NeuroAda: Activating Each Neuron's Potential for Parameter-Efficient Fine-Tuning

Zhi Zhang[*]
ILLC, University of Amsterdam
`zzhang2626@gmail.com`

Yixian Shen[*]
PCS, University of Amsterdam
`y.shen@uva.nl`

Congfeng Cao
ILLC, University of Amsterdam
`c.cao@uva.nl`

Ekaterina Shutova
ILLC, University of Amsterdam
`e.shutova@uva.nl`

Abstract

Existing parameter-efficient fine-tuning (PEFT) methods primarily fall into two categories: addition-based and selective in-situ adaptation. The former, such as LoRA, introduce additional modules to adapt the model to downstream tasks, offering strong memory efficiency. However, their representational capacity is often limited, making them less suitable for fine-grained adaptation. In contrast, the latter directly fine-tunes a carefully chosen subset of the original model parameters, allowing for more precise and effective adaptation, but at the cost of significantly increased memory consumption. To reconcile this trade-off, we propose NeuroAda, a novel PEFT method that enables fine-grained model finetuning while maintaining high memory efficiency. Our approach first identifies important parameters (i.e., connections within the network) as in selective adaptation, and then introduces bypass connections for these selected parameters. During finetuning, only the bypass connections are updated, leaving the original model parameters frozen. Empirical results on **23+** tasks spanning both natural language generation and understanding demonstrate that NeuroAda achieves state-of-the-art performance with as little as $\leq$ **0.02**% trainable parameters, while reducing CUDA memory usage by up to **60%**. We release our code here: https://github.com/FightingFighting/NeuroAda.git.

1 Introduction

Large language models (LLMs) demonstrate remarkable generalization capabilities on various NLP tasks (Dong et al., 2023; Li et al., 2025), yet achieving optimal performance on downstream tasks often still requires fine-tuning. As model sizes grow, full-parameter fine-tuning becomes increasingly impractical due to substantial computational and memory demands. For example, fine-

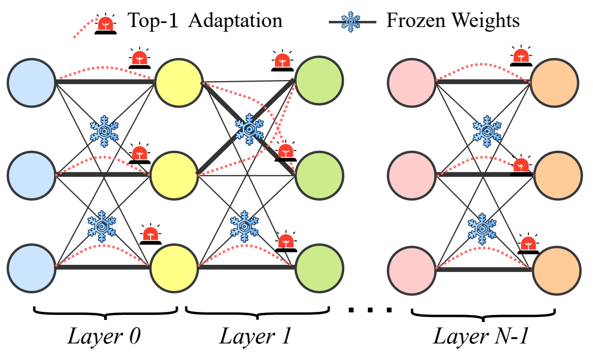

Figure 1: Overview of NeuroAda. For each neuron, top-1 weights are adapted, while the rest remain frozen. Bold dark indicates selected pretrained weights; red dashed edges represent newly introduced trainable parameters.

tuning LLaMA 2-13B without CPU offloading requires 26 GB for trainable parameters in FP16, 52 GB for Adam optimizer states (two FP32 moments per parameter), 26 GB for gradients, and an additional 2–4 GB for activations depending on batch size and sequence length. This results in a memory footprint of approximately 106–108 GB in total, far exceeding the capacity of commodity GPUs and necessitating premium hardware (e.g., A100 80G). This highlights the pressing need for more efficient and scalable adaptation strategies.

A growing body of work on parameter-efficient fine-tuning (PEFT) addresses the computational and memory overhead of full-model adaptation by introducing a set of trainable parameters while keeping the backbone frozen. One major class of these methods is known as *addition-based adaptation*, which augments the pretrained model with additional modules designed to inject task-specific flexibility. These additions vary in form and location, including adapter layers inserted into projection blocks (Pfeiffer et al., 2020; Sung et al., 2022), nonlinear activation reparameterizations (Zhang et al., 2021), prompt tuning applied to input embed-

[*]Equal contribution.

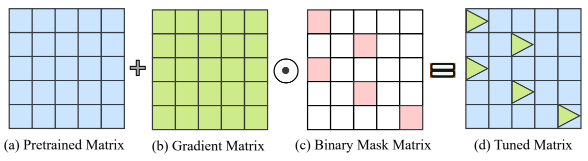

Figure 2: Mask-based sparse tuning employs a binary mask matrix to suppress gradient updates for unselected parameters. However, this approach incurs significant memory overhead, as gradients for the entire original parameter matrix must still be computed and retained by the optimizer.

dings (Lin et al., 2020; Liao et al., 2023b,a), latent representation perturbations (Wu et al., 2024a), and low-rank matrix decompositions applied directly to weight spaces, such as in LoRA (Hu et al., 2022a) and its variants (Zhang et al., 2023; Kopiczko et al., 2023). These methods typically offer improved memory efficiency by restricting gradient computation and optimizer state updates to the newly introduced modules, as opposed to the entire model in full fine-tuning. However, their scalability is constrained: as model size increases, the limited representational capacity of the added modules often leads to diminishing returns (He et al., 2024).

Another prominent line of research is *selective in-situ adaptation*, which fine-tunes a carefully selected subset of a pretrained model's original parameters, without introducing any additional parameters, modules, or layers. Structure-based approaches, such as BitFit (Ben Zaken et al., 2022a) and Partial-k (Jia et al., 2022) updating only the bias terms and the last k layers, respectively, exemplify early efforts in this direction. More recently, fine-grained and unstructured parameter selection methods have attracted increasing attention. These approaches aim to identify task-relevant parameters at a more granular level, such as GPS (Zhang et al., 2024b) and SPT (He et al., 2023), which demonstrate strong performance on vision tasks by selectively fine-tuning subsets of parameters that are most critical for the target task. Compared to structured approaches and *addition-based adaptation* methods, these unstructured strategies offer greater flexibility in parameter selection, enabling more precise and targeted model adaptation, and thereby substantially improving downstream performance (Shen et al., 2024; Fu et al., 2023). However, this sparse tuning paradigm leads to higher memory usage due to mask-based implementations. As shown in Figure 2, although only a small portion of the parameters is selected for updating, memory consumption remains comparable to that of full fine-tuning. This limitation becomes particularly problematic with the increase of model size (Zhai et al., 2022; Shen et al., 2024; Dong et al., 2025b).

These limitations motivate us to explore whether a unified approach can be designed that achieves the fine-grained parameter tuning characteristic of selective in-situ adaptation, while retaining the memory efficiency advantages of addition-based methods. To this end, we propose **NeuroAda**, an additive, overlay-style adaptation method that utilizes a carefully designed approach to introduce new parameters to enable fine-grained adjustments while maintaining memory efficiency. Specifically, as shown in Figure 1, NeuroAda first selects the top-k highest-magnitude input connections (weights/parameters) for each neuron in the network prior to finetuning and then, for each selected parameter, a bypass connection (initialized to zero) is introduced. During finetuning, only the newly introduced parameters are fine-tuned, while the original parameters remain frozen. This approach inherits both the performance benefits of *selective in-situ adaptation* and the memory efficiency of *addition-based methods*. Crucially, for each neuron, at least one of its input connections is selected for update, ensuring that all neurons have the potential to modify their activation states and thus change the state of the entire network for effective adaptation. Consequently, NeuroAda presents a scalable and practical solution for large LLMs.

NeuroAda offers four key advantages that make it both practical and effective in real-world scenarios: (1) **Highly efficient computation:** NeuroAda eliminates the need of the mask for sparse finetuning, which typically requires full gradient computation. (2) **Highly efficient GPU memory usage:** Only the newly added parameters are updated, significantly lowering memory usage by avoiding optimizer state tracking for the full model. (3) **Task-agnostic and generalizable:** Parameter selection is based on weight magnitudes from the pretrained model, enabling consistent selection across tasks, making the method broadly applicable and easy to deploy. (4) **Fine-grained, neuron-level adaptation:** NeuroAda ensures every neuron has the potential to change the activation state of each neuron during finetuning, maximizing the representational expressiveness of individual neurons. Empirically, NeuroAda achieves state-of-the-art performance on 23+ tasks compared with other PEFT methods, including both natural language generation and understanding, highlighting its practical effectiveness.

2 Related Work

Addition-based Adaptation. Includes adapter-based methods (He et al., 2021; Pfeiffer et al., 2020; Lin et al., 2020; Liao et al., 2023b,a) and low-rank reparameterization techniques such as LoRA (Hu et al., 2022a) and its variants AdaLoRA (Zhang et al., 2023), VeRA (Kopiczko et al., 2023), QLoRA (Dettmers et al., 2024), and DoRA (Liu et al., 2024), which introduce trainable low-rank matrices into projection layers. While LoRA avoids inference-time overhead by merging updates into base weights, it often suffers from scalability issues and diminishing returns when applied to large models or complex tasks (Liu et al., 2024). A parallel direction modifies hidden states instead of weights: activation steering (Liu et al., 2023; Li et al., 2023), concept erasure (Belrose et al., 2023; Avitan et al., 2024; Singh et al., 2024), and block-level editing (Wu et al., 2024a) offer instance-specific control but require task-specific adaptation.

Selective In-Situ Adaptation. This class of work fine-tunes a subset of the model's original parameters without introducing any additional modules or weights, often achieving strong performance with minimal architectural changes. However, its practical memory and compute benefits frequently fall short in large-scale settings. Methods such as SIFT (Song et al., 2023), SHiRA (Bhardwaj et al., 2024), SpIEL (Ansell et al., 2024) and work from (Zhang et al., 2024a) enforce sparsity constraints, yet still require full backward passes to compute gradients for the entire weight space. More targeted approaches, including SMT (He et al., 2024) and GPS (Zhang et al., 2024b), improve efficiency by selecting submatrices or top-k gradients per neuron, but rely on gradient-based warm-up and dynamic masking. These mechanisms introduce additional overhead from binary mask storage, dense optimizer states, and full-gradient tracking, making them difficult to scale to large language models. In contrast, our method, NeuroAda, inherits the advantages of both paradigms by avoiding gradient-based selection and selecting the top-k weights per neuron.

3 Methodology

In this section, we first present the necessary preliminaries, and then introduce NeuroAda, a new adaptation framework that activates each neuron's potential by selectively updating a small subset

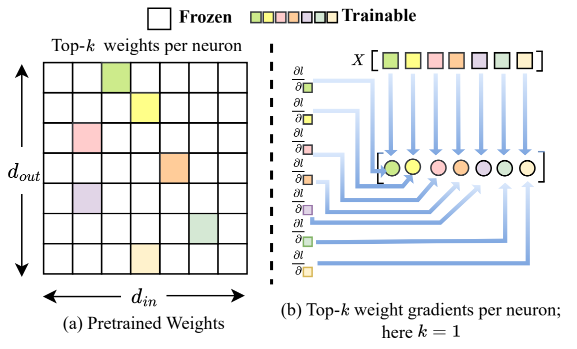
Frozen
Trainable
Top-k weights per neuron
d_{out}
d_{in}
(a) Pretrained Weights
x
$\frac{\partial l}{\partial}$
(b) Top-k weight gradients per neuron; here $k = 1$

Figure 3: Neuron-wise Top-k Weight Selection and Gradient Computation. (a) Pretrained weight matrix of size $d_{out} \times d_{in}$, where for each neuron (row), only the top-k weights(i.e., highest-magnitude) are selected for adaptation (colored), and the rest remain frozen (white). (b) Corresponding gradient matrix restricted to the top-k weights per neuron (here $k = 1$), showing gradients only for trainable entries. This strategy enables fine-grained, neuron-level adaptation while preserving most of the pretrained model, effectively activating each neuron's potential through less-invasive tuning.

of its weights. Specifically, we freeze all pre-trained model weights and introduce sparse, additive overlay-style adaptation method in which top-k bypasses of input connections (weights/parameters) are introduced into each neuron in the neural network for adaptation. This neuron-wise adaption preserves the original parameters intact while enabling targeted learning signals at fine granularity. As illustrated in Figure 3, NeuroAda ensures that every neuron participates in adaptation, supporting both efficiency and generalization. During inference, the small number of learned deltas can be merged into the base weights, resulting in no additional overhead at runtime.

3.1 Preliminaries

Let $\mathcal{M}_{\mathbf{\Phi}}$ be an L-layer pretrained language model with parameters $\mathbf{\Phi} = \{\mathbf{W}^{(\ell)}, \mathbf{b}^{(\ell)}\}_{\ell=1}^{L}$. For any linear sub-layer we write $\mathbf{h}_{\text{out}} = \mathbf{W}\mathbf{h}_{\text{in}} + \mathbf{b}$, where $\mathbf{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ and each *row* of $\mathbf{W}$ corresponds to a **neuron**. During standard fine-tuning all entries of $\mathbf{W}$ are updated, yielding heavy computational and memory costs (§1).

Sparse additive updates. NeuroAda freezes $\mathbf{\Phi}$ and introduces a *delta-parameter tensor* $\mathbf{\Delta}$ with the *same shape* as $\mathbf{\Phi}$ but *sparsity constrained*:

$$\mathbf{\Phi}' = \mathbf{\Phi} + \mathbf{\Delta}, \qquad \|\mathbf{\Delta}\|_0 \ll \|\mathbf{\Phi}\|_0, \tag{1}$$

where $\|\cdot\|_0$ counts non-zero elements. Only $\mathbf{\Delta}$ is trainable; the base model remains intact, so $\mathbf{\Delta}$ can

Table 1: Memory comparison per projection. Mask-based sparse tuning methods require 1 bit per weight[1]. NeuroAda with $k = 1$ only stores one BF16 value (2 bytes) and one index (2 bytes) per row, totaling 4 bytes per neuron. This yields over $100\times$ memory savings for a single linear layer.

Model	d_{model}	**Mask [MB]**	**NeuroAda[MB]**	**Saving Ratio**
LLaMA-1 7B	4096	$\frac{4096^2}{8\times2^{20}} \approx 2.00$	$\frac{4096\times4}{2^{20}} \approx 0.016$	$\approx 125\times$
LLaMA-2 7B	4096	2.00	0.016	$125\times$
LLaMA-1 13B	5120	$\frac{5120^2}{8\times2^{20}} \approx 3.13$	$\frac{5120\times4}{2^{20}} \approx 0.020$	$\approx 156\times$
LLaMA-2 13B	5120	3.13	0.020	$156\times$

be *merged in-place* after training, incurring zero inference overhead.

3.2 Top-k selection

A core design goal is that *every neuron receives at least a small learning signal.* For each neuron—that is, for each row $\mathbf{w} \in \mathbb{R}^{d_{\text{in}}}$ of a weight matrix, we identify the indices of its k largest-magnitude components:

$$\mathcal{I}(\mathbf{w}) \;=\; \underset{j\in\{1,\dots,d_{\text{in}}\}}{\arg\text{top k}} \,\big|\mathbf{w}_j\big|. \tag{2}$$

We then allocate trainable delta weights only at these positions:

$$\big[\boldsymbol{\Delta}\big]_{i,j} \;=\; \begin{cases} \theta_{i,j} & \text{if } j \in \mathcal{I}(\mathbf{w}_i) \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where $\theta_{i,j}$ is a trainable parameter defined only for $j \in \mathcal{I}(\mathbf{w}i)$ and initialized to 0. For all other positions, $\boldsymbol{\Delta}i, j$ is fixed to zero and excluded from both optimization and memory storage. While NeuroAda uses weight magnitude for top-k selection, the framework is flexible: task-guided criteria such as gradient magnitude or random ticketing can be substituted into $\mathcal{I}(\cdot)$. We employ magnitude due to its task-agnostic stability and the advantage of requiring no warm-up or additional computation. This design choice is empirically validated in our ablation study, where magnitude-based selection achieves strong performance without relying on task-specific signals, as shown in Figure 7.

Mask-free implementation. Since the top-k sparsity pattern is determined *a priori*, Eq. (3) can be implemented without maintaining a full binary mask over the weight matrix. Instead, we store a compact list of *indices* and corresponding *BF16 values*—only k entries per row—eliminating the need for dense masking or indexing during training. This design leads to substantial memory savings and indexing efficiency. As shown in Table 1, for a single projection layer in LLaMA-2 13B, a 1-bit-per-weight binary mask requires over 3 MB of memory, while NeuroAda with k=1 uses only 0.02 MB—over **156**$\times$ smaller. These savings scale across layers and are especially beneficial for high-throughput training on limited-memory devices.

3.3 Featherlight adaptation

During fine-tuning we optimize only $\{\theta_{i,j}\}$ while re-using the forward path of the frozen backbone. For a linear layer the forward pass becomes

$$\mathbf{h}_{\text{out}} \;=\; \underbrace{\mathbf{W}\mathbf{h}_{\text{in}}}_{\text{frozen}} \;+\; \underbrace{\big(\mathbf{P}\odot\boldsymbol{\Theta}\big)\mathbf{h}_{\text{in}}}_{\text{trainable }\boldsymbol{\Delta}}, \tag{4}$$

where $\mathbf{P}$ is an index matrix with zeros everywhere except $\big[P\big]_{i,j} = 1$ when $(i, j) \in \mathcal{I}(\mathbf{w}_i)$, $\odot$ denotes element-wise product, and $\boldsymbol{\Theta}$ is the dense tensor of learnable $\theta_{i,j}$.[2]

Lightweight backward pass and optimizer states. During back-propagation, NeuroAda updates only the k selected coordinates per neuron. As a result, the dominant memory contributors in full-model training—BF16/FP32 gradients and the two FP32 moment estimates in the AdamW optimizer—are reduced proportionally by a factor of $\frac{k}{d_{\text{in}}}$. Because all delta parameters are stored directly in BF16 and no FP32 master weights are needed, the optimizer maintains only $2 \times k$ FP32 values per row instead of $2 \times d_{\text{in}}$.This yields a substantial memory reduction in the optimizer state maintained by AdamW. In standard dense fine-tuning, AdamW stores two FP32 moment estimates (first and second moments) for each trainable parameter, resulting in:

$$\text{AdamW Mem. (Masked):} \quad 2 \times d_{\text{out}} \times d_{\text{in}} \times 4\text{(bytes)}, \tag{5}$$

where 4 bytes denotes the size of a 32-bit float. In contrast, NeuroAda updates only k weights per row, so the optimizer state becomes:

$$\text{AdamW Mem. (NeuroAda):} 2 \times d_{\text{out}} \times k \times 4\text{(bytes)}. \tag{6}$$

This reduces memory usage by a factor of $\frac{d_{\text{in}}}{k}$ per linear layer. For example, with $d_{\text{in}} = 5120$ and $k = 1$, this corresponds to a $5120\times$ reduction in AdamW state memory.

[1]While 1-bit-per-weight is a theoretical lower bound for binary mask storage, actual implementations in PyTorch and other frameworks use byte-addressable storage (e.g., `BoolTensor`), leading to significantly higher memory overhead.

[2]We implement this with fused scatter-add so the additional multiply is executed only on the k selected positions; no dense mask is materialised.

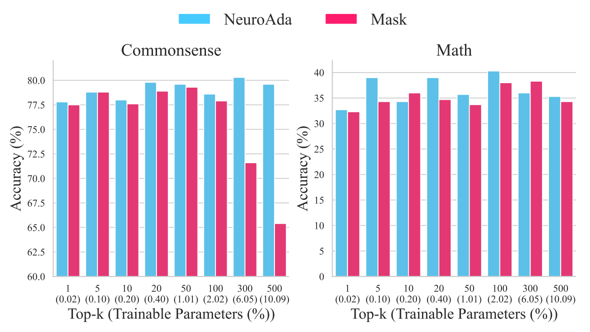

Figure 4: Performance comparison between our NeuroAda and mask-based methods on LLaMA-7B. Top-k means selecting top-k input connections per neuron in the neural network.

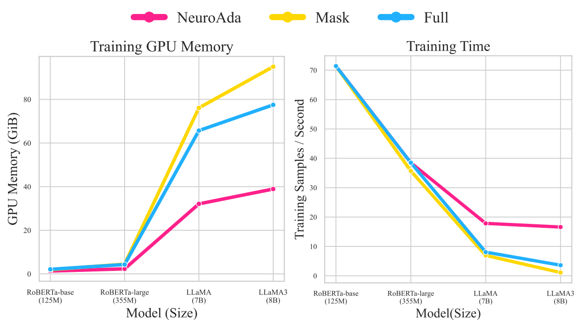

Figure 5: Training GPU memory and training efficiency on different models (RoBERTa-base, RoBERTa-large, LLaMA-7B, LLaMA3-8B) with NeuroAda, mask-based and full fine-tuning method.

4 Neuron-wise Sparse Adaptation: Comparative Analysis

In this section, we first compare the mask-based method, which applies binary masks to zero out the gradients of unselected (frozen) parameters (see Figure 2), with our NeuroAda, which introduces new trainable parameters to bypass the selected ones, rather than directly tuning them, for sparse fine-tuning. The comparison is conducted in terms of effectiveness, GPU memory usage, and training efficiency. We then further investigate the effectiveness of the proposed method NeuroAda, which aims to ensure that all neurons in the network have the potential to update their activation states during fine-tuning. This is done by analyzing the proportion of neurons involved in fine-tuning and examining different parameter selection strategies for activating them.

Experiment setup To ensure a fair comparison between our NeuroAda and mask-based methods, as well as across different parameter selection strategies, we conduct a hyperparameter search over the different learning rates for each experiment using the training set. The best-performing configuration is then selected based on validation set performance. This is necessary because PEFT methods are generally sensitive to the choice of learning rate (Wu et al., 2024b). The hyperparameter search space is presented in Table 7 in Appendix. The details of used datasets: COMMONSENSE15K and GSM8K are provided in Appendix C.1.

Question 1: *Can our method NeuroAda be a competitive or even superior alternative to the mask-based sparse tuning approach?* We address this by comparing their task performance, GPU memory usage, and training efficiency.

Performance To comprehensively and fairly evaluate the effectiveness of the two methods, we compare them under the same proportion of trainable parameters, ranging from 0.02% to 10%, on the COMMONSENSE15K and GSM8K tasks. As shown in Figure 4, our proposed method NeuroAda performs comparably to, or even better than, the mask-based method across most parameter budgets on both datasets. In particular, the NeuroAda outperforms the mask-based method by approximately 9% and 14% in accuracy when using 6.05% and 10.09% of trainable parameters, respectively, on the commonsense reasoning task.

Training memory and time We evaluate models of varying sizes, including RoBERTa-base, RoBERTa-large (Liu et al., 2019), LLaMA-7B (Touvron et al., 2023a), and LLaMA3-8B (Vavekanand and Sam, 2024). Specifically, we sample 500 examples from the MNLI task in the GLUE natural language understanding benchmark (Wang et al., 2019), and another 500 examples from the natural language reasoning task GSM8K. We use these samples to train the RoBERTa and LLaMA models, respectively. All experiments are conducted on a single NVIDIA H100 GPU with a batch size of 2. We report the GPU memory usage and training time for each model. Figure 5 shows our proposed addition-based sparse training method NeuroAda consumes less GPU memory compared to the mask-based counterpart, especially as the model size increases. For example, the NeuroAda achieves up to 60% memory savings on LLaMA3-8B. In addition, the NeuroAda enables significantly faster training, particularly for larger models. It processes 16.6 sam-

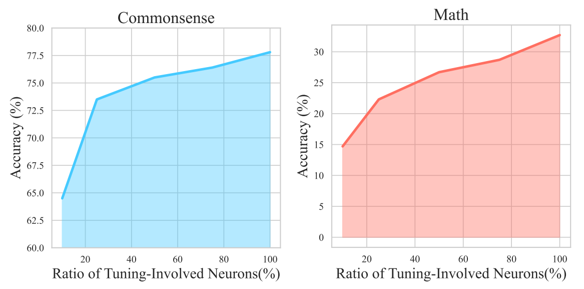

Figure 6: Comparison across different proportions of neurons involved in the fine-tuning process.

ples per second, compared to only 1.1 samples per second with the mask-based method.

Question 2: *How effective is the proposed method NeuroAda in enabling all neurons to update their activation states during fine-tuning for downstream task adaptation?* To answer this question, we first investigate how different parameter selection strategies can be used to ensure that all neurons have the potential to update their activation states during fine-tuning. We further analyze how task performance on COMMONSENSE15K and GSM8K varies with the proportion of neurons allowed to update their activation states during training.

Involved number of neurons. Our proposed method selects the top-k input connections for each neuron in the network, ensuring that at least one input connection per neuron is selected for tuning. This design enables all neurons to update their activation states during fine-tuning, allowing better adaptation to downstream tasks. To demonstrate its effectiveness, we select parameters from various proportion of neurons per layer for tuning and evaluate the resulting performance on the COMMONSENSE15K and GSM8K tasks. As shown in Figure 6, increasing the number of neurons involved during training leads to consistent performance improvements on both tasks. This suggests that enabling all neurons to update their activation states is beneficial—and likely necessary—for effective downstream task adaptation.

Different selection strategies To explore the effectiveness of enabling all neurons in the network to update their activation states during training, we experiment with different parameter selection strategies for each neuron under different trainable parameter budget. Specifically, for each neuron, we select the top-k input connections based on four criteria: highest weight magnitude, highest gradient absolute value, lowest weight magnitude, and random selection from all input connections. As shown in Figure 7, all selection methods yield comparable performance on both the COMMONSENSE15K and GSM8K tasks across different trainable parameter budgets. The average accuracies across all budgets for all selection methods are closely aligned, ranging from 77.69% to 79.24% on COMMONSENSE15K, and from 35.89% to 36.54% on GSM8K. These results again highlight the importance of involving all neurons in the adaptation process, regardless of the specific selection method used. Additionally, across both tasks, the *Magnitude* selection method achieves the highest win rate across different parameter budgets compared to the other strategies. Therefore, we adopt the *Magnitude* selection strategy as the default in NeuroAda.

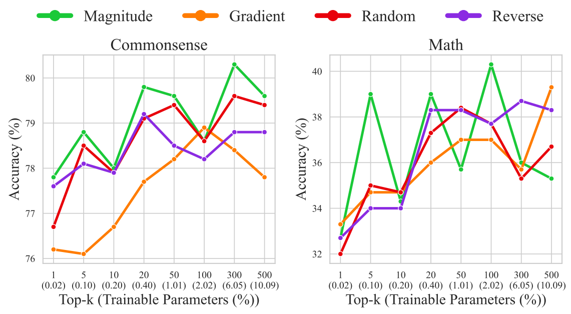

Figure 7: Comparison of different parameter selection strategies for involving neurons for the fine-tuning process. Among all input connections for each neuron in the network, Top-k connection with highest magnitude (*Magnitude*), highest gradient absolute value (*Gradient*), lowest magnitude (*Reverse*) are selected for training using addition-based method. *Random* means randomly selecting Top-k input connections per neuron.

5 Experiments

We evaluate NeuroAda on 23+ datasets spanning commonsense reasoning (Section 5.1), arithmetic reasoning (Section 5.2), and natural language understanding (Section 5.3). Experiments cover both encoder-only (RoBERTa-base (Liu et al., 2019)) and decoder-only (LLaMA (Touvron et al., 2023a,b)) models up to 13B parameters. We benchmark against strong PEFT baselines, including BitFit (Ben Zaken et al., 2022b), prefix-tuning (Li and Liang, 2021), adapters (He et al., 2021), LoRA (Hu et al., 2022b), DoRA (Liu et al., 2024), SMT (He et al., 2024), RED (Wu et al., 2024a), DiReFT, and LoReFT (Wu et al., 2024b). Following LoReFT, all models use `torch.bfloat16` and run on a single NVIDIA A100 or H100 GPU.

Our comparison considers not only benchmark

Table 2: Performance comparison with existing PEFT methods on eight commonsense reasoning datasets across four models: LLaMA-7B/13B, LLaMA2-7B, and LLaMA3-8B. *Most baseline results are taken from Hu et al. (2023a). †Results are from Wu et al. (2024b), ‡ results are taken from He et al. (2024) and *results are from Liu et al. (2024), as they share the same experimental setting with Hu et al. (2023a). For a fair comparison, our NeuroAda is also trained for 3 epochs to align with these baselines. $^{+}$When 3-epoch results are not available in the original paper, we re-trained the baselines using the official code and their reported best hyperparameters. All results for our method are averaged over three runs with different random seeds. Our method selects the top-20 and top-1 input connections per neuron for high-budget and low-budget parameter groups, respectively.

Model	**PEFT**	**Params (%)**	**Accuracy (↑)**								
							Commonsense Reasoning				
			BoolQ	**PIQA**	**SIQA**	**HellaS.**	**WinoG.**	**ARC-e**	**ARC-c**	**OBQA**	**Avg.**
ChatGPT*	–	–	73.1	85.4	68.5	78.5	66.1	89.8	79.9	74.8	77.0
LLaMA (7B)	Series*	1.953%	63.0	79.2	76.3	67.9	75.7	74.5	57.1	72.4	70.8
	Parallel*	3.542%	67.9	76.4	78.8	69.8	78.9	73.7	57.3	75.2	72.3
	LoRA*	0.826%	68.9	80.7	77.4	78.1	78.8	77.8	61.3	74.8	74.7
	DoRA$_{\text{half}}$*	0.427%	70.0	82.6	79.7	83.2	80.6	80.6	65.4	77.6	77.5
	DoRA*	0.838%	68.5	82.9	79.6	84.8	80.8	81.4	65.8	81.0	78.1
	SMT‡	0.840%	68.7	81.7	78.3	91.6	78.8	84.1	67.7	77.4	78.7
	NeuroAda	**0.404%**	**73.1**	**85.4**	**80.9**	**94.3**	**84.3**	**87.8**	**71.7**	**84.2**	**82.7**
	PrefT*	0.039%	64.3	76.8	73.9	42.1	72.1	72.9	54.0	60.6	64.6
	DiReFT^{+}	0.031%	66.1	82.5	78.8	92.6	81.9	83.2	67.1	79.8	79.0
	LoReFT†	0.031%	68.3	83.5	79.7	**92.7**	**82.6**	83.2	67.4	78.5	79.5
	NeuroAda	**0.020%**	**69.6**	**83.6**	**80.5**	92.3	81.1	**84.0**	**68.1**	**80.4**	**80.0**
LLaMA (13B)	Series*	1.586%	71.8	83.0	79.2	88.1	82.4	82.5	67.3	81.8	79.5
	Parallel*	2.894%	72.5	84.9	79.8	92.1	84.7	84.2	71.2	82.4	81.5
	LoRA*	0.670%	72.1	83.5	80.5	90.5	83.7	82.8	68.3	82.4	80.5
	DoRA$_{\text{half}}$*	0.347%	72.5	85.3	79.9	90.1	82.9	82.7	69.7	83.6	80.8
	DoRA*	0.681%	72.4	84.9	81.5	92.4	84.2	84.2	69.6	82.8	81.5
	SMT‡	0.680%	71.1	84.4	81.7	93.7	83.2	86.7	73.7	85.2	82.4
	NeuroAda	**0.327%**	**73.3**	**87.9**	**82.7**	**96.0**	**86.9**	**90.2**	**77.1**	**88.6**	**85.3**
	PrefT*	0.031%	65.3	75.4	72.1	55.2	68.6	79.5	62.9	68.0	68.4
	DiReFT^{+}	0.025%	70.2	**86.6**	**82.5**	**95.0**	85.2	86.3	73.5	84.4	83.0
	LoReFT†	0.025%	72.0	85.6	82.1	94.8	**85.3**	86.9	73.0	85.0	83.1
	NeuroAda	**0.016%**	**73.0**	86.4	82.2	94.5	84.0	**87.6**	**74.5**	**86.0**	**83.5**
Llama2 (7B)	LoRA*	0.826%	69.8	79.9	79.5	83.6	82.6	79.8	64.7	81.0	77.6
	DoRA$_{\text{half}}$*	0.427%	72.0	83.1	79.9	89.1	83.0	84.5	71.0	81.2	80.5
	DoRA*	0.838%	71.8	83.7	76.0	89.1	82.6	83.7	68.2	82.4	79.7
	SMT‡	0.840%	72.0	83.8	80.8	93.3	82.8	86.7	74.0	81.0	81.8
	NeuroAda	**0.404%**	**73.6**	**86.5**	**81.1**	**94.8**	**87.8**	**89.1**	**75.9**	**85.6**	**84.3**
	DiReFT^{+}	0.031%	68.2	**83.4**	**79.8**	**93.4**	83.1	84.6	**70.3**	79.4	80.3
	LoReFT^{+}	0.031%	66.6	81.8	79.3	**93.4**	82.6	83.0	70.2	80.8	79.7
	NeuroAda	**0.020%**	**71.4**	82.8	**79.8**	93.3	**84.0**	**85.4**	70.1	**81.2**	**81.0**
Llama3 (8B)	LoRA*	0.700%	70.8	85.2	79.9	91.7	84.3	84.2	71.2	79.0	80.8
	DoRA$_{\text{half}}$*	0.361%	74.5	88.8	80.3	95.5	84.7	90.1	79.1	87.2	85.0
	DoRA*	0.710%	74.6	**89.3**	79.9	95.5	85.6	90.5	80.4	85.8	85.2
	SMT‡	0.710%	75.7	88.4	81.4	96.2	88.2	92.7	83.2	88.6	86.8
	NeuroAda	**0.343%**	**75.0**	**89.3**	**83.0**	**96.5**	**89.2**	**93.0**	**82.9**	**89.6**	**87.3**
	DiReFT^{+}	0.026%	73.0	89.8	81.4	96.1	87.8	92.3	79.9	85.4	85.7
	LoReFT^{+}	0.026%	72.9	89.1	81.7	96.1	88.0	92.0	80.1	85.0	85.6
	NeuroAda	**0.017%**	71.7	84.9	81.4	93.9	85.4	88.8	77.0	83.8	83.4

performance but also parameter efficiency. To demonstrate the robustness of our method under varying parameter budget constraints, we categorize all baseline methods into two groups based on the proportion of trainable parameters: those with $\geq 0.1\%$ and those with $< 0.1\%$. Note that these two groups differ by orders of magnitude in the number of trainable parameters, allowing us to assess performance across both relatively high and extremely low parameter budgets. We then compare our NeuroAda against these baselines under comparable levels of parameter efficiency to ensure a fair and meaningful evaluation. Specifically, we select the top-1 and top-20 input connections per neuron for matching the budget of the two groups, resulting in 0.016% and 0.327% trainable parameters on LLaMA-13B, respectively.

5.1 Commonsense reasoning

Following the experimental protocol of Hu et al. (2023b), we fine-tune LLaMA-7B/13B, LLaMA2-7B, and LLaMA3-8B on COMMONSENSE170K, a composite dataset consisting of eight commonsense reasoning tasks, as described in Appendix A. Then, We evaluate each task on its test set and compare our results with baselines from Hu et al. (2023b), as well as DoRA (Liu et al., 2024), DiReFT, LoReFT (Wu et al., 2024b), and SMT (He et al., 2024) under the same setting.

Hyperparameter tuning Inspired by Wu et al. (2024b), we use COMMONSENSE15K, a subset of COMMONSENSE170K, to perform hyperparameter search. The search space is detailed in Table 6. Specifically, we split COMMONSENSE15K into training and validation sets, as described in Section 4. Our hyperparameter search is conducted only on LLaMA-7B, and the best-performing configuration on the validation set is subsequently applied to all other models, including LLaMA-7B/13B, LLaMA2-7B, and LLaMA3-8B, for training on COMMONSENSE170K.

Results As shown in Table 2, our NeuroAda achieves state-of-the-art performance under both parameter budget regimes ($\geq 0.1\%$ and $< 0.1\%$). Notably, under the higher parameter budget setting, NeuroAda outperforms all baselines by a considerable margin. For example, its average accuracy surpasses the second-best baseline, SMT, by 4%. In addition, NeuroAda remains effective even under the lower parameter budget setting, consistently outperforming other baselines in this regime.

5.2 Arithmetic reasoning

Following Hu et al. (2023b) and Wu et al. (2024b), we fine-tune LLaMA-7B/13B, LLaMA2-7B, and LLaMA3-8B on MATH10K, a composite dataset comprising seven arithmetic reasoning tasks, and evaluate each task separately. The dataset details are provided in Appendix C.1.

Hyperparameter tuning Following Wu et al. (2024b), we perform hyperparameter search on the LLaMA-7B model using the GSM8K dataset, which is split into training and validation sets as in Wu et al. (2024b). The best-performing configuration on the validation set is then applied to all models, including LLaMA-7B/13B, LLaMA2-7B, and LLaMA3-8B—for training on the MATH10K dataset. The full hyperparameter search space is provided in Table 5 in Appendix.

Results As shown in Table 3, NeuroAda consistently achieves the highest average accuracy across all model sizes and parameter budgets. Under a higher parameter budget (e.g., 0.327% on LLaMA-13B), it outperforms all baselines by a clear margin. For example, NeuroAda outperforms the second-best baseline, LoRA, by 6%, while using even fewer trainable parameters. Even under extremely low budgets (e.g., 0.020% on LLaMA2-7B), NeuroAda remains competitive and surpassing other low-budget baselines by up to 6% with even fewer trainable parameters.

5.3 Natural language understanding

We evaluate the effectiveness of our method on the GLUE benchmark (Wang et al., 2018), a widely used suite of sequence classification tasks for evaluating natural language understanding (NLU), using the RoBERTa-base model. To ensure fair comparison, we follow the training, evaluation, and hyperparameter tuning procedures in Wu et al. (2024b).

Results We report the result in table 4 in the Appendix due to the space limitation. It shows NeuroAda achieves the highest average score across both moderate (0.2674%) and extreme low-budget (0.0297%) regimes. Compared to LoRA (0.239%), it improves the average GLUE score by +0.7. Under the extreme budget, it surpasses LoReFT by +0.8, RED by +0.7, and DiReFT by +1.8, despite using fewer parameters. Notably, NeuroAda achieves the best score on 6 out of 8 tasks in the low-budget setting, demonstrating its strong generalization even with minimal adaptation capacity.

Table 3: Performance comparison with existing PEFT methods on seven arithmetic reasoning datasets across four models: LLaMA-7B/13B, LLaMA2-7B, and LLaMA3-8B. *Most baseline results are taken from Hu et al. (2023a). †Results are from Wu et al. (2024b), ‡ results are taken from He et al. (2024) and *results are from Liu et al. (2024), as they share the same experimental setting with Hu et al. (2023a). For a fair comparison, our NeuroAda is also trained for 3 epochs to align with these baselines. $^{+}$When 3-epoch results are not available in the original paper, we re-trained the baselines using the official code and their reported best hyperparameters. All results for our method are averaged over three runs with different random seeds. Our method selects the top-20 and top-1 input connections per neuron for high-budget and low-budget parameter groups, respectively.

Model	**PEFT**	**Params (%)**	**Accuracy (↑)**							
			Arithmetic Reasoning							
			MulAri	**GSM8K**	**AddSub**	**AQuA**	**SinEq**	**SVAMP**	**MAWPS**	**Avg.**
GPT-3.5$_{175B}$*	–	–	83.8	56.4	85.3	38.9	88.1	69.9	–	70.4
LLaMA (7B)	Series*	1.953%	92.8	33.3	80.0	15.0	83.5	52.3	–	59.5
	Parallel*	3.542%	94.5	35.3	86.6	18.1	86.0	49.6	–	61.7
	LoRA*	0.826%	95.0	**37.5**	83.3	18.9	84.4	52.1	–	61.9
	SMT‡	0.860%	91.5	34.2	85.8	23.6	84.6	53.6	–	62.2
	SMT‡	1.260%	93.4	35.6	86.8	24.2	85.3	54.8	–	63.4
	NeuroAda	**0.404%**	**96.0**	36.5	**92.4**	**22.0**	**94.1**	**53.2**	–	**68.4**
	PrefT*	0.039%	–	24.4	–	14.2	–	38.1	63.4	35.0
	DiReFT^{+}	0.031%	–	20.5	–	21.3	–	39.9	68.1	37.5
	LoReFT†	0.031%	–	21.6	–	22.4	–	43.6	69.5	39.3
	NeuroAda	**0.020%**	–	**30.3**	–	**22.8**	–	**48.9**	**77.7**	**44.9**
LLaMA (13B)	Series*	1.586%	93.0	44.0	80.5	22.0	87.6	50.8	–	63.0
	Parallel*	2.894%	94.3	43.3	83.0	20.5	89.6	55.7	–	64.4
	LoRA*	0.670%	94.8	**47.5**	87.3	18.5	89.8	54.6	–	65.4
	NeuroAda	**0.327%**	**97.5**	43.9	**92.2**	**21.7**	**93.9**	**61.4**	–	**71.4**
	PrefT*	0.031%	–	31.1	–	15.7	–	41.4	66.8	38.8
	DiReFT^{+}	0.025%	–	32.1	–	23.2	–	51.2	76.1	46.7
	LoReFT†	0.025%	–	35.5	–	23.4	–	54.6	81.8	48.8
	NeuroAda	**0.016%**	–	**43.0**	–	**25.6**	–	**56.7**	**83.6**	**52.2**
LLaMA2 (7B)	DiReFT^{+}	0.031%	–	26.4	–	**23.6**	–	48.4	71.8	42.6
	LoReFT^{+}	0.031%	–	26.2	–	18.5	–	46.7	76.9	42.1
	NeuroAda	**0.020%**	–	**36.1**	–	22.8	–	**52.1**	**82.4**	**48.4**
LLaMA3 (8B)	DiReFT^{+}	0.026%	–	57.2	–	**30.1**	–	68.6	87.8	60.9
	LoReFT^{+}	0.026%	–	56.9	–	24.8	–	70.9	88.2	60.2
	NeuroAda	**0.017%**	–	**63.7**	–	26.4	–	**75.0**	**88.7**	**63.5**

6 Conclusion

This paper introduced NeuroAda, a featherlight and scalable fine-tuning framework that activates each neuron's potential through top-k magnitude-based weight selection. By inheriting the performance benefits of sparse tuning and the memory efficiency of addition-based methods, NeuroAda avoids structural modifications, runtime masking, and full-gradient computation. Its static selection of high-magnitude weights per neuron enables task-agnostic, fine-grained adaptation with significantly reduced memory and computational overhead. Empirical results across diverse reasoning and language understanding tasks show that NeuroAda surpasses strong adaptation baselines, achieving robust generalization under an extremely few number of trainable parameters and tight memory budgets.

7 Limitations

While NeuroAda demonstrates strong empirical performance across diverse tasks and architectures, our current evaluation is limited to models up to 13 billion parameters. We anticipate that the benefits of our method may further amplify at larger scales, but assessing its efficacy on models beyond 13B remains an important direction for future work. Evaluating scalability and stability under extreme model sizes is critical for deployment in real-world, high-capacity systems. Also, we expect future research to verify NeuroAda on VLM models with vision-related tasks (Dong et al., 2025a).

References

Alan Ansell, Ivan Vulić, Hannah Sterz, Anna Korhonen, and Edoardo M Ponti. 2024. Scaling sparse fine-tuning to large language models. *arXiv preprint arXiv:2401.16405*.

Matan Avitan, Ryan Cotterell, Yoav Goldberg, and Shauli Ravfogel. 2024. What changed? converting representational interventions to natural language. *arXiv e-prints*, pages arXiv–2402.

Nora Belrose, David Schneider-Joseph, Shauli Ravfogel, Ryan Cotterell, Edward Raff, and Stella Biderman. 2023. Leace: Perfect linear concept erasure in closed form. *Advances in Neural Information Processing Systems*, 36:66044–66063.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022a. BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, Dublin, Ireland. Association for Computational Linguistics.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022b. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, Dublin, Ireland. Association for Computational Linguistics.

Kartikeya Bhardwaj, Nilesh Pandey, Sweta Priyadarshi, Viswanath Ganapathy, Shreya Kadambi, Rafael Esteves, Shubhankar Borse, Paul Whatmough, Risheek Garrepalli, Mart van Baalen, et al. 2024. Sparse high rank adapters. *Advances in Neural Information Processing Systems*, 37:13685–13715.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7432–7439.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. 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)*, pages 2924–2936, Minneapolis, Minnesota. Association for Computational Linguistics.

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:1803.05457*.

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

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36.

Kuicai Dong, Yujing Chang, Xin Deik Goh, Dexun Li, Ruiming Tang, and Yong Liu. 2025a. Mmdocir: Benchmarking multi-modal retrieval for long documents.

Kuicai Dong, Yujing Chang, Shijie Huang, Yasheng Wang, Ruiming Tang, and Yong Liu. 2025b. Benchmarking retrieval-augmented multimomal generation for document question answering.

Kuicai Dong, Aixin Sun, Jung-jae Kim, and Xiaoli Li. 2023. Open information extraction via chunks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15390–15404, Singapore. Association for Computational Linguistics.

Zihao Fu, Haoran Yang, Anthony Man-Cho So, Wai Lam, Lidong Bing, and Nigel Collier. 2023. On the effectiveness of parameter-efficient fine-tuning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 12799–12807.

Haoyu He, Jianfei Cai, Jing Zhang, Dacheng Tao, and Bohan Zhuang. 2023. Sensitivity-Aware Visual Parameter-Efficient Tuning. ArXiv:2303.08566 [cs].

Haoze He, Juncheng Billy Li, Xuan Jiang, and Heather Miller. 2024. Sparse Matrix in Large Language Model Fine-tuning. ArXiv:2405.15525 [cs].

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

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–533, Doha, Qatar. Association for Computational Linguistics.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022a. Lora: Low-rank adaptation of large language models. *International Conference on Learning Representations*.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022b. LoRA: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022*, Virtual Event.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Lee. 2023a. 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, Singapore. Association for Computational Linguistics.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Lee. 2023b. 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, Singapore. Association for Computational Linguistics.

Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. 2022. Visual Prompt Tuning. ArXiv:2203.12119 [cs].

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597.

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, San Diego, California. Association for Computational Linguistics.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki Markus Asano. 2023. Vera: Vector-based random matrix adaptation. *arXiv preprint arXiv:2310.11454*.

Hector Levesque, Ernest Davis, and Leora Morgenstern. 2012. The Winograd Schema Challenge. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*.

Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. 2023. Inference-time intervention: Eliciting truthful answers from a language model. *Advances in Neural Information Processing Systems*, 36:41451–41530.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

Xiangyang Li, Kuicai Dong, Yi Quan Lee, Wei Xia, Hao Zhang, Xinyi Dai, Yasheng Wang, and Ruiming Tang. 2025. CoIR: A comprehensive benchmark for code information retrieval models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 22074–22091, Vienna, Austria. Association for Computational Linguistics.

Baohao Liao, Yan Meng, and Christof Monz. 2023a. Parameter-efficient fine-tuning without introducing new latency. *arXiv preprint arXiv:2305.16742*.

Baohao Liao, Shaomu Tan, and Christof Monz. 2023b. Make pre-trained model reversible: From parameter to memory efficient fine-tuning. *Advances in Neural Information Processing Systems*, 36.

Zhaojiang Lin, Andrea Madotto, and Pascale Fung. 2020. Exploring versatile generative language model via parameter-efficient transfer learning. *arXiv preprint arXiv:2004.03829*.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv:1705.04146*.

Sheng Liu, Haotian Ye, Lei Xing, and James Zou. 2023. In-context vectors: Making in context learning more effective and controllable through latent space steering. *arXiv preprint arXiv:2311.06668*.

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. *arXiv:2402.09353*.

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

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. *arXiv:1809.02789*.

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, Online. Association for Computational Linguistics.

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

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1743–1752, Lisbon, Portugal. Association for Computational Linguistics.

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.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. 2019. Social IQa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4463–4473, Hong Kong, China. Association for Computational Linguistics.

Shufan Shen, Junshu Sun, Xiangyang Ji, Qingming Huang, and Shuhui Wang. 2024. Expanding sparse tuning for low memory usage. *arXiv preprint arXiv:2411.01800*.

Shashwat Singh, Shauli Ravfogel, Jonathan Herzig, Roee Aharoni, Ryan Cotterell, and Ponnurangam Kumaraguru. 2024. MiMiC: Minimally modified counterfactuals in the representation space. *arXiv:2402.09631*.

Weixi Song, Zuchao Li, Lefei Zhang, Hai Zhao, and Bo Du. 2023. Sparse is enough in fine-tuning pre-trained large language models. *arXiv preprint arXiv:2312.11875*.

Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. 2022. Vl-adapter: Parameter-efficient transfer learning for vision-and-language tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5227–5237.

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models.

Raja Vavekanand and Kira Sam. 2024. Llama 3.1: An in-depth analysis of the next-generation large language model. In -.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Glue: A multi-task benchmark and analysis platform for natural language understanding.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Muling Wu, Wenhao Liu, Xiaohua Wang, Tianlong Li, Changze Lv, Zixuan Ling, Jianhao Zhu, Cenyuan Zhang, Xiaoqing Zheng, and Xuanjing Huang. 2024a. Advancing parameter efficiency in fine-tuning via representation editing. *arXiv:2402.15179*.

Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D. Manning, and Christopher Potts. 2024b. ReFT: Representation Finetuning for Language Models. ArXiv:2404.03592.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? *arXiv:1905.07830*.

Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. 2022. Scaling vision transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12104–12113.

Q Zhang, M Chen, A Bukharin, P He, Y Cheng, W Chen, and T Zhao. 2023. Adaptive budget allocation for parameter-efficient fine-tuning. preprint (2023). *arXiv preprint arXiv:2303.10512*.

Renrui Zhang, Rongyao Fang, Wei Zhang, Peng Gao, Kunchang Li, Jifeng Dai, Yu Qiao, and Hongsheng Li. 2021. Tip-adapter: Training-free clip-adapter for better vision-language modeling. *arXiv preprint arXiv:2111.03930*.

Zhi Zhang, Jiayi Shen, Congfeng Cao, Gaole Dai, Shiji Zhou, Qizhe Zhang, Shanghang Zhang, and Ekaterina Shutova. 2024a. Proactive gradient conflict mitigation in multi-task learning: A sparse training perspective. *arXiv preprint arXiv:2411.18615*.

Zhi Zhang, Qizhe Zhang, Zijun Gao, Renrui Zhang, Ekaterina Shutova, Shiji Zhou, and Shanghang Zhang. 2024b. Gradient-based parameter selection for efficient fine-tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 28566–28577.

A Datasets

Our experiments primarily target both natural language understanding (NLU) and natural language generation (NLG) tasks. Specifically, we evaluate commonsense reasoningand arithmetic reasoning for NLG task and GLUE for NLU task.

A.1 Commonsense Reasoning

We evaluate the our method on eight widely-used datasets that span various forms of open-ended question answering:

- **BoolQ**(Clark et al., 2019): a yes/no question-answering dataset composed of naturally occurring questions. Following prior work, we remove the associated passages to focus solely on the question context.
- **PIQA**(Bisk et al., 2020): designed to test physical commonsense, this dataset requires selecting the most plausible action in a given hypothetical situation.
- **SIQA**(Sap et al., 2019): targets social commonsense by asking models to reason about human actions and their social consequences.
- **HellaSwag**(Zellers et al., 2019): involves selecting the most coherent sentence completion given a narrative context, emphasizing grounded commonsense inference.
- **WinoGrande**(Sakaguchi et al., 2021), inspired by the Winograd Schema Challenge(Levesque et al., 2012), tests pronoun resolution in context, requiring fine-grained commonsense reasoning.
- **ARC-Easy (ARC-e)**(Clark et al., 2018): a benchmark of multiple-choice elementary science questions with relatively straightforward reasoning.
- **ARC-Challenge (ARC-c)**(Clark et al., 2018): a more difficult subset of ARC designed to be resistant to simple co-occurrence-based solutions.
- **OpenBookQA (OBQA)** (Mihaylov et al., 2018): a knowledge-intensive QA dataset requiring multi-hop reasoning across both textual context and external knowledge.

We adopt the same experimental protocol as described in Hu et al. (2023b), aggregating the training sets of the above datasets into a unified corpus referred to as COMMONSENSE170K. Fine-tuning is conducted on this joint dataset, and evaluation is performed on the individual test sets of each benchmark. Detailed dataset statistics and simplified training examples are also available in Hu et al. (2023b).

A.2 Arithmetic reasoning

We train and evaluate our method using seven benchmark datasets that span a diverse range of mathematical word problem domains:

- **AddSub**(Hosseini et al., 2014), a dataset composed of elementary arithmetic problems involving addition and subtraction.
- **AQuA**(Ling et al., 2017), which presents algebraic word problems in a multiple-choice format.
- **GSM8K**(Cobbe et al., 2021), consisting of grade-school math problems that require multi-step reasoning.
- **MAWPS**(Koncel-Kedziorski et al., 2016), a collection of math word problems with varied linguistic and arithmetic complexity.
- **MultiArith**(Roy and Roth, 2015), featuring problems that demand reasoning through multiple arithmetic steps.
- **SingleEq**(Koncel-Kedziorski et al., 2015), which includes math problems that can be solved by formulating a single equation of varying complexity.
- **SVAMP** (Patel et al., 2021), designed to test a model's robustness to structural variations in math problem formulations by rephrasing original problems in a challenging way.
- **MAWPS** (Koncel-Kedziorski et al., 2016) is a collection of math word problems of varying complexity, involving basic arithmetic operations such as addition, subtraction, multiplication, and division. Each instance is annotated with both the natural language problem and its corresponding symbolic equation, facilitating studies in semantic parsing and numerical reasoning.

We follow the experimental design of Hu et al. (2023b), which also provides dataset statistics and representative examples for each benchmark. Our training is conducted on a unified dataset—MATH10K—which comprises training samples from four datasets: GSM8K, MAWPS, MAWPS-single, and AQuA. Following (Wu et al., 2024b), we compare our method with LoReFT and DiReFT on four tasks: GSM8K, MAWPS, SVAMP and AQuA and with other baselines on all above tasks except for MAWPS Hu et al. (2023b).

A.3 Natural language understanding

Following the evaluation protocol established by Wu et al. (2024a), we ensure a fair assessment on the GLUE validation set by partitioning it into two subsets. One subset, determined using a fixed random seed, is reserved for in-training evaluation, while the other is used exclusively for final testing. After each training epoch, we evaluate the model on the in-training subset and select the checkpoint with the best performance across all epochs for testing. For datasets with relatively large validation sets (i.e., QQP, MNLI, and QNLI), we randomly sample 1,000 instances for in-training evaluation. For smaller datasets, we use 50% of the validation data for this purpose. As for the evaluation metrics, we adopt the Matthews correlation coefficient for CoLA, the Pearson correlation coefficient for STS-B, and classification accuracy for the remaining tasks. For MNLI, we report results on the matched subset only.

B Results on natural language understanding tasks

we evaluate our NeuroAda on the GLUE for natural language understanding tasks. The results are shown in table 4.

C Hyperparameters

C.1 Hyperparameter tuning and decoding strategy

Arithmeric reasoning Following Wu et al. (2024b), we adopt their training and validation splits of the GSM8K dataset. Models are trained on the training set, and hyperparameters are selected based on performance on the validation set. All hyperparameters are tuned using LLaMA-7B, and the resulting configuration is directly applied to LLaMA-13B, LLaMA2-7B, and LLaMA3-8B without additional tuning. We use a maximum sequence length of 512 tokens during training and hyperparameter tuning, and generate up to 32 new tokens during inference. Table 5 summarizes the full hyperparameter search space.

Dataset MATH10K is annotated with language model-generated chain-of-thought reasoning steps. In the tasks, models are required to generate a chain of thought (Wei et al., 2022) before producing the final answer. The included tasks are AddSub (Hosseini et al., 2014), AQuA (Ling et al., 2017), GSM8K (Cobbe et al., 2021), MAWPS (Koncel-Kedziorski et al., 2016), MultiArith (Roy and Roth, 2015), SingleEq (Koncel-Kedziorski et al., 2015), and SVAMP (Patel et al., 2021). See Appendix A.2 for detailed descriptions of each task. For fair comparison, we follow both evaluation settings used in prior work. Specifically, Wu et al. (2024b) report results on GSM8K, AQuA, SVAMP, and MAWPS, while Hu et al. (2023b) evaluate on MultiArith, GSM8K, AddSub, AQuA, SingleEq, and SVAMP. We compare our method with theirs under both settings to ensure a comprehensive and fair evaluation.

Commonsense reasoning Motivated by Wu et al. (2024b), we perform hyperparameter tuning on COMMONSENSE15K, a subset of the full COMMONSENSE170K benchmark. Details of the search space are provided in Table 6. We divide COMMONSENSE15K into training and validation splits using a ratio 7:3. Hyperparameter tuning is conducted exclusively on LLaMA-7B, and the optimal configuration identified on the validation set is reused for all other models, including LLaMA-7B/13B, LLaMA2-7B, and LLaMA3-8B, during training on the full COMMONSENSE170K dataset.

For the commonsense reasoning benchmark, we adopt greedy decoding without sampling, as the task requires multi-token classification. In contrast, for the arithmetic reasoning benchmark, we follow the decoding setup used by Hu et al. (2023b), employing a higher decoding temperature of 0.3. This change is made to avoid instability issues caused by near-zero token probabilities, which can lead to decoding errors in the `HuggingFace` implementation when using beam search.

Dataset Our comparative experiments are primarily conducted on LLaMA-7B, using two lightweight reasoning datasets to enable rapid evaluation and comparison: COMMONSENSE15K for

Table 4: Performance comparison with existing PEFT methods on RoBERTa-base for the GLUE benchmark. *Most baseline results are taken from Wu et al. (2024a). The result is presented as the mean with standard deviation (SD) over five runs with different random seeds. †Results are from Wu et al. (2024b) as it shares the same experimental setting with Wu et al. (2024a). For a fair comparison, our NeuroAda also follows the same setting.

Model	PEFT	Params (%)	Accuracy (↑) (SD)								
			MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoBERTa Base	FT	100%	87.3$_{(0.34)}$	94.4$_{(0.96)}$	87.9$_{(0.91)}$	62.4$_{(3.29)}$	92.5$_{(0.22)}$	91.7$_{(0.19)}$	78.3$_{(3.20)}$	90.6$_{(0.59)}$	85.6
	Adapter*	0.318%	87.0$_{(0.28)}$	93.3$_{(0.40)}$	88.4$_{(1.54)}$	**60.9**$_{(3.09)}$	92.5$_{(0.02)}$	**90.5**$_{(0.08)}$	76.5$_{(2.26)}$	90.5$_{(0.35)}$	85.0
	LoRA*	0.239%	86.6$_{(0.23)}$	93.9$_{(0.49)}$	88.7$_{(0.76)}$	59.7$_{(4.36)}$	**92.6**$_{(0.10)}$	90.4$_{(0.08)}$	75.3$_{(2.79)}$	90.3$_{(0.54)}$	84.7
	Adapter$^{\text{FNN}*}$	0.239%	87.1$_{(0.10)}$	93.0$_{(0.05)}$	88.8$_{(1.38)}$	58.5$_{(1.69)}$	92.0$_{(0.28)}$	90.2$_{(0.07)}$	77.7$_{(1.93)}$	90.4$_{(0.31)}$	84.7
	NeuroAda	0.2674%	**87.7**$_{(0.12)}$	**94.3**$_{(0.10)}$	**89.7**$_{(0.13)}$	56.1$_{(0.09)}$	92.0$_{(0.18)}$	90.2$_{(0.23)}$	**82.0**$_{(0.08)}$	**91.0**$_{(0.23)}$	**85.4**
	BitFit*	0.080%	84.7$_{(0.08)}$	94.0$_{(0.87)}$	88.1$_{(1.57)}$	54.0$_{(3.07)}$	91.0$_{(0.05)}$	87.3$_{(0.02)}$	69.8$_{(1.51)}$	89.5$_{(0.35)}$	82.3
	RED*	0.016%	83.9$_{(0.14)}$	93.9$_{(0.31)}$	89.2$_{(0.98)}$	**61.0**$_{(2.96)}$	90.7$_{(0.35)}$	87.2$_{(0.17)}$	78.0$_{(2.06)}$	90.4$_{(0.32)}$	84.3
	DiReFT †	0.015%	82.5$_{(0.22)}$	92.6$_{(0.76)}$	88.3$_{(1.23)}$	58.6$_{(1.99)}$	91.3$_{(0.19)}$	86.4$_{(0.27)}$	76.4$_{(1.48)}$	89.3$_{(0.56)}$	83.2
	LoReFT †	0.015%	83.1$_{(0.26)}$	93.4$_{(0.64)}$	89.2$_{(2.62)}$	60.4$_{(2.60)}$	91.2$_{(0.25)}$	87.4$_{(0.23)}$	79.0$_{(2.76)}$	90.0$_{(0.29)}$	84.2
	NeuroAda	0.0297%	**85.1**$_{(0.09)}$	**94.3**$_{(0.16)}$	**90.7**$_{(0.07)}$	59.8$_{(0.12)}$	**92.1**$_{(0.12)}$	**88.1**$_{(0.24)}$	**79.1**$_{(0.23)}$	**90.7**$_{(0.10)}$	**85.0**

Table 5: Hyperparameter search space for the LLaMA-7B model using our NeuroAda on the validation set of the GSM8K. The best-performing settings are $\underline{\text{underlined}}$ for selecting top-1 and $\utilde{\text{wavy underline}}$ for selecting top-20 parameters per neuron in the model. Greedy decoding (without sampling) is used throughout the hyperparameter tuning process.

Hyperparameters (LLaMA-7B on GSM8K)	
Optimizer	AdamW
LR	$\{6\times10^{-4}, 9\times10^{-4}, 1\times10^{-3}, \utilde{3\times10^{-3}}, 6\times10^{-3}, \underline{9\times10^{-3}}, 1\times10^{-2}, 3\times10^{-2}\}$
Weight decay	{0}
LR scheduler	Linear
Batch size	$\{\utilde{8}, \underline{16}, 32\}$
Warmup ratio	$\{0.00, \underline{0.06}, \utilde{0.10}\}$
Epochs	{3}
Top-k	$\{\underline{1}, \utilde{20}\}$

commonsense reasoning and GSM8K for arithmetic reasoning. COMMONSENSE15K is a subset of the larger COMMONSENSE170K dataset, originally partitioned by Hu et al. (2023b). (1) COMMONSENSE170K comprises eight commonsense reasoning tasks, including BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-e, ARC-c (Clark et al., 2018), and OBQA (Mihaylov et al., 2018), as detailed in Appendix A.1. All examples are presented as multiple-choice questions, where the model is required to directly output the correct option without generating intermediate rationales. We adopt the prompt template from Hu et al. (2023b), with an additional string normalization step (removal of leading and trailing whitespace). We split COMMONSENSE15K into training and validation sets using a 7:3 ratio for our experiments. (2) GSM8K(Cobbe et al., 2021) dataset comprises grade-school-level arithmetic word problems that require multi-step reasoning to arrive at the correct answer. In contrast to COMMONSENSE15K, solving GSM8K typically involves generating a chain-of-thought (Wei et al., 2022) prior to producing the final answer. We adopt the same prompt template as used in Hu et al. (2023b).

Natural language understanding. Following Wu et al. (2024b), we perform hyperparameter tuning for each GLUE task individually using both RoBERTa-base. Hyperparameters are selected based on performance on a held-out validation set with a fixed random seed of 42. To obtain the final results, we evaluate the models using four addi-

Table 6: Hyperparameter search space for the LLaMA-7B model using our NeuroAda on the validation set of the COMMONSENSE15K. The best-performing settings are underlined for selecting top-1 and wavy underline for selecting top-20 parameters per neuron in the model. Greedy decoding (without sampling) is used throughout the hyperparameter tuning process.

Hyperparameters (LLaMA-7B on COMMONSENSE15K)	
Optimizer	AdamW
LR	$\{\uwave{7\times10^{-4}}, 9\times10^{-4}, 2\times10^{-3}, 4\times10^{-3}, 6\times10^{-3}, \underline{8\times10^{-3}}, 1\times10^{-2}, 2\times10^{-2}\}$
Weight decay	$\{0\}$
LR scheduler	Linear
Batch size	$\{\underline{8}, \uwave{16}, 32\}$
Warmup ratio	$\{0.00, \uwave{0.06}, \underline{0.10}\}$
Epochs	$\{3\}$
Top-k	$\{\underline{1}, \uwave{20}\}$

tional unseen seeds: 43, 44, 45, 46. We adopt the evaluation protocol of Wu et al. (2024a). For QQP with RoBERTa-large, due to observed stochasticity across repeated runs with the same seed, we report the best result from three trials for each seed. As noted by Wu et al. (2024a), the evaluation results on RTE are unstable due to the dataset's small size. We follow their approach and adjust the set of random seeds accordingly to ensure fair comparison. Additionally, we replace one or two random seeds for CoLA to improve evaluation stability.

Preliminary Analysis We compare the mask-based method—which applies binary masks to zero out the gradients of unselected (frozen) parameters—with the addition-based method, which introduces new trainable parameters to bypass the selected ones, rather than tuning them directly, for the purpose of sparse fine-tuning. This comparison is conducted across three dimensions: effectiveness, GPU memory usage, and training efficiency. We further investigate the effectiveness of the proposed addition-based method, NeuroAda, which is designed to ensure that all neurons in the network retain the potential to update their activation states during fine-tuning. To this end, we analyze the proportion of neurons actively involved in the tuning process and evaluate various parameter selection strategies for determining which neurons are activated.

To ensure a fair comparison between the addition-based and mask-based approaches, as well as across different parameter selection strategies, we conduct a hyperparameter search over a range of learning rates for each experiment using the training set. The optimal configuration is selected based on performance on the validation set. This step is essential, as PEFT methods are generally sensitive to the choice of learning rate (Wu et al., 2024b). The complete hyperparameter search space is provided in Table 7.

D Advantages of NeuroAda

NeuroAda significantly reduces the backward computation costs with four core advantages. (1) It achieves **mask-free sparsity** by statically selecting top-k weights per neuron and storing only a compact set of BF16 deltas and integer indices. This avoids the memory and compute overhead associated with dynamic binary masks commonly used in gradient-based sparse methods, as quantified in Table 1. (2) It requires **no warm-up or task-specific signal**: the magnitude-based selection operates entirely offline and consistently across tasks, eliminating the need for gradient accumulation or adaptive heuristics. (3) It ensures **neuron-level coverage** by allocating trainable updates to every row of the weight matrix. This guarantees that all neurons participate in learning and avoids the dead filter problem often observed in block- or layer-wise pruning. (4) It introduces **no inference-time overhead**: the sparse update tensor $\mathbf{\Delta}$ is merged into the original weights post-training, preserving the model's structure, runtime efficiency, and compatibility with standard inference stacks.

Table 7: Hyperparameter search space for the LLaMA-7B model using our NeuroAda on the validation set of the GSM8K and COMMONSENSE15K for different number of trainable parameters. Greedy decoding (without sampling) is used throughout the hyperparameter tuning process.

Hyperparameters on LLaMA-7B for different number of trainable parameters	
Optimizer	AdamW
LR	$\{6\times10^{-5}, 9\times10^{-5}, 5\times10^{-4}, 7\times10^{-4}, 9\times10^{-4}, 3\times10^{-3}, 6\times10^{-3}, 9\times10^{-3}, 1\times10^{-2}\}$
Weight decay	{0}
LR scheduler	Linear
Batch size	{16}
Warmup ratio	{0.06}
Epochs	{3}
Top-k	{1 5 10 20 50 100 300 500}

E Algorithm

NeuroAda follows a three-phase procedure designed for efficiency, simplicity, and compatibility with standard inference infrastructure. As shown in Algorithm 1, the process begins with an offline selection phase, where the top-k highest-magnitude weights are identified per neuron based on the pretrained weight matrix. This static selection removes the need for gradient-based importance scoring or dynamic masking during training.

During training, only the selected top-k coordinates per neuron are updated, with all other parameters kept frozen. This enables mask-free, neuron-wise sparse adaptation that significantly reduces gradient computation and optimizer state memory. Crucially, NeuroAda performs updates directly over a small number of stored deltas, requiring no structural changes or auxiliary layers.

After training, the sparse deltas are merged into the frozen weights via an in-place update, resulting in a model that retains its original structure and supports efficient, standard inference. This design ensures minimal runtime overhead while offering strong adaptation capabilities through fine-grained parameter selection.

Algorithm 1: NeuroAda: Sparse top-k neuron-wise adaptation with merge-compatible updates.

```
Input: pretrained weight matrix $\mathbf{\Phi} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, top-$k$
       budget $k \ll d_{\text{in}}$, training mini–batches
       $\{(\mathbf{x}, \mathbf{y})\}$, learning-rate $\eta$
Output: adapted model with merged weights $\mathbf{\Phi} + \mathbf{\Delta}$
Phase 1: Offline Top-$k$ Selection;
foreach $i = 1, \dots, d_{out}$ do        // row $\equiv$ neuron
    // store $k$ index positions
    $\mathcal{I}_i \leftarrow \mathrm{TopK}\big(|\mathbf{\Phi}_{i,:}|, k\big)$
end
Phase 2: Sparse Training (mask-free);
For each neuron $i$, initialize $\mathbf{\Delta}_{i,\mathcal{I}_i} \leftarrow 0$;
foreach mini-batch $(\mathbf{x}, \mathbf{y})$ do
    // forward pass;
    $\mathbf{h} \leftarrow (\mathbf{\Phi} + \mathbf{\Delta})\,\mathbf{x}$;
    compute loss $\mathcal{L}(\mathbf{h}, \mathbf{y})$;
    // backward: update only selected
      entries;
    foreach $i = 1, \dots, d_{out}$ do
        $g_i \leftarrow \nabla_{\mathbf{\Delta}_{i,\mathcal{I}_i}} \mathcal{L}$;
        $\mathbf{\Delta}_{i,\mathcal{I}_i} \mathrel{-}= \eta\, g_i$
    end
end
Phase 3: One-shot Merge and Inference;
foreach $i = 1, \dots, d_{out}$ do
    $\mathbf{\Phi}_{i,\mathcal{I}_i} \leftarrow \mathbf{\Phi}_{i,\mathcal{I}_i} + \mathbf{\Delta}_{i,\mathcal{I}_i}$;
end
Delete $\mathbf{\Delta}$ from memory.;
# PyTorch (illustrative)
y = F.linear(x, merged_weight, bias)
```