BoLA: Block-wise Lottery Ticket Adaptation for Large Language Models

Anonymous ACL submission

Abstract

We propose **Bl**ock-wise **L**ottery Ticket **A**daptation (BoLA), a novel and simple sparse fine-tuning framework designed to enhance parameter efficiency in adapting large language models (LLMs) to new domains. Unlike conventional parameter-efficient fine-tuning (PEFT) methods such as LoRA and DoRA, which rely on dense adaptation, BoLA introduces a block-wise sparse selection mechanism. This mechanism searches for and updates only a subset of the parameters that are relevant for domain-specific learning. By integrating lottery ticket-style search with block-level granularity, BoLA mitigates catastrophic forgetting and enables interpretable, efficient adaptation while remaining compatible with existing PEFT techniques. Experiments on the math and commonsense reasoning benchmark demonstrate that BoLA achieves competitive performance with LoRA and DoRA. Our experiment code is available at `https://anonymous.4open.science/r/peft-B728`.

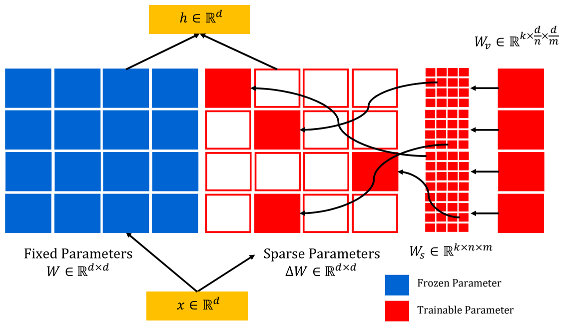

Figure 1: Overview of the proposed BoLA, which constructs trainable sparse weights $\Delta W \in \mathbb{R}^{d \times d}$ for fine-tuning, where W_s are block-wise score weights, and W_v are block-wise value weights. Blue indicates frozen parameters, red indicates trainable parameters, and white indicates zero parameters.

1 Introduction

With the advent of large language models (LLMs), it is no longer necessary to train separate models for each individual NLP task. Instead, a single general-purpose model can perform a wide range of tasks simply by providing suitable instructions. It is well-established that the performance of these models follows empirical scaling laws: as the number of parameters increases, model quality improves according to a power-law relationship (Kaplan et al., 2020). However, training such LLMs requires massive computational resources, imposing significant demands in terms of hardware and energy consumption. For example, if the number of trainable parameters is ϕ, the model state is 16 bits, and the optimizer state is 32 bits, then $16 \times \phi$ bytes of computing resources are required (Suhoi, 2024). Consequently, training an LLM with 8B parameters typically requires approximately 128 GB or more of GPU memory. This substantial memory footprint poses a challenge for researchers and engineers attempting to fine-tune such models for specific domains or tasks.

To address this issue, parameter-efficient fine-tuning (PEFT) methods (Houlsby et al., 2019) have been proposed, that fine-tune pre-trained models using only a minimal number of trainable parameters. Among these, reparameterized PEFT approaches such as LoRA (Hu et al., 2022) and DoRA (Liu et al., 2024) employ low-rank decomposition to reduce the number of trainable parameters. Moreover, since these methods do not alter the model architecture, they have gained significant popularity due to their comparable performance to full fine-tuning but with reduced GPU memory requirements. Notably, QLoRA (Dettmers et al., 2023) enables the fine-tuning of 8B-scale LLMs using as little as 24 GB of GPU memory, depending on the precision and optimization techniques employed. These advancements significantly improve the accessibility and efficiency of adapting large-scale models to a wide range of domains and tasks.

Since PEFT methods are dense methods that update all parameters of the target module, they are prone to catastrophic forgetting (Ramasesh et al., 2022; Dong et al., 2024; Luo et al., 2025), a phenomenon in which the pre-trained knowledge of the original model is overwritten. However, it has been found that sparse methods (Panda et al., 2024; Xu and Zhang, 2024; Rios et al., 2025) also work well. Sparse methods offer several features over dense methods:

- They update only a specified subset of parameters, minimizing contamination of the original weights and helping to avoid catastrophic forgetting.
- They provide interpretability by identifying which parameters encode knowledge relevant to the target task or domain.

Sparse methods typically adapt models through a two-stage pipeline consisting of first searching for the optimal parameter subset, and then fine-tuning those parameters. However, this search phase introduces additional computational overhead, increasing the complexity of implementation.

To tackle this problem, we propose **Blo**ck-wise **L**ottery Ticket **A**daptation (BoLA) as shown in Figure1, which is designed to simultaneously search for and update block-wise target parameters, inspired by Lottery Ticket Hypothesis (LTH) (Frankle and Carbin, 2019; Ramanujan et al., 2020). We evaluate BoLA across several language models and domains, demonstrating the relationship between training domains and training parameters through score maps. The experimental results show that BoLA is highly compatible with LoRA and DoRA, and can serve as a substitute for them despite being a sparse method. The sparsity of BoLA minimizes contamination of the original weights and helps to avoid catastrophic forgetting. Furthermore, by leveraging score maps, we show that the number of trainable parameters required tends to increase as the domain data becomes more distant from the distribution trained by the pretrained model. The summary of our contributions are as follows:

- We introduce BoLA as a new and simple sparse-PEFT method capable of mitigating catastrophic forgetting during model merging, which is difficult to achieve with dense-PEFT methods such as LoRA and DoRA.
- We demonstrate that BoLA can provide model interpretability by making it easier to identify where domain-specific knowledge is acquired.
- We demonstrate that BoLA is highly compatible with LoRA and DoRA and can be used as a substitute for them.

2 Related Works

Lottery Ticket Hypothesis (LHT) (Frankle and Carbin, 2019) posits that within a randomly initialized neural network, there exist subnetworks that, when trained in isolation, can achieve test accuracy comparable to that of the original network. This hypothesis enables pruning pretrained models to arbitrary sizes. These subnetworks, called winning tickets, are identified by examining the magnitude of the trained dense weights to prune unimportant connections (Frankle and Carbin, 2019; Ramanujan et al., 2020). Several more-efficient methods have been proposed to identify winning tickets earlier and more effectively during the dense training phase (Chen et al., 2022; Yuan et al., 2025). Methods that perform training and pruning simultaneously have also been proposed (You et al., 2025).

Parameter-Efficient Fine-Tuning (PEFT) methods (Houlsby et al., 2019) are designed to reduce the high expense of fine-tuning large language models. Since the number of trainable parameters directly affects GPU memory consumption during training (Suhoi, 2024), PEFT methods attempt to reduce the number of trainable parameters using various techniques. These methods can be divided into several categories (Han et al., 2024), among which reparameterized PEFT is notable for its high performance and lack of extra inference burden. Reparameterized PEFT represents the change in fine-tuned weight W' change with $W' = W_o + \Delta W$. Moreover, the storage cost is reduced because it is sufficient to store only the weights ΔW. We further classify reparameterized PEFT into dense-PEFT and sparse-PEFT, based on whether trainable weights ΔW are constructed as a dense or sparse matrix.

Dense-PEFT employs a dense matrix for trainable weights $\Delta W \in \mathbb{R}^{d\times d}$. LoRA (Hu et al., 2022) applies low-rank matrix decomposition to approximate $\Delta W = BA$, where $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times d}$. The rank r is much smaller than the dimension d, which leads to a reduction in the number of trainable parameters from $2rd$ to d^2 where $r \ll d$. DoRA (Liu et al., 2024) decomposes ΔW into its magnitude and direction as $\Delta W = m\Delta W'/\|\Delta W'\|$, thereby enhancing performance by bridging the gap between the learning patterns of full fine-tuning (Full-FT) and LoRA while maintaining a low computational cost. Variants of LoRA (Edalati et al., 2022; Zhang et al., 2023; Kopiczko et al., 2024; Hayou et al., 2024) have also been proposed. Additionally, methods to construct the trainable weight ΔW using Fourier transforms (?Gao et al., 2024) or specially designed operations have been proposed (Jiang et al., 2024; Tan et al., 2024). These dense-PEFT methods update all parameters of the target module and leads to catastrophic forgetting (Ramasesh et al., 2022; Dong et al., 2024; Luo et al., 2025). Therefore, it is not suited for model merging that combines models from multiple domains.

Sparse-PEFT employs a sparse matrix for trainable weights $\Delta W \in \mathbb{R}^{d\times d}$. A sparse matrix is a matrix in which most elements are zero. LoTA (Panda et al., 2024) introduces a sparse function to build $\Delta W = \text{Sparse}(I, V)$ where $I \in \mathbb{R}^k$ and $V \in \mathbb{R}^k$ indicate indices and values respectively. The sparse function constructs ΔW from a set of indices I and corresponding values V, representing only non-zero entries, thereby inducing sparsity. LoTA (Panda et al., 2024) identifies the index through lottery ticket scores obtained in a prior stage before fine-tuning. Although SpaRTA (Rios et al., 2025) randomly select indices, they similarly require a prior stage to identify and update them before fine-tuning. These sparse-PEFT methods require computational or design cost to determine which parameters to update before fine-tuning.

Our method simultaneously performs the search and update of trainable parameters inspired by LTH. Therefor it does not require a prior stage to identify indices and can be easily applied as a replacement for popular dense-PEFT methods such as LoRA and DoRA. To the best of our knowledge, this is the first PEFT framework that simultaneously searches for and updates target parameters. We validate the performance and efficacy of our method through comprehensive experiments.

3 BoLA: Our Sparse PEFT

Figure 1 introduces our newly proposed sparse-PEFT, **Bl**ock-wise **L**ottery Ticket **A**daptation (BoLA). To mitigate the computational burden of individually identifying important parameters, BoLA partitions the weights into several blocks and conducts exploration at the block level, as illustrated in Figure 1. BoLA searches for and updates block-wise target parameters, also known as lottery tickets (Frankle and Carbin, 2019), based on the block-wise magnitude of sparse weights. BoLA employs the reparametrization PEFT framework, where the fine-tuned weights $W' \in \mathbb{R}^{d\times d}$ can be represented as:

$$W' = W_o + \Delta W, \quad (1)$$

where $W_o \in \mathbb{R}^{d\times d}$ is the frozen pre-trained weights, $\Delta W \in \mathbb{R}^{d\times d}$ is the trainable block sparse weights, d is the dimension of input and output. As depicted in Figure 1, the sparse block weight is one in which nonzero elements are clustered into dense blocks, while the remainder consists of zero blocks. To construct sparse weights, both the values V and their corresponding insertion indices I are required. In this approach, since the sparse weights are constructed at the block level, the trainable weight ΔW can be represented as:

$$\Delta W = \text{BlockSparse}(I, V) \quad (2)$$

where $\text{BlockSparse}(I, V)$ is a function that generates a block-sparse weight given indices $I \in \mathbb{R}^k$ and values $V \in \mathbb{R}^{k\times(d/n)\times(d/m)}$. In this context, k denotes the number of trainable blocks, and n and m specify the division numbers along the input and output dimensions, respectively. The values with the same position are accumulated. Both I and V are derived from trainable score weights $W_s \in \mathbb{R}^{k\times n\times m}$ and trainable block weights $W_v \in \mathbb{R}^{k\times(d/n)\times(d/m)}$ as:

$$I = \arg\max_{n\times m}(W_s), \quad (3)$$

$$V = W_v, \quad (4)$$

where $\arg\max(\cdot)$ returns the index of the maximum value. Equation 3 identifies the top block positions with the highest scores across the $n \times m$ grid for each candidate in $W_s \in \mathbb{R}^{k\times n\times m}$, where each score reflects the importance of a block. Assuming that blocks with larger magnitudes correspond to more important block parameters, we identify the top-k indices $I \in \mathbb{R}^k$ by computing the $\arg\max$ over $W_s \in \mathbb{R}^{k\times n\times m}$ along with $n \times m$ dimension, which encodes the magnitude of each block. The corresponding values to be inserted at these positions are derived from $W_v \in \mathbb{R}^{k\times(d/n)\times(d/m)}$. That is, the indices are obtained from W_s, while the values themselves are sourced from W_v. The same

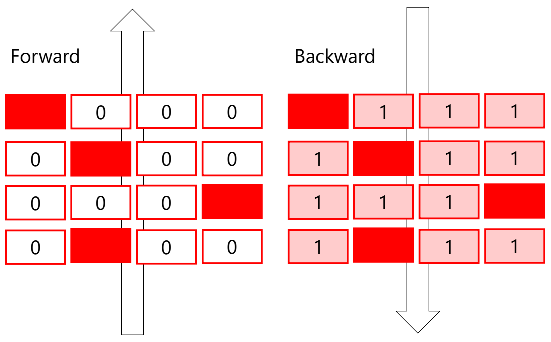

Figure 2: Illustration of our straight through estimator with ΔW. The non-trainable sparse blocks are set to zero during the forward pass, and set to one during the backward pass to allow gradients.

block can be selected multiple times, in which case its associated values are cumulatively aggregated. In order to ensure that gradients can flow into W_s by assigning it the role of a magnitude M, equation 1 can be rewritten as:

$$W' = W_o + M \odot \Delta W, \tag{5}$$

$$M = \sum_k G^k \odot W_s^k, \tag{6}$$

where $\odot$ denotes the element-wise (Hadamard) product and the matrix $G \in \mathbb{R}^{(k \times n \times m)}$ is defined to scale the selected blocks by a scalar factor α. The weights W_s are initialized using a uniform Kaiming distribution (He et al., 2015), while W_v is initialized to zero. This initialization ensures that the trainable weights ΔW start from zero, similar to LoRA variants (Hu et al., 2022; Edalati et al., 2022; Zhang et al., 2023; Kopiczko et al., 2024; Hayou et al., 2024; Liu et al., 2024), which is key for stable training (Liao et al., 2023).

Since this involves a sparse parameter update, gradients do not propagate through the score parameters W_s outside the top-k selection. Inspired by the motivation behind the Edge-popup algorithm (Ramanujan et al., 2020), we adopt straight-through estimation (Bengio et al., 2013) to allow gradients to flow through all score parameters, including those not selected in the forward pass. This enables dynamic block selection during training. In the backpropagation process, as shown in Figure 2, the non-trainable sparse blocks are set to one:

$$\nabla_{W_s} \mathcal{L} = \nabla_W \mathcal{L} \cdot \Delta W, \tag{7}$$

$$\Delta W = \begin{cases} \Delta W_{i,j} & \text{if } (i,j) \in I, \\ 1.0 & \text{if otherwise,} \end{cases} \tag{8}$$

then it allows gradients to propagate seamlessly even through blocks that do not contain any assigned values. This mechanism not only enables gradient flow through all parameters W_s but also allows the model to dynamically re-evaluate and update block importance throughout training, leading to more flexible and adaptive sparse representations.

The number of trainable parameters $\phi = knm + kdd/nm = k(nm + dd/nm)$ is determined by the weights $W_s \in \mathbb{R}^{k \times n \times m}$ and $W_v \in \mathbb{R}^{k \times (d/n) \times (d/m)}$ where n and m are block size and k is the number of trainable blocks. Setting $nm = d$ yields $\phi = k(d + d) = 2kd$ such that the number of trainable parameters is similar to $\phi = 2rd$ as in LoRA (Hu et al., 2022). As a result, this formulation enables straightforward replacement of LoRA, thereby positioning BoLA as a compatible and efficient alternative for parameter-efficient fine-tuning.

4 Experiments

We conduct a series of experiments to demonstrate the effectiveness of our method across various language models and tasks. Specifically, we use RoBERTa as a small-scale model and LLaMA3 as a large-scale model. For evaluation, we consider datasets from the GLUE benchmark, along with benchmarks targeting mathematical and commonsense reasoning. We adopt LoRA (Hu et al., 2022), DoRA (Liu et al., 2024), and SpaRTA (Rios et al., 2025) as strong baselines. To ensure a fair comparison, we configured the hyperparameters such that the number of trainable parameters matched that of LoRA and SpaRTA. However, in DoRA, the number of trainable parameters was increased by the dimension d of the magnitude. Furthermore, we maintained hyperparameter settings as consistent as possible across different experimental conditions to facilitate reproducibility (see Appendix A for details). All experiments are conducted on NVIDIA H100 80GB GPUs. We used four GPUs for RoBERTa-125M and eight GPUs for Llama3-8B. RoBERTa-125M is trained using float32 precision owing to its smaller size, while LLaMA3-8B is trained using bfloat16 precision to reduce memory consumption and accelerate training on H100 GPUs. All experiments are implemented using PyTorch, Hugging Face Transform-

Method	#Params.	MNL	SST2	MRPC	CoLA	QNLI	QQP	RTE	SST-B	Avg.
Full-FT	125M	**87.6**	94.7	88.7	62.1	92.9	**91.8**	78.3	90.6	85.9
LoRA	0.3M	87.5	95.1	89.5	63.6	**93.1**	90.9	78.0	**90.9**	**86.1**
DoRA	0.3M	85.2	94.4	87.8	**64.8**	92.4	88.5	**79.8**	**90.9**	85.5
SpaRTA	0.3M	86.9	**95.2**	**89.7**	61.1	92.5	90.5	75.5	90.8	85.3
BoLA	0.3M	84.2	93.6	89.0	60.1	90.7	88.3	75.8	89.5	83.9

Table 1: RoBERTa-125M with different adaptation methods on the GLUE benchmark. #Params. indicates that the number of trainable paramters excluded with classification head module. We report the overall (matched and mismatched) accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. For all metrics, higher values indicate better performance. The highest values are indicated in bold.

Method	#Params.	MultiArith	GSM8K	AddSub	AQuA	SingleEq	SVAMP	Avg.
Base		22.8	6.3	25.1	**20.5**	13.0	10.5	16.4
LoRA	56.6M	48.0	21.7	79.5	13.8	76.2	51.7	48.5
DoRA	57.4M	51.8	23.4	77.7	15.7	77.2	**54.9**	50.1
SpaRTA	56.6M	52.0	**24.3**	82.5	10.2	80.5	51.0	50.1
BoLA	56.6M	**52.8**	19.9	**87.1**	9.1	**81.1**	**54.9**	**50.8**

Table 2: Llama3-8B with different adaptation methods on the math benchmark. #Params. indicates that the number of trainable paramters. We report the accuracy on each dataset and its higher value is better for all metrics. The highest values are indicated in bold.

Method	#Params.	BoolQ	PIQA	SIQA	HS	WG	ARC-e	ARC-c	OBQA	Avg.
Base		51.2	63.2	33.5	9.3	45.0	27.4	27.6	23.6	35.1
LoRA	56.6M	**75.0**	79.4	**80.0**	96.0	85.6	90.4	79.4	85.2	83.9
DoRA	57.4M	74.6	**89.3**	79.9	95.5	85.6	90.5	**80.4**	**85.8**	84.1
SpaRTA	56.6M	73.7	87.7	79.1	95.3	84.1	**91.8**	79.8	84.0	84.4
BoLA	56.6M	74.4	87.1	79.7	**96.4**	**85.2**	91.5	79.4	82.1	**84.5**

Table 3: Llama3-8B with different adaptation methods on the commonsense benchmark. #Params. indicates that the number of trainable paramters. We report the accuracy on each dataset and its higher value is better for all metrics. The highest values are indicated in bold. Note that HS and WG are abbreviations for HellaSwag and WinoGrande, respectively.

ers (Wolf et al., 2020), and the PEFT (Mangrulkar et al., 2022) library.

4.1 GLUE Benchmark

We evaluate BoLA in comparison with LoRA, DoRA, and SpaRTA using the relatively small language model RoBERTa-125M (Liu et al., 2019) for English GLUE benchmark (Wang et al., 2018). Although GLUE consists of nine downstream tasks, consistency with previous work (Hu et al., 2022), we report the results on eight downstream tasks and their average. We also report the number of trainable parameters, excluding the classification head to ensure a fair comparison of adaptation modules. We present the results in Table 1. The results for each run are taken from the best epoch. On average, BoLA performed slightly worse than Full-FT. However, it achieved competitive performance with Full-FT on MRPC and SST-B. This limitation arises from the fact that BoLA can update only kd parameters, which hinders the effective incorporation of domain knowledge into RoBERTa-125M.

4.2 Math and Commonsense Benchmark

We evaluate BoLA in comparison with LoRA, DoRA, and SpaRTA using the LLM LLaMA3-8B (Llama-Team, 2024) and benchmarks for math and commonsense reasoning. The math and commonsense benchmarks consist of eight and six tasks, respectively, each with predefined training and testing sets. Following the experimental setup of (Hu et al., 2023), we merge the training data from all eight commonsense tasks to construct a unified training dataset and evaluate the individual testing dataset for each task. We use the math 10K

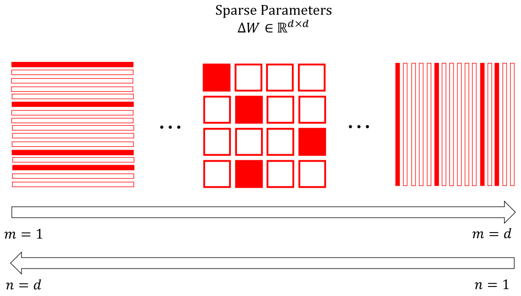

Figure 3: Transition diagram of block shapes determined by the hyperparameters n and m. Red and white blocks represent selected and unselected parameters, respectively. We define configurations as “structured” when either $n = d$ or $m = d$

and commonsense 170K datasets[1] to fine-tune the models.

The results of the math benchmark are shown in Table 2. BoLA demonstrated improved performance over the baseline on the MultiArith, AddSub, SingleEq, and SVAMP tasks. On average, BoLA achieves performance comparable to DoRA and SpaRTA. The results of the commonsense benchmark are shown in Table 3. BoLA demonstrated improved performance over the baseline on the HS and WG tasks. On average, BoLA achieves higher performance than DoRA and SpaRTA. In the case of commonsense tasks, the base model already performs reasonably well, and thus no performance degradation was observed across any of the PEFT methods. It is observed that, in contrast to dense approaches, sparse methods benefit more significantly from larger model sizes.

5 Ablation Study

In this section, we introduce the impact of hyperparameter in BoLA.

5.1 Structured vs. Unstructured

The hyperparameters n and m determine the number of block divisions and the input and output dimensions, thereby defining the shape of the trainable block weights. Figure 4 presents the performance trends as n is varied under the constraint $nm = d$. In this study, we define configurations as “structured” when either $n = d$ or $m = d$, meaning that block weights span the full input or output dimension. All other configurations are considered “unstructured”.

[1] https://github.com/AGI-Edgerunners/LLM-Adapters

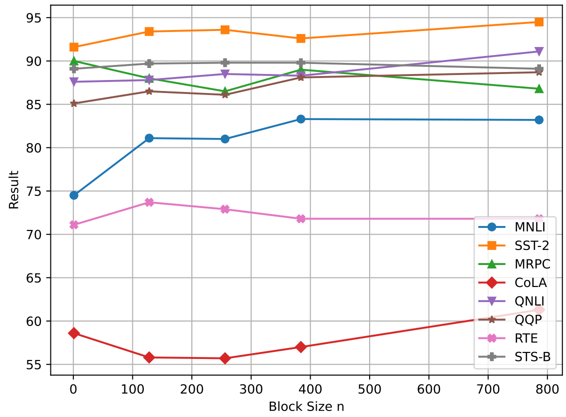

Figure 4: Evaluation results for various block sizes n for RoBERTa-125M on GLUE tasks. The horizontal axis corresponds to size m, which is varied from 1 to $d = 768$ under the constraint that $nm = d = 768$ remain constant. Each marker represents the evaluation result for a downstream task included in the GLUE benchmark.

We evaluate the impact of block weight shapes on RoBERTa-125M performance using the GLUE benchmark, as shown in Figure 4. Overall, it is observed that increasing the block size n generally leads to improved performance. This indicates that a higher output dimensionality plays a crucial role, and therefore, it is preferable to set the value of n to its maximum. Since the structured configuration with the maximum n performs well across many downstream tasks, it is more effective to focus on tuning the number of base blocks k and the scaling factor α. This is analogous to tuning the rank r and scaling factor α in methods such as LoRA and DoRA.

6 Analysis of Sparse Update

In this section, we present two complementary benefits of sparse-PEFT compared with dense-PEFT, namely, enhanced interpretability and reduced performance dropout during model merging.

6.1 Interpretability of Model Structure

The sparsity of BoLA enhances model interpretability by making it easier to identify where domain-specific knowledge is acquired. Figure 5 shows the Llama3-8B (Llama-Team, 2024) weights ΔW of LoRA and BoLA, trained on math and commonsense datasets. LoRA updates all parameters, whereas BoLA searches for and updates only a

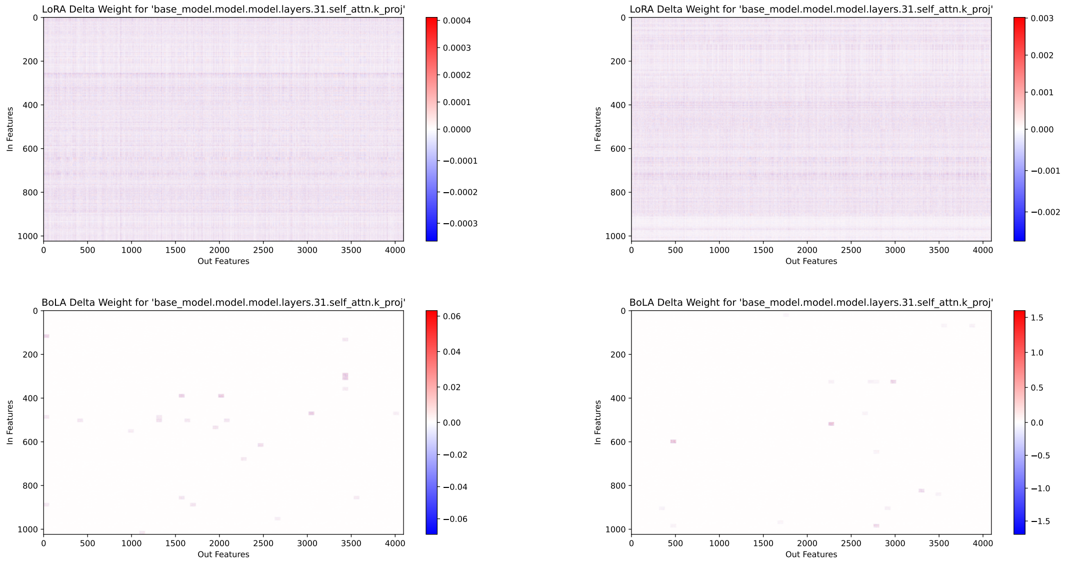

Figure 5: Visualization map of the trained weights ΔW in the last layer of Llama3-8B. The top shows LoRA weights and the bottom shows BoLA weights. The maps on the left and right were trained on mathematical and commonsense reasoning data, respectively. Color intensity indicates the magnitudes of the weights. In BoLA, although the number of selectable blocks is $k = 32$, it may appear to be fewer due to index duplication.

small, important subset. Therefore, in BoLA, it becomes evident that most of the trainable weights ΔW are zero.

BoLA divides the weights into n and m blocks in the input and output dimensions, respectively, and only the k blocks with the highest scores are selected and updated. Interestingly, BoLA tends to utilize all of k blocks for the math domain. In contrast, BoLA tends to utilize significantly fewer blocks for the commonsense domain. This suggests that the model tends to acquire a greater amount of knowledge in the math domain, where its initial performance is weaker, while acquiring only limited knowledge in the commonsense domain, where it initially performs well.

As shown in Figure 5, while LoRA has full access to all parameters, BoLA achieves comparable performance by accessing only $k/nm = 32/4096 = 0.78\%$ of the total parameters. Therefore, only a small subset of parameters is crucial for domain-specific adaptation in LLMs. This observation is consistent with prior work suggesting that certain parameters play specialized roles (Yu et al., 2025).

6.2 Sparse Model Merge

One notable property of sparse-PEFT is the minimal overlap between weights trained on different domains, which contributes to reduced interference during model merging. LoTA (Panda et al., 2024) leverages this property to demonstrate that catastrophic forgetting can be mitigated during model merging. Catastrophic forgetting (Ramasesh et al., 2022; Dong et al., 2024; Luo et al., 2025) refers to the degradation in performance on previously learned tasks that occurs when LLMs are fine-tuned on multiple domains or tasks in a sequential manner. Figures 6 and 7 show that sparse-PEFT BoLA similarly reduces catastrophic forgetting during model merging. The models used in this table were trained as described in section 4. In the math benchmark, LoRA exhibits significantly degraded performance across all datasets when merging weights trained on math with those trained on commonsense, resulting in an average drop of 2.7 points. Notably, MultiArith experiences substantial performance degra-

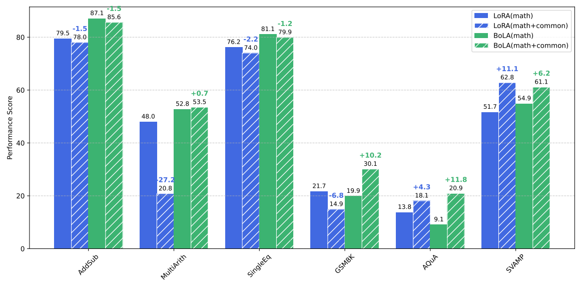

Figure 6: Performance comparison of model merging on the math benchmark. Bar plots show the performance of math-domain LoRA and BoLA models (filled), and their counterparts after merging with commonsense-domain models (hatched).

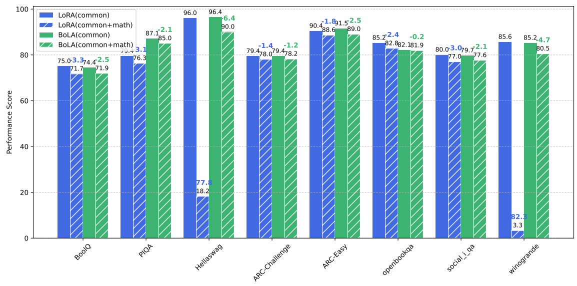

Figure 7: Performance comparison of model merging on the commonsense benchmark. Bar plots show the performance of commonsense-domain LoRA and BoLA models (filled), and their counterparts after merging with math-domain models (hatched).

dation. In contrast, BoLA exhibits a performance improvement of 3.6 points on average after merging, demonstrating its robustness to catastrophic forgetting. No notable performance degradation occurs on MultiArith. In the commonsense benchmark, LoRA significantly degraded performance across all datasets when merging weights trained on math with those trained on commonsense, resulting in an average drop of 22.1 points. Notably, Hellaswag and Winogrande experience substantial performance degradation. In contrast, BoLA exhibits only a minor performance decline of 2.7 points on average after merging, demonstrating its robustness to catastrophic forgetting. No notable performance degradation occurs on Hellaswag and Winogrande. BoLA models trained independently on the math and commonsense domains can be merged without significant performance degradation.

Unlike dense-PEFT approaches, which often suffer from catastrophic forgetting and require costly fine-tuning, sparse-PEFT enables seamless integration of instruction-following and domain-specific capabilities. As shown in Figure 8, a key challenge in domain adaptation using dense-PEFT is that all parameters are overwritten during training, potentially erasing the model's original instruction-following behavior. In contrast, sparse-PEFT updates only a small subset of parameters, allowing domain adaptation while preserving the core characteristics of the instruction model.

Furthermore, if the instruction model itself is trained using sparse-PEFT, subsequent domain-specific adaptation can be achieved with even fewer parameter updates. This layered sparsity enables efficient and modular fine-tuning, making sparse-PEFT a highly practical approach for developing locally specialized models.

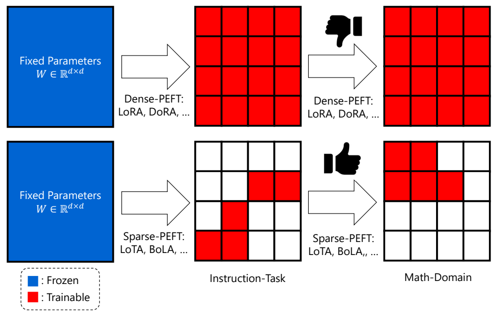

Figure 8: Workflow diagram illustrating model development using dense-PEFT and sparse-PEFT. During domain-specific training, dense-PEFT updates all model parameters, which may lead to overwriting the original instruct model. In contrast, sparse-PEFT selectively updates only a small subset of parameters, thereby preserving the instruct model and reducing the risk of unintended changes.

In summary, sparse-PEFT offers a robust and efficient approach to developing domain-specialized models without compromising the integrity or performance of the base model.

7 Conclusion

This work introduces **Bl**ock-wise **L**ottery Ticket **A**daptation (BoLA), as a novel and simple sparse-PEFT for large language models designed to simultaneously search for and update parameters, a capability not found in previous methods. Furthermore, BoLA is highly compatible with popular dense-PEFT methods LoRA and DoRA, across various downstream tasks and model architectures. Therefore, BoLA serves as a costless alternative to LoRA and DoRA, as its ability to merge fine-tuned weights back into the pre-trained model guarantees no extra inference overhead. Furthermore, unlike dense-PEFT, BoLA provides interpretability in knowledge acquisition and supports model merging without catastrophic forgetting. For future work, we wish to explore the generalizability of our method in other domains beyond audio and vision.

Limitations

Our PEFT approach demonstrates the potential to alleviate catastrophe forgetting during model merging. However, when the characteristics of the target domains differ significantly, there is no guarantee that the trained parameters will remain mutually exclusive. As a future direction, we plan to extend the method to allow users to explore and update critical parameters within user-specified regions, enabling more effective control over parameter redundancy.

References

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *Preprint*, arXiv:1308.3432.

Xiaohan Chen, Jason Zhang, and Zhangyang Wang. 2022. Peek-a-boo: What (more) is disguised in a randomly weighted neural network, and how to find it efficiently. In *International Conference on Learning Representations*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Guanting Dong, Hongyi Yuan, Keming Lu, Chengpeng Li, Mingfeng Xue, Dayiheng Liu, Wei Wang, Zheng Yuan, Chang Zhou, and Jingren Zhou. 2024. How abilities in large language models are affected by supervised fine-tuning data composition. *Preprint*, arXiv:2310.05492.

Ali Edalati, Marzieh Tahaei, Ivan Kobyzev, Vahid Partovi Nia, James J. Clark, and Mehdi Rezagholizadeh. 2022. Krona: Parameter efficient tuning with kronecker adapter. *Preprint*, arXiv:2212.10650.

Jonathan Frankle and Michael Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *Preprint*, arXiv:1803.03635.

Ziqi Gao, Qichao Wang, Aochuan Chen, Zijing Liu, Bingzhe Wu, Liang Chen, and Jia Li. 2024. Parameter-efficient fine-tuning with discrete fourier transform. *Preprint*, arXiv:2405.03003.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. *Preprint*, arXiv:2403.14608.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. 2024. LoRA+: Efficient low rank adaptation of large models. In *Forty-first International Conference on Machine Learning*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Preprint*, arXiv:1502.01852.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*.

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

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

Ting Jiang, Shaohan Huang, Shengyue Luo, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, Qi Zhang, Deqing Wang, and Fuzhen Zhuang. 2024. Mora: High-rank updating for parameter-efficient fine-tuning. *Preprint*, arXiv:2405.12130.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *Preprint*, arXiv:2001.08361.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki M Asano. 2024. VeRA: Vector-based random matrix adaptation. In *The Twelfth International Conference on Learning Representations*.

Baohao Liao, Shaomu Tan, and Christof Monz. 2023. Make your pre-trained model reversible: From parameter to memory efficient fine-tuning. *Preprint*, arXiv:2306.00477.

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

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. *Preprint*, arXiv:1907.11692.

Llama-Team. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2025. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *Preprint*, arXiv:2308.08747.

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

Ashwinee Panda, Berivan Isik, Xiangyu Qi, Sanmi Koyejo, Tsachy Weissman, and Prateek Mittal. 2024. Lottery ticket adaptation: Mitigating destructive interference in llms.

Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. 2020. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Vinay Venkatesh Ramasesh, Aitor Lewkowycz, and Ethan Dyer. 2022. Effect of scale on catastrophic forgetting in neural networks. In *International Conference on Learning Representations*.

Jesus Rios, Pierre Dognin, Ronny Luss, and Karthikeyan N. Ramamurthy. 2025. Sparsity may be all you need: Sparse random parameter adaptation. *Preprint*, arXiv:2502.15975.

Daniil Suhoi. 2024. Efficient deep learning: A comprehensive overview of optimization techniques. Hugging Face Blog.

Wenxuan Tan, Nicholas Roberts, Tzu-Heng Huang, Jitian Zhao, John Cooper, Samuel Guo, Chengyu Duan, and Frederic Sala. 2024. More fine-tuning with 10x fewer parameters. In *ICML 2024 Workshop on Foundation Models in the Wild*.

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.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Jing Xu and Jingzhao Zhang. 2024. Random masking finds winning tickets for parameter efficient fine-tuning. *Preprint*, arXiv:2405.02596.

Haoran You, Baopu Li, Zhanyi Sun, Xu Ouyang, and Yingyan Celine Lin. 2025. Supertickets: Drawing task-agnostic lottery tickets from supernets via jointly architecture searching and parameter pruning. *Preprint*, arXiv:2207.03677.

Mengxia Yu, De Wang, Qi Shan, Colorado J Reed, and Alvin Wan. 2025. The super weight in large language models. *Preprint*, arXiv:2411.07191.

Fei Yuan, Chang Ma, Shuai Yuan, Qiushi Sun, and Lei Li. 2025. KS-lottery: Finding certified lottery tickets for multilingual transfer in large language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 9077–9090, Albuquerque, New Mexico. Association for Computational Linguistics.

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

A Hyperparameters

For all experiments, we select the learning rate that achieves the highest validation performance from the range $\{1\text{e}{-3}, 5\text{e}{-4}, 1\text{e}{-4}, 5\text{e}{-5}, 1\text{e}{-5}\}$, and report the corresponding results.

Hyperparameter	LoRA	DoRA	SpaRTA	BoLA
Epochs	30			
Dropout	0.1			
Sequence Length	512			
Optimizer	AdamW			
LR	1e-3	5e-4	1e-3	1e-3
LR Scheduler	Linear			
Batch Size	64			
Warmup Ratio	0.06			
Weight Decay	0.1			
Target Modules	Q,K			
r	8	8		
(n, m)				$(24, 32)$
k			12,288	8
α	16	16	4	4
Dropout	0.1			

Table 4: Hyperparameter configurations of LoRA, DoRA, SpaRTA, and BoLA for RoBERTa-125M on the GLUE benchmark.

Hyperparameter	LoRA	DoRA	SpaRTA	BoLA
Epochs	3			
Dropout	0.1			
Sequence Length	512			
Optimizer	AdamW			
LR	5e-5	5e-5	1e-5	1e-3
LR Scheduler	Linear			
Batch Size	128			
Warmup Ratio	0.06			
Weight Decay	0.0			
Target Modules	Q,K,V,U,D			
r	32	32		
(n, m)				$(64, 64)$
k			353,894	32
α	64	64	4	4
Dropout	0.1			

Table 5: Hyperparameter configurations of LoRA, DoRA, SpaRTA, and BoLA for Llama3-8B on the math benchmark.

Hyperparameter	LoRA	DoRA	SpaRTA	BoLA
Epochs		3		
Dropout		0.1		
Sequence Length		512		
Optimizer		AdamW		
LR	1e-4	1e-4	1e-5	5e-5
LR Scheduler		Linear		
Batch Size		128		
Warmup Ratio		0.06		
Weight Decay		0.0		
Target Modules		Q,K,V,U,D		
r	32	32		
(n, m)				$(64, 64)$
k			353,894	32
α	64	64	4	4
Dropout		0.1		

Table 6: Hyperparameter configurations of LoRA, DoRA, SpaRTA, and BoLA for Llama3-8B on the commonsense benchmark.