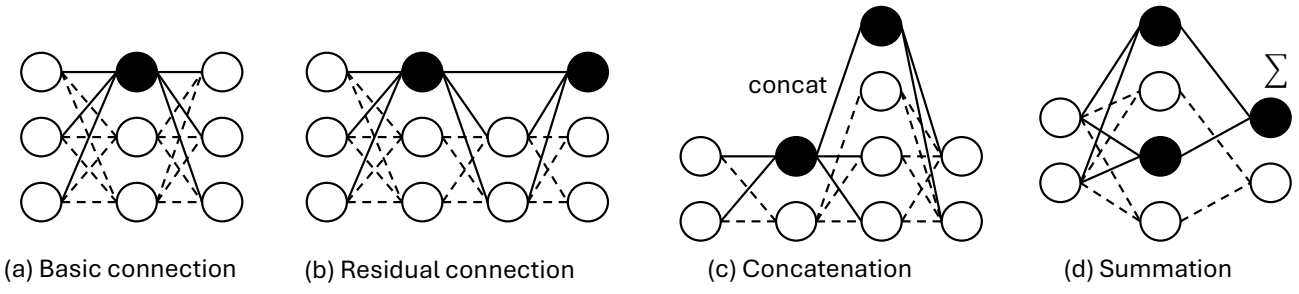

Figure 3. Common dependencies of parameters in neural networks.

In this study, we also evaluate the influences of incorporating parameter dependency in our approach. We put the experimental results of whether incorporating parameter dependency in Appendix C.2. In the experiments, parameter dependency becomes the following process for our approach: first, searching for dependencies by tracing the computation graph of gradient;

next, evaluating the importance of parameter groups; and finally, fine-tuning the parameters within those important groups collectively. For instance, if $\mathbf{W}^a_{:j}$ and $\mathbf{W}^b_{i:}$ are dependent, where $\mathbf{W}^a_{:j}$ is the j-th column in parameter matrix (or the j-th input channels/features) of layer a and $\mathbf{W}^b_{i:}$ is the i-th row in parameter matrix (or the i-th output channels/features) of layer b, then $\mathbf{W}^a_{:j}$ and $\mathbf{W}^b_{i:}$ will be fine-tuned simultaneously while the corresponding $\mathbf{M}^a_{dep}$ for $\mathbf{W}^a_{:j}$ becomes column selection matrix and $\mathbf{W}^a_s$ becomes $\mathbf{W}^a_{f,dep}\mathbf{M}^a_{dep}$. Consequently, fine-tuning 2.5% output channels for layer b will result in fine-tuning additional 2.5% input channels in each dependent layer. Therefore, for the 5% of desired fine-tuning ratio, the fine-tuning ratio with considering dependencies is set to 2.5%[4] for the approach that includes dependencies.

The forward function of layer a for column selection mentioned above can be written as the following equation:

$$f(\hat{\mathbf{W}}^a, \mathbf{x}) = f(\mathbf{W}^a, \mathbf{x}) + f(\mathbf{M}^a\mathbf{W}^a_f, \mathbf{x}) + f(\mathbf{W}^a_{f,dep}\mathbf{M}^a_{dep}, \mathbf{x}).$$

Note that in this example, as the dependency is connection between the output feature/channel of b and the input feature/channel of a, the dimension d^a_{in} is equal to d^b_{out} where $\mathbf{W}^a \in \mathbb{R}^{d^a_{out} \times d^a_{in}}$, $\mathbf{W}^b \in \mathbb{R}^{d^b_{out} \times d^b_{in}}$.

C. Ablation Studies and Related Analysis

In this section, we first discuss the hyperparameter settings. While we do not include DeBERTaV3 (He et al., 2023), DeiT (Touvron et al., 2021), ViT (Dosovitskiy, 2020), ResNet101 (He et al., 2016), and ResNeXt101 (Xie et al., 2017) in the main context, we fine-tune DeBERTaV3-base (He et al., 2023) on GLUE and fine-tune DeiT, ViT, ResNet101, and ResNeXt101 on Tiny-ImageNet (Tavanaei, 2020), CIFAR100 (Krizhevsky et al., 2009), and Caltech101 (Li et al., 2022). For these tasks, we set the fine-tuning ratio at 5%, meaning the trainable parameters are a total of 5% of the backbone plus classification layers. Following this, we discuss the computational resource requirements for fine-tuning. Figure 5 and Figure 4 illustrate the computation and cache requirements during backpropagation for our method, LoRA, and DoRA.

For DeBERTaV3, the learning rate is set to $2 \cdot 10^{-5}$ with linear decay, where the decay rate is 0.01. The model is fine-tuned on the full training split of 8 tasks from the GLUE benchmark. The maximum sequence length is fixed to 256 with longer sequences truncated and shorter sequences padded.

For image models, the learning rate is set to 10^{-4} with cosine annealing decay (Loshchilov & Hutter, 2017), where the minimum learning rate is 10^{-9}. All image models used in this study have been pre-trained on ImageNet. Note that memory efficiency is not emphasized for small-scale models, as dataset-related memory—particularly with large batch sizes—dominates consumption in these cases. The main advantage of our method in these cases is the reduced FLOPs due to fewer trainable parameters.

C.1. Hyperparameter Settings

We report the results of three approaches over several epochs as table 3 and table 4. Overall, full fine-tuning over higher epochs is more prone to overfitting, while head fine-tuning shows the exact opposite trend. Except for the results on caltech101[5], the loss patterns across all models consistently reflect this trend, and most accuracy results further support this conclusion. However, our approach demonstrates a crucial advantage by effectively balancing the tradeoff between performance and computational resources.

Table 3 clearly shows that both our approach and full fine-tuning achieve optimal results within a few epochs, while head fine-tuning requires more training. Notably, all models have been pre-trained on ImageNet-1k, which may explain the strong performance observed with head fine-tuning on Tiny-ImageNet. However, even with this advantage, full fine-tuning still outperforms head fine-tuning, and our approach surpasses both. In just 5 epochs, our approach achieves results comparable to full fine-tuning on all datasets with significantly lower trainable parameters.

In contrast to Table 3, the results in Table 4 show more variation. Although the validation loss follows a similar trend, we report only the evaluation metrics due to the different patterns observed in these metrics. One potential reason for this variation is the varying amounts of training data across the GLUE tasks. As shown in the table, tasks with fewer samples often require more epochs to achieve better performance for both full fine-tuning and our approach. Conversely, for tasks with large amounts of training data such as 'MNLI', 'QNLI', 'QQP', and 'SST-2', the results show tiny improvement from

[4] In some complex models, considering dependencies results in slightly more than twice the number of trainable parameters. However, in most cases, the factor is 2.

[5] The inconsistent trend observed in Caltech101 results is likely due to its significantly smaller sample size.

	CIFAR100			Tiny-ImageNet			Caltech101		
	Full	Head	SPruFT	Full	Head	SPruFT	Full	Head	SPruFT
#ep	loss, acc	loss, acc	loss, acc	loss, acc	loss, acc	loss, acc	loss, acc	loss, acc	loss, acc
	DeiT			DeiT			DeiT		
#param:	86.0M	0.2M	4.6M	86.1M	0.3M	4.8M	86.0M	0.2M	4.6M
5	**0.36**, **90.18**	0.76, 80.25	**0.37**, **88.70**	**0.54**, **87.55**	0.60, 85.09	**0.40**, **89.69**	0.11, 97.33	1.09, 89.02	0.30, 95.41
10	0.44, 90.04	0.64, 81.83	0.42, 88.62	0.69, 86.32	0.54, 85.72	0.49, 88.96	**0.11**, **97.55**	0.53, 93.22	0.17, 96.28
30	0.62, 89.03	**0.55**, **83.42**	0.64, 88.61	0.94, 84.27	**0.52**, **86.06**	0.72, 88.67	0.11, 97.11	**0.22**, **95.06**	**0.12**, **96.50**
	ViT			ViT			ViT		
#param:	85.9M	0.1M	4.5M	86.0M	0.2M	4.6M	85.9M	0.1M	45.2M
5	**0.38**, **90.13**	1.01, 74.78	**0.40**, **88.13**	**0.51**, **88.45**	0.65, 84.10	**0.36**, **90.87**	0.12, 97.16	1.60, 85.70	0.43, 93.96
10	0.45, 89.85	0.85, 77.05	0.45, 87.55	0.66, 86.78	0.58, 84.95	0.44, 90.48	**0.11**, 97.20	0.85, 89.98	0.23, 95.54
30	0.62, 88.78	**0.71**, **79.51**	0.69, 87.83	0.96, 84.20	**0.55**, **85.49**	0.61, 90.56	0.12, **97.24**	**0.33**, **92.65**	**0.16**, **96.02**
	ResNet101			ResNet101			ResNet101		
#param:	42.7M	0.2M	2.2M	42.9M	0.4M	2.4M	42.7M	0.2M	2.2M
5	**0.50**, 86.21	1.62, 60.78	**0.59**, 82.36	**0.92**, **77.78**	1.64, 62.06	**0.76**, **79.66**	0.14, 96.50	1.25, 82.33	0.48, 92.56
10	0.58, **86.41**	1.39, 63.06	0.60, 82.33	1.10, 76.81	1.50, 63.19	0.79, 79.54	**0.14**, **96.54**	0.69, 90.24	0.23, 95.58
30	0.80, 84.72	**1.21**, **65.63**	0.80, **82.49**	1.54, 74.09	**1.43**, **64.47**	1.08, 78.58	0.18, 95.80	**0.31**, **93.00**	**0.16**, **95.89**
	ResNeXt101			ResNeXt101			ResNeXt101		
#param:	87.0M	0.2M	4.9M	87.2M	0.4M	5.1M	87.0M	0.2M	4.9M
5	**0.47**, **87.30**	1.42, 65.07	**0.47**, 85.94	**0.86**, **79.51**	1.46, 65.59	**0.61**, **83.88**	**0.12**, **97.07**	1.25, 83.16	0.28, 95.84
10	0.56, 87.17	1.23, 67.55	0.53, 86.04	1.01, 79.27	1.35, 66.73	0.69, 83.47	0.13, 96.89	0.68, 90.94	0.18, 96.28
30	0.71, 86.59	**1.08**, **69.45**	0.69, **86.33**	1.41, 76.55	**1.29**, **67.93**	0.90, 82.83	0.16, 96.63	**0.31**, **92.87**	**0.14**, **96.76**

Table 3. Fine-tuning on CIFAR100 and Tiny-ImageNet. #ep and #param represent the number of epochs and the number of trainable parameters, where SPruFT is our method with Taylor importance. Full and Head indicate full fine-tuning and head-finetuning, which only fine-tunes the classification layer. All reported losses and accuracies are based on validation results. **Bold** denotes the best results of each fine-tuning approach (in the same column) on the same model and dataset.

		task	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B
		#train	8.5k	393k	3.7k	108k	364k	2.5k	67k	7k
method	#param	epochs	mcc	acc	acc	acc	acc	acc	acc	corr
Full	184.42M	3	**69.96**	**89.42**	89.71	**93.57**	**92.08**	80.14	**95.53**	90.44
Full		5	69.48	89.29	87.74	93.36	92.08	**83.39**	94.72	90.14
Full		10	68.98	88.55	**90.20**	93.15	91.97	80.51	93.81	**90.71**
Head	592.13K	3	24.04	62.64	**68.38**	70.73	80.18	**52.71**	65.48	5.66
Head		5	45.39	61.75	**68.38**	**72.32**	80.59	47.29	**78.44**	26.88
Head		10	**47.32**	**63.98**	**68.38**	71.99	**80.96**	47.29	74.66	**49.59**
SPruFT	103.57M	3	64.08	89.58	81.62	93.10	90.70	70.40	95.18	86.58
SPruFT		5	65.40	**90.21**	86.03	**93.17**	90.93	74.37	95.30	87.36
SPruFT		10	**65.56**	89.55	**87.50**	93.15	**91.57**	**80.14**	**95.41**	**89.14**

Table 4. Fine-tuning DeBERTaV3 on GLUE. 'mcc', 'acc', and 'corr' represent 'Matthews correlation', 'accuracy', and 'Pearson correlation', respectively. #param is the number of trainable parameters. SPruFT is our method with Taylor importance, while Full and Head indicate full fine-tuning and head-finetuning, which only fine-tunes the classification layer. All reported metrics are based on validation results, and are percentages. **Bold** denotes the best results of each fine-tuning approach on the same task.

3 to 10 epochs. Nevertheless, the results still demonstrate that our approach significantly balances the tradeoff between performance and computational resources. Our method achieves near full fine-tuning performance with remarkably less trainable parameters.

We refer from Table 4 for epochs choosing of fine-tuning Llama2 and Llama3. Table 4 shows that 5 or 10 epochs are reasonable for most tasks using our approach. Given that the maximum sequence lengths of Llama are longer than DeBERTaV3, we have opted for only 5 epochs in the main experiments to balance computational resources and performance.

C.2. Considering Dependency

We evaluate our approach with and without considering parameter dependency, as shown in Table 5 and Table 6.

	data	CIFAR100		Tiny-ImageNet		Caltech101	
model	dep	ℓ^2	Taylor	ℓ^2	Taylor	ℓ^2	Taylor
DeiT	✗	**88.05**	**<u>88.70</u>**	**89.31**	**<u>89.69</u>**	**95.01**	**<u>95.41</u>**
	✓	86.43	87.33	85.56	85.92	65.35	78.04
ViT	✗	**87.13**	**<u>88.06</u>**	**90.78**	**<u>90.87</u>**	**92.69**	**<u>93.96</u>**
	✓	85.24	86.83	88.83	88.95	56.30	77.82
RN	✗	**82.25**	**<u>82.36</u>**	**<u>79.83</u>**	**79.66**	**<u>93.13</u>**	**92.56**
	✓	78.63	78.62	69.87	69.24	54.68	52.71
RNX	✗	**<u>86.12</u>**	**85.94**	**<u>83.88</u>**	**<u>83.88</u>**	**95.71**	**<u>95.84</u>**
	✓	84.71	85.01	79.39	78.95	92.13	91.82

Table 5. Fine-tuning image models by our SPruFT for 5 epochs. "dep" refers to whether parameter dependencies are involved or not. ℓ^2 and Taylor represent the magnitude and Taylor importance. All reported results are validation accuracies. **Bold** indicates the superior results achieved through dependency searching compared to not searching. <u>Underline</u> highlights the best fine-tuning results.

We utilize various importance metrics to fine-tune both models using our approach, with and without incorporating parameter dependencies, and report the results to compare their performances. Searching for dependencies in structured pruning is natural, as dependent parameters are pruned together. However, important neurons in a given layer do not always have dependent neurons that are also important in their respective layers. As demonstrated in Table 5, fine-tuning without considering parameter dependencies outperforms fine-tuning incorporating dependencies in all cases. For importance metrics, although the differences between them are not substantial, all results consistently conclude that the Quantile-Mean Taylor importance demonstrates a slight improvement over the standard Taylor importance. Furthermore, both the Quantile-Mean Taylor and standard Taylor metrics outperform the magnitude importance.

	task	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B
imp	dep	mcc	acc	acc	acc	acc	acc	acc	corr
Taylor	✗	65.56	89.55	**87.50**	93.15	**91.57**	**80.14**	**95.41**	89.14
	✓	**67.49**	**89.85**	87.25	**93.30**	91.63	79.42	95.07	**89.98**
ℓ^2	✗	65.40	89.77	83.33	92.64	91.34	74.73	94.04	**88.69**
	✓	**66.80**	**90.22**	**84.07**	**93.94**	**91.57**	**79.06**	**95.07**	87.39

Table 6. Fine-tuning DeBERTaV3 on GLUE by our SPruFT for 10 epochs. "dep" refers to whether parameter dependencies are involved or not. Taylor and ℓ^2 indicate the magnitude and Taylor importance. The importance score is Taylor. We do not apply QMTaylor since the number of labels is tiny. 'mcc', 'acc', and 'corr' represent 'Matthews correlation', 'accuracy', and 'Pearson correlation', respectively. All reported metrics are based on validation results. **Bold** indicates the best results of whether considering dependencies.

Table 6 suggests a slightly different conclusion: the impact of parameter dependencies on performance is minor, nearly negligible[6]. However, searching for dependencies involves additional implementations and computational overhead. Combining the results of image models, the conclusion is not searching for the parameter dependencies. For importance metrics, this experiment shows that magnitude and Taylor importance perform similarly.

[6]The results of using magnitude importance on the RTE task show significant variation, but this is likely due to the small sample size and the hardness of the task, which result in the unstable performances observed in our experiments. Aside from RTE, the results on other tasks are not significantly different.

C.3. Memory Measurement

In this study, we detail the memory measurement methodology employed. The total memory requirements can be categorized into three main components:

$$\text{mem}_{\text{TTL}} = \text{mem}_{\text{M}} + \text{mem}_{\text{FT}} + \text{mem}_{\text{Aux}},$$

where:

1. mem_{TTL} is the total memory consumed during training.
2. mem_{M} represents the memory consumed by the base model itself.
3. mem_{FT} corresponds to the memory required for the fine-tuning parameters and their gradients.
4. mem_{Aux} accounts for any additional memory usage, including optimizer states, caching, and other intermediate computations.

We yield mem_{M} by measuring the memory usage during inference on the training data using the pre-trained model. The combined memory usage of mem_{FT} and mem_{Aux} is calculated as the difference between mem_{TTL} and $\text{mem}_{\text{Model}}$. For simplicity, we consistently report $\text{mem}_{\text{FT}} + \text{mem}_{\text{Aux}}$ as "mem" in this study.

	Llama2(7B)				Llama3(8B)			
FT setting	#param	mem_{TTL}	mem_{M}	mem	#param	mem_{TTL}	mem_{M}	mem
LoRA, $r = 64$	159.9M(2.37%)	53.33GB	29.87GB	23.46GB	167.8M(2.09%)	64.23GB	33.86GB	30.37GB
RoSA, $r = 32, d = 1.2\%$	157.7M(2.34%)	74.56GB	29.87GB	44.69GB	167.6M(2.09%)	82.26GB	33.86GB	48.40GB
DoRA, $r = 64$	161.3M(2.39%)	74.72GB	29.87GB	44.85GB	169.1M(2.11%)	85.31GB	33.86GB	51.45GB
VeRA, $r = 64$	1.4M(0.02%)	52.84GB	29.87GB	22.97GB	1.4M(0.02%)	63.35GB	33.86GB	29.49GB
SPruFT, $r = 128$	145.8M(2.16%)	**47.49GB**	29.87GB	**17.62GB**	159.4M(1.98%)	**58.35GB**	33.86GB	**24.49GB**

Table 7. The requirements of computation resources for fine-tuning. 'mem' traces $\text{mem}_{\text{TTL}} - \text{mem}_{\text{M}}$. All fine-tuning parameters are stored in full precision and the max token is 2048. We also examined the training time and observed that DoRA requires 50% to 100% more time than other methods, while LoRA, RoSA, and our approach need similar training time (differing only by a few seconds). However, due to the influence of various factors on training time and the difficulty of ensuring a fair comparison, we chose not to include these results in our report.

In Figure 2, we yield the memory cost of 'dropout' by tracing the memory usage of fine-tuning Llama2 using these PEFT methods with and without dropout. Additionally, we examined memory usage under varying r and d to obtain the maximum potential memory savings from reducing the number of trainable parameters in each method. For instance, in the case of LoRA, fine-tuning the model with $r = 64$ and $r = 32$ shows a memory saving of approximately 1.25GB when reducing r from 64 to 32. This implies that the memory consumed by trainable parameters in LoRA with $r = 64$ is approximately $1.25 \times 2 = 2.51$GB. Table 8 shows the exact values of Figure 2 for max token$= 2048$. Notably, we also use RoSA to demonstrate that quantizing trainable parameters (Gholami et al., 2022; Dettmers et al., 2022; 2024) with mixed-precision training (Micikevicius et al., 2018) saves significantly more memory than simply reducing trainable parameters, as shown by the memory difference between RoSA and RoSA-bf16 in Figure 2 (see also related works on quantization (Dettmers et al., 2022; Guo et al., 2024a; Li et al., 2024; Dettmers et al., 2024)).

C.4. Resource Requirements

Table 7 and Table 8 presents the resource requirements of various PEFT methods. We compare our approach with LoRA and several of its variants that maintain or surpass LoRA's performance. As shown, our method is the most resource-efficient among these approaches. The subsequent ablation study further demonstrates that our approach achieves performance comparable to LoRA. We exclude comparisons with VeRA (Kopiczko et al., 2024), which proposes sharing a single pair of random low-rank matrices across all layers to save memory footprint. While VeRA achieves some memory savings, its performance often deteriorates.

We note that while our approach offers significant memory efficiency, this benefit is less pronounced in small-scale models, where the primary memory consumption arises from the dataset—especially with large batch sizes. The main advantage of our method in these cases is the reduced FLOPs due to fewer trainable parameters. Therefore, we do not highlight memory efficiency in small-scale model scenarios.

PEFT	Llama2(7B)				
	#param	mem	dropout	param	other
LoRA	159.9M(2.37%)	23.46GB	6.18GB	2.51GB	14.77GB
RoSA	157.7M(2.34%)	44.69GB	6.22GB	1.72GB	36.75GB
RoSA-bf16	157.7M(2.34%)	39.55GB	6.22GB	0.92GB	32.41GB
DoRA	161.3M(2.39%)	44.85GB	6.29GB	2.49GB	36.07GB
VeRA	1.37M(0.02%)	22.97GB	2.15GB	0.11GB	20.71GB

Table 8. The requirements of computation resources for fine-tuning full precision Llama2 using LoRA (Hu et al., 2022), RoSA (Nikdan et al., 2024), DoRA (Liu et al., 2024), and VeRA (Kopiczko et al., 2024). In this analysis, we set $r = 64$ for LoRA, DoRA, and VeRA and set $r = 32, d = 1.2\%$ for RoSA. The fine-tuning parameters of RoSA-bf16 are in bfloat16 under mixed-precision training (Micikevicius et al., 2018), while all others are in full precision. Note that RoSA has its own official implementation, whereas LoRA, DoRA, and VeRA are integrated into the PEFT library provided by Hugging Face. This may influence the memory used by each method. 'mem' traces the peak memory usage during training, excluding the memory consumed by the model itself, while 'dropout' accounts for the memory consumption associated with LoRA's dropout layer. 'param' represents the maximum memory savings from reducing the number of trainable parameters. This is quantified by varying r and d for each method and calculating the memory difference attributed to trainable parameters. 'other' indicates the memory usage from any other intermediate values in backpropagation and optimizer states.

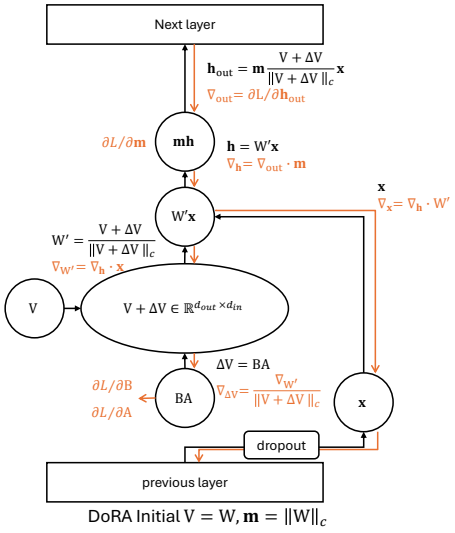

Figure 4. The illustration of DoRA's computation graph. Black operations occur during the forward pass, while orange operations take place during the backward pass.

In Section 3, we explain that the memory usage of DoRA is significantly higher than that of LoRA due to its complex computation. We demonstrate the computation graph of DoRA here, as shown in Figure 4. DoRA decomposes $\mathbf{W}$ into magnitude $\mathbf{m}$ and direction $\mathbf{V}$ and computes the final parameters matrix by $\mathbf{W}' = \mathbf{m}\frac{\mathbf{V}+\Delta\mathbf{V}}{||\mathbf{V}+\Delta\mathbf{V}||_c}$. This complicated computation significantly increases memory usage because it requires caching a lot of intermediate values for computing gradients of $\mathbf{B}$, $\mathbf{A}$, and $\mathbf{m}$. As illustrated in Figure 4, each node passed by backpropagation stores some intermediate values for efficient gradient computing.

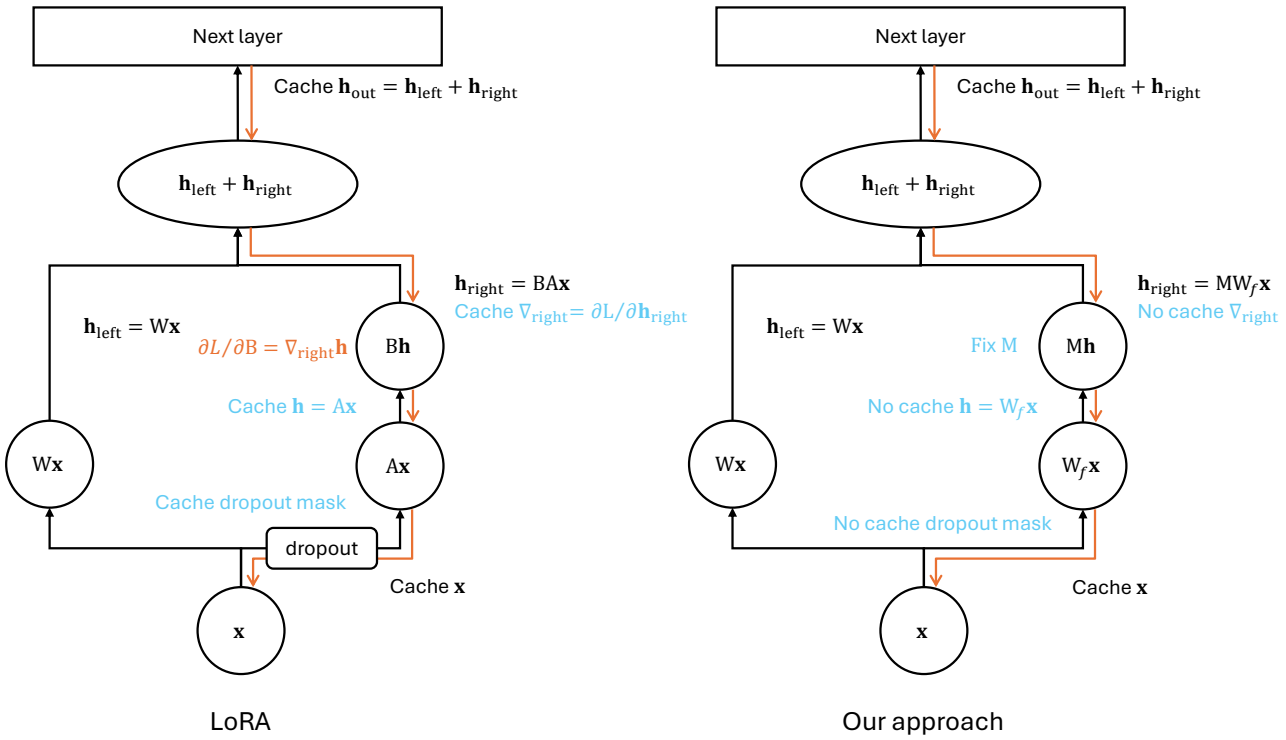

Figure 5. The illustration of backpropagation highlights the operations involved. Black operations occur during the forward pass, while orange operations take place during the backward pass. Blue operations highlight the benefits of our approach. Notably, since $\mathbf{M}$ is non-trainable, caching $\Delta\mathbf{W}\mathbf{x}$ during the forward pass is unnecessary, leading to significant memory savings. Additionally, in practice, PyTorch caches $\frac{\partial L}{\partial \mathbf{h}_{right}}$ to efficiently compute $\frac{\partial L}{\partial \mathbf{B}}$, although this caching is not strictly required for backpropagation.

C.5. Cache Benefit

In the main context, we have already shown the memory cost of dropout layers in LoRA, in this section, we will discuss some other benefits of our approach. Figure 5 illustrates the computation and cache requirements in backpropagation (Rumelhart et al., 1986). For simplicity, we replace the notation $f(\cdot,\cdot)$ with different $\mathbf{h}$. With the same number of trainable parameters, our approach eliminates the need to cache $\mathbf{h} = \Delta\mathbf{W}\mathbf{x}$ shown in the figure. While this benefit is negligible under lower rank settings (r) or when the number of fine-tuning layers is small, it becomes significant as the model size and rank settings increase. Although the caching requirement for $\mathbf{h}$ can be addressed by recomputing $\mathbf{h} = \mathbf{A}\mathbf{x}$ during backpropagation, this would result in increased time complexity during training.

D. Details of Datasets

D.1. Vision Benchmarks

CIFAR100: CIFAR100 (Krizhevsky et al., 2009) has 100 classes with 600 images of size 32x32 per class, while the CIFAR10 has 10 classes with 6000 images per class. In this study, we use the CIFAR100 downloaded from huggingface (`https://huggingface.co/datasets/uoft-cs/cifar100`) with 500 training images and 100 validation images per

class. In our experiments, we resize the images to 256x256, crop the center to 224x224, and normalize them using the CIFAR mean $(0.507, 0.487, 0.441)$ and standard deviation $(0.267, 0.256, 0.276)$ for the three channels.

Tiny-ImageNet: Tiny-ImageNet (Tavanaei, 2020) has 200 classes with images of size 64x64, while the full ImageNet-1k (Deng et al., 2009) has all 1000 classes where each image is the standard size 224x224. In this study, we use the Tiny-ImageNet downloaded from huggingface (`https://huggingface.co/datasets/zh-plus/tiny-imagenet`) with 500 training images and 50 validation images per class. In our experiments, we resize the images to 256x256, crop the center to 224x224, and normalize them using the mean $(0.485, 0.456, 0.406)$ and standard deviation $(0.229, 0.224, 0.225)$ for the three channels.

caltech101: Caltech101 (Li et al., 2022) consists of 101 classes, with images of varying sizes typically having edge lengths between 200 and 300 pixels. Each class contains approximately 40 to 800 images, resulting in a total of around 9,000 images. In this study, we use the Caltech101 dataset provided by PyTorch (`https://pytorch.org/vision/main/generated/torchvision.datasets.Caltech101.html`), allocating 75% of the images for training and the remaining 25% for validation. In our experiments, we preprocess the images by resizing them to 256×256, cropping the center to 224×224, and normalizing them using the mean $(0.485, 0.456, 0.406)$ and standard deviation $(0.229, 0.224, 0.225)$ for the three channels.

D.2. General Language Understanding Evaluation Benchmark (GLUE)

CoLA: The Corpus of Linguistic Acceptability (CoLA) is a dataset for assessing linguistic acceptability (Warstadt et al., 2018). This task is a binary classification for predicting whether a sentence is grammatically acceptable. The dataset is primarily from books and journal articles on linguistic theory.

MNLI: The Multi-Genre Natural Language Inference (MultiNLI) is a dataset designed to evaluate a model's ability to perform natural language inference (NLI). The task is to predict whether the premise entails the hypothesis, contradicts the hypothesis, or neither. The data set contains 433k sentence pairs annotated with textual entailment information (Williams et al., 2018).

MRPC: The Microsoft Research Paraphrase Corpus (Dolan & Brockett, 2005) is a dataset designed for evaluating paraphrase detection systems. It consists of sentence pairs, with binary labels of whether the two sentences in the pair are equivalent. The data are automatically extracted from online news and labeled by humans.

QNLI: The Stanford Question Answering Dataset (SQuAD) is a dataset designed for machine comprehension of text (Rajpurkar et al., 2016). The dataset consists of question-paragraph pairs, where one of the sentences in the paragraph contains the answer to the corresponding question. The paragraphs are from Wikipedia and the questions are written by human annotators.

QQP: The Quora Question Pairs (QQP) dataset is a dataset of question pairs (`https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs`). The task is to determine whether two questions are semantically equivalent.

RTE: The Recognizing Textual Entailment (RTE) datasets are a series of challenges that evaluate models' ability to determine whether a premise can entail a given hypothesis (Dagan et al., 2006; Bar-Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009). The data are constructed based on the texts from Wikipedia and news. The datasets have been used to evaluate the performance of both traditional language models and the state-of-the-art LLMs.

SST-2: The Stanford Sentiment Treebank is a dataset of sentences extracted from movie reviews (Socher et al., 2013). Each sentence is labeled as either positive or negative. The task is to predict whether the sentence is positive or negative.

STS-B: The Semantic Textual Similarity Benchmark (STSB) is a dataset with sentence pairs collected from news headlines, video and image captions, and natural language inference data (Cer et al., 2017). The task is to predict the semantic similarity between pairs of sentences. Each pair of sentences is annotated with a similarity score ranging from 0 to 5, where 0 indicates no semantic similarity and 5 indicates semantically equivalent.

D.3. Text-Generation Datasets

GSM8k: GSM8K (Grade School Math 8K) is a dataset of 8792 high-quality grade school math problems, including problems in diverse languages. These problems take between 2 and 8 steps of elementary calculations using basic arithmetic

operations ($+ - \times \div$) to solve. The dataset was created to support the task of question answering on basic mathematical problems to evaluate the model's ability of basic arithmetic reasoning.

ARC: The AI2 Reasoning Challenge (ARC) dataset consists of grade-school level, multiple-choice science questions (Clark et al., 2018). ARC dataset includes a Challenge Set and an Easy Set. The easy set contains questions that can be answered with straightforward reasoning, while the challenge set requires deeper understanding and more reasoning skills. The ARC-Easy includes 2251 training samples, 570 validation samples, and 2376 test samples and the ARC-Challenge includes 1119 training samples, 299 validation samples, and 1172 test samples.

BoolQ: Boolean Questions (BoolQ) is a dataset of yes/no question answering (Clark et al., 2019) and includes 9427 training samples and 3270 validation samples. The dataset is designed to assess models' comprehension and reasoning abilities. Each example contains question, passage, answer, and title.

HellaSwag: HellaSwag is a dataset designed to evaluate the models' abilities in generating reasonable contexts (Zellers et al., 2019). It consists of prompts with a short context followed by multiple possible continuations. The goal is to find the correct or most plausible option. The training set, validation set, and test set have 39905 samples, 10042 samples, 10003 samples, respectively.

OpenBookQA: OpenBookQA is a question-answering dataset (Mihaylov et al., 2018) comprising 4957 training samples, 500 validation samples, and 500 test samples. It requires reasoning ability and a deeper understanding of common knowledge to answer questions. Each data contains a short passage with multiple possible answers. The dataset emphasizes the integration of world knowledge and reasoning skills, making it a challenging benchmark for natural language processing models. It tests models' abilities to understand and apply factual information effectively to solve problems.

WinoGrande: WinoGrande is a dataset of 44k problems for choosing the right option for a given sentence (Sakaguchi et al., 2021). It includes 40938 samples in the training set, 1,267 in the validation set, and 1,267 in the test set. The dataset is designed to assess models' commonsense reasoning abilities. The examples contain sentences with fill-in-blanks that require the model to select the most appropriate option to complete the sentence.

SocialIQA: The SocialIQA dataset is a benchmark designed to evaluate a model's ability to reason about social interactions, including understanding social dynamics, intentions, and the effects of human actions (Sap et al., 2019). SocialIQA includes 33410 samples in the training set and 1954 in the validation set.

PIQA: The PIQA (Physical Interaction Question Answering) dataset is a benchmark designed to evaluate a model's ability to understand and reason about everyday physical interactions and affordances (Bisk et al., 2020). Here are some key details about PIQA: (Sakaguchi et al., 2021). PIQA contains 16113 samples in the training set and 1838 in the validation set.