LoRA-NAS: Optimizing Spatial Adapter Placement in Large Language Models via Evolutionary Search

Alejandro Rosales-Perez
Centro de Investigación en Matemáticas
Avenida Alianza Centro 502, PIIT, Monterrey, 66628, Nuevo León, Mexico
Email: alejandro.rosales@cimat.mx

***Abstract*—Parameter-Efficient Fine-Tuning is essential for adapting Large Language Models to downstream tasks under limited computational resources. While Low-Rank Adaptation (LoRA) is widely adopted, conventional implementations typically use uniform placement heuristics, inserting adapters into all layers without considering task relevance. Dynamic methods such as AdaLoRA optimize adapter rank but often retain dense topologies and overlook the spatial distribution of adapters. This study introduces LoRA-NAS, a framework that employs evolutionary search to optimize the spatial allocation of adapters. Moreover, LoRA-NAS also optimizes the hyperparameters for LoRA adapters, customized by each layer. By formulating adapter placement as an optimization problem, LoRA-NAS autonomously identifies the most effective subset of layers for adaptation and hyperparameters. Extensive experiments on the GLUE benchmark with LLaMA-3-8B demonstrate that LoRA-NAS consistently surpasses both standard uniform baselines and leading dynamic-rank methods. The results indicate that selecting the placement of adaptation layers has a greater impact on generalization than adaptation capacity.**

***Index Terms*—Neural Architecture Search, Evolutionary Algorithms, Parameter-Efficient Fine-Tuning**

I. Introduction

The rapid advancement of Large Language Models (LLMs), such as the LLaMA-3 [1] and models like GPT-4 [2], has significantly expanded the capabilities of natural language processing, enabling complex reasoning and creative generation. Despite these advancements, leveraging LLMs for specialized, domain-specific tasks remains challenging. Although pre-trained models demonstrate strong zero-shot performance, they frequently require fine-tuning to align with specific instructions or to acquire domain-specific expertise. Traditional full fine-tuning, which updates all model parameters, is increasingly impractical as model sizes reach billions of parameters [3]. The substantial memory requirements for optimizer states and gradients, along with the storage demands of maintaining separate model copies for each task, make this approach infeasible for most real-world scenarios.

Parameter-Efficient Fine-Tuning (PEFT) [4] has become a central research focus for addressing scalability limitations in large models. PEFT methods adapt these models by updating only a small subset of parameters, while keeping most pre-trained weights fixed. Among these approaches, Low-Rank Adaptation (LoRA) [5] is distinguished by its memory efficiency and minimal impact on inference latency. LoRA introduces trainable rank-decomposition matrices into transformer layers, enabling effective adaptation with a minimal parameter budget.

Although LoRA has been widely adopted, standard implementations often rely on heuristics and inflexible design choices. Practitioners often apply a uniform strategy by inserting adapters into all layers, or by limiting them to specific modules such as query and value projections, typically guided by manual intuition rather than empirical evidence. This methodology overlooks the complexity of deep neural networks, in which different layers contribute unequally to various tasks [6]. Recent approaches, such as AdaLoRA [7], address this limitation by dynamically allocating parameter budgets via singular value decomposition (SVD), enabling rank pruning during training. While these dynamic methods represent progress, they focus on optimizing adapter rank rather than the spatial placement of adapters within the network.

Neural Architecture Search (NAS) is a research area focused on the automatic design of neural network architectures, achieving state-of-the-art results across multiple domains. Recent studies have incorporated LoRA into NAS frameworks. For example, Shears [8] and LoNAS [9] use weight sharing and evolutionary algorithms to optimize LoRA ranks across layers during LLM compression. These methods leverage NAS to fine-tune adapter configurations, thereby improving efficiency. However, they do not address the challenge of identifying optimal adapter placement within LLMs.

The placement of adapters in LLMs may be more critical than their size, and we believe that NAS can help address the challenge of optimal adapter placement. Instead of introducing new layer types or connection structures, NAS enables the exploration of the space of possible adapter placements and their associated hyperparameters. This paper presents LoRA-NAS, a novel framework that integrates NAS with low-rank adaptation. Rather than relying on static heuristics or continuous rank relaxation, LoRA-NAS formulates adapter placement as a search problem. An evolutionary algorithm explores the extensive search space of possible layer configurations to identify the optimal subset of modules for adaptation on a given dataset. This approach facilitates the construction of highly specialized architectures that allocate trainable parameters precisely where they are most needed, thereby reducing redundancy. The main contributions are as follows:

1) This work introduces LoRA-NAS to automate the iden-

tification of optimal adapter placement, enabling task-specific architectures that are more efficient than uniform baselines.
2) We present a rigorous comparative analysis showing that LoRA-NAS outperforms both static LoRA configurations and state-of-the-art dynamic rank allocation methods, such as AdaLoRA and LoNAS. Our findings indicate that selecting a smaller, more relevant set of layers is more effective than rank sparsity, which reduces dimensions across all layers.
3) To address concerns about the reproducibility of NAS and PEFT research, we conduct extensive experiments using multi-seed protocols. We show that the performance improvements achieved by LoRA-NAS are statistically significant and robust across a range of Natural Language Understanding benchmarks, thereby validating the reliability of our search method.

The remainder of this paper is organized as follows. Section II presents the related work on PEFT and NAS. Section III describes the proposed method. Then, Section IV details the dataset used and reference methods, and Section V reports the experimental results. Finally, Section VI provides the main conclusions of the paper.

II. Related Works

This section reviews the related works. Section II-A presents PEFT and the evolution towards dynamic allocation strategies and Section II-B examines the emerging application of NAS within the context of LLMs.

A. Parameter-Efficient Fine-Tuning and Dynamic Allocation

The transition from full fine-tuning to PEFT has been motivated by the necessity to adapt billion-parameter models within limited computational budgets. For instance, tuning a model with 8 billion parameters on a single GPU requires significant time and resource optimization due to high computational demands [5], [10]. Initial methods, such as Adapters [11], introduced bottleneck layers between existing Transformer sub-layers. Later, LoRA [5] offered a more efficient alternative by incorporating low-rank trainable decomposition matrices into the frozen weights, which significantly reduced inference latency through weight merging.

Standard LoRA implementations primarily depend on static and heuristic placement strategies. Practitioners frequently distribute adapters uniformly across all layers or confine them to specific modules, such as query and value projections, often without empirical justification. Recent research on adapter placement challenges this uniformity by demonstrating that layer sensitivity to downstream tasks varies considerably.

Dynamic PEFT methods have been developed to allow more flexibility. Techniques such as AdaLoRA [7] and DyLoRA [12] utilize SVD to adaptively allocate the parameter budget, pruning the ranks of less important modules during training. Although these methods effectively optimize adaptation rank, they generally assume a dense topology by maintaining active adapter modules across most network layers. The rank-sparsity approach fundamentally differs from topological sparsity, in which entire layers are excluded from adaptation. Optimizing the spatial distribution of adapters, determining where to adapt rather than solely how much, represents a critical yet underexplored efficiency dimension that may help prevent overfitting.

B. Neural Architecture Search for LLM Adaptation

NAS automates the design of optimal network architectures and has achieved state-of-the-art results in several domains. However, applying traditional NAS to LLMs is challenging due to the high computational cost of training super-networks or evaluating numerous candidates. As a result, recent efforts have focused on NAS for PEFT [13], where the search space is limited to adaptation module configurations rather than the base model.

Recent studies, including Shears [8] and LoNAS [9], have advanced this direction. LoNAS employs evolutionary algorithms to identify the optimal rank for each layer, thereby automating the rank allocation process. Similarly, Shears combines LoRA with sparsity constraints to achieve model compression. Although these methods share the objective of automation, they primarily focus on the rank dimension and conceptualize architecture search as a budget-allocation problem rather than addressing structural considerations.

LoRA-NAS differs from these approaches by framing adapter placement as a combinatorial search problem. Rather than continuously adjusting the rank, this method explores the discrete space of layer selection. By identifying the optimal subset of layers for intervention, LoRA-NAS aims to discover minimal, task-specific topologies. This strategy complements existing rank-search methods and provides an alternative pathway to efficiency, supporting the hypothesis that effective adaptation in LLMs depends on both topology and capacity.

III. Low-Rank Adapter by Neural Architecture Search

The LoRA-NAS framework addresses the challenge of efficiently adapting LLMs by formulating the process as an optimization problem. The primary goal is to discover the optimal topological configuration of Low-Rank Adapters. Unlike traditional methods that predetermine adapter placement, LoRA-NAS jointly explores the search space of layer selection and hyperparameter configuration to maximize task-specific performance within computational constraints. Fig. 1 graphically depicts LoRA-NAS.

A. Problem Formulation and Search Space

Let M_θ be a pre-trained LLM parameterized by weights θ. In standard LoRA, a set of target modules are updated via low-rank matrices $\mathbf{B} \in \mathcal{R}^{d\times r}$ and $\mathbf{A} \in \mathcal{R}^{k\times k}$, such that $\Delta W = \mathbf{BA}$, where ΔW represents the change in weights. The rank r and scaling factor α are typically hyperparameters fixed across all layers.

This assumption is reconsidered by introducing a layer-wise search space. Let $L = \{0, 1, \ldots N-1\}$ represent the indices of

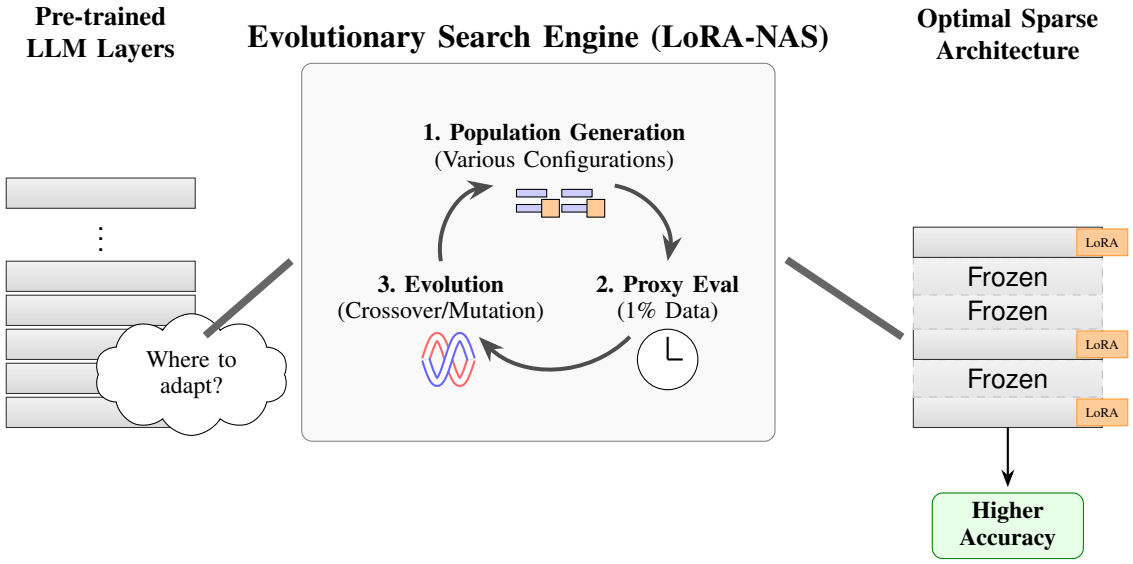

Fig. 1. Overview of the LoRA-NAS Framework. The process begins with a pre-trained LLM (left), and an evolutionary algorithm (middle) iteratively evolves a population of candidate architectures using a low-fidelity proxy evaluation. The search converges on an optimal architecture (right).

adaptable layers in the transformer. A candidate architecture $\mathcal{A}$ is defined as a tuple $\mathcal{A} = (I, r, \alpha)$, where:

- Layer-wise configuration (I): A subset $I \subseteq L$ specifies the active layers where adapters are inserted. The cardinality of this subset is variable, enabling the search to dynamically identify sparse or dense adaptation strategies. Fewer layers can enhance speed and efficiency, especially for time-constrained applications, while more layers might improve accuracy by capturing complex dependencies, albeit at a higher computational cost.
- Hyperparameter configuration (r, α): Discrete values are selected from predefined sets R and S, where $r \in R$ determines the capacity of the update matrices and $\alpha \in S$ controls the adaptation scaling.

The optimization objective is to find the configuration $\mathcal{A}^\star$ that maximizes the validation accuracy on a downstream task T. In simpler terms, we aim to identify the configuration that achieves the highest level of accuracy when tested on a specific task. This is represented mathematically as:

$$\mathcal{A}^\star = \arg\max_{\mathcal{A} \in Z} \left(\mathcal{L}\left(M_\theta, \mathcal{A}\left(D_{train} \right), D_{val} \right) \right) \tag{1}$$

where Z denotes the combinatorial search space. The size of Z is $2^N \times |R| \times |S|$, for an N-layer LLM. For a 32-layer model with three options for rank and alpha, the search space exceeds 3.810^{10} combinations. This scale renders grid search computationally infeasible and necessitates the use of efficient heuristic approaches such as evolutionary algorithms.

B. Evolutionary Search Strategy

To efficiently traverse this extensive search space, LoRA-NAS utilizes an evolutionary algorithm optimized for variable-length representations. The search process is iterative, evolving a population P of candidate architectures across G generations. We formulate adapter placement as a discrete combinatorial search problem. While Gradient-based NAS methods (e.g., DARTS) are highly efficient in other domains, they require a continuous relaxation of the search space, which necessitates maintaining a massive super-network in GPU memory. For billion-parameter models like LLaMA-3-8B, this continuous relaxation creates prohibitive memory bottlenecks. Reinforcement Learning (RL) is another alternative, but RL agents frequently suffer from high variance and sample inefficiency when navigating large, discrete search spaces with sparse, delayed rewards (such as validation accuracy). EAs bypass these limitations by naturally handling non-differentiable, discrete metrics and allowing candidate architectures to be instantiated and evaluated independently, strictly within memory limits.

1) Individual Representation: Each individual $I_j \in P$ is encoded as a structured vector containing:

- Layer locus: a variable-length list of integers representing the indices of adapted layers, for example, $I_j = 1, 5, 28$. This structure enables the algorithm to capture strictly positional dependencies, such as adaptation limited to early layers.
- Hyperparameter locus: Two discrete scalars representing r and α.

The initial population P_0 is generated by randomly sampling layer subsets of size k and uniformly sampling r and α. This constraint on initial sparsity encourages the algorithm to begin with lightweight models, increasing complexity only when warranted. For parent selection, tournament selection is employed, where a subset of individuals is sampled from the population, and the individual with the highest fitness is selected for reproduction. By using a tournament size of three, this approach balances exploitation through competitive

selection with maintaining population diversity, thus mitigating risks of premature convergence. This size is chosen because it provides an effective trade-off between exploring promising solutions and preserving genetic diversity.

2) Genetic Operators: To generate offspring, we apply specialized crossover and mutation operators designed to explore both the topological and parametric dimensions:

- Layer-wise crossover: Given two parents p_1 and p_2 with layer sets I_1 and I_2, the offspring's layer set I_{child} is constructed by inheriting a random subset of layers from the union $I_1 \cup I_2$. This process enables the recombination of effective features, such as combining adaptation in early layers from p_1 with late-layer adaptation from p_2. Integrating early-layer adaptations allows the system to quickly capture foundational linguistic patterns, while late-layer adaptations refine the model's understanding of nuanced contextual elements. Hyperparameters r and α are stochastically inherited from either parent.
- Mutation: To promote exploration, mutation modifies the architecture with probability m_p. This operator may add a new layer (sampled from $L \setminus I$) or remove an existing layer from I. This bidirectional mutation enables the search to escape local optima by pruning redundant layers or activating previously unused ones. By removing layers, the model can maintain a leaner architecture, reducing complexity and enhancing generalization capabilities in small data regimes.
- Hyperparameter mutation: Independently, the hyperparameters r and α may be resampled from their respective domains, allowing the fine-tuning of the adaptation capacity.

C. Efficient Fitness Estimation

A significant bottleneck in NAS for LLMs is the high evaluation cost. Training a candidate architecture to convergence for fitness assessment is prohibitively expensive. To address this challenge, LoRA-NAS employs a low-fidelity proxy evaluation strategy.

A proxy dataset $D_p \subset D_t$ is constructed, comprising 1% of the total training samples. This sample is selected using a stratified random sampling method to ensure representativeness across different data categories. For each candidate $\mathcal{A}$, the adapter configuration is instantiated and fine-tuned for a single epoch on D_p. The fitness score is determined by the accuracy measured on a held-out validation subset, D_v, consisting of 5% of the data.

$$\text{Fitness}\left(\mathcal{A}\right) \approx \text{acc}\left(M_\theta, \mathcal{A}, D_p, D_v\right) \tag{2}$$

The low-fidelity proxy evaluation is employed to balance search fidelity with computational feasibility. Although this method only approximates the true loss landscape, it enables exploration of a substantially larger search space, thereby prioritizing the identification of global configurations rather than precise individual accuracy estimation. This strategy reduces evaluation time from hours to minutes per individual, enabling the search to assess hundreds of configurations within a reasonable GPU-time budget.

Once the evolutionary search is over, the best-performing individual A_{best} is selected. This optimal configuration is then subjected to a full fine-tuning stage on the complete dataset to obtain the final model performance reported in the experiments.

IV. Experimental Settings

This section outlines the experimental protocol established to validate the efficacy of LoRA-NAS. It provides an overview of the benchmark datasets, baseline comparisons encompassing both dynamic and heuristic methods, and the implementation details required for reproducibility.

A. Datasets and Evaluation Metrics

To evaluate the generalization capability of the proposed method across diverse linguistic challenges, five representative datasets from the GLUE benchmark are utilized. These tasks encompass distinct aspects of Natural Language Understanding (NLU):

- **SST-2 (Stanford Sentiment Treebank):** A binary sentiment analysis task classifying movie reviews as positive or negative, testing polarity robustness.
- **MNLI (Multi-Genre Natural Language Inference):** A large-scale entailment task requiring the model to classify sentence pairs as entailment, contradiction, or neutral, stressing reasoning across varied contexts.
- **QQP (Quora Question Pairs):** A semantic similarity task to determine if two questions are paraphrases of each other, evaluating capability on paraphrase detection.
- **RTE (Recognizing Textual Entailment):** A smaller dataset for textual entailment, serving as a test for performance in low-resource scenarios, assessing adaptability with limited data.
- **MRPC (Microsoft Research Paraphrase Corpus):** A paraphrase detection task based on news articles, exploring the model's sensitivity to paraphrase nuances.

Accuracy is reported as the performance metric. To address the stochastic nature of NAS and fine-tuning, all results for LoRA-NAS and baselines are averaged across five independent runs. The mean accuracy and standard deviation are provided.

B. Baselines and Reference Methods

We used LLaMA-3-8B as the base LLM, which has 32-layers. To rigorously evaluate the contribution of LoRA-NAS, it is compared against a set of baselines, progressing from static configurations to state-of-the-art dynamic allocation methods. This comparison hierarchy starts with simpler, less adaptive methods and moves towards more complex, highly adaptive techniques, providing a comprehensive assessment of LoRA-NAS's efficacy:

- **LLaMA-Zero-Shot (Zero-Shot):** The pre-trained LLaMA-3-8B model evaluated directly on the tasks without any gradient updates, establishing a lower bound for performance.

- **Standard LoRA (Uniform):** A standard fine-tuning configuration in which adapters are inserted uniformly across all query and value projections in every layer. The rank is fixed at $r = 16$ and $\alpha = 32$. This configuration represents the standard default setup.
- **AdaLoRA (Dynamic Rank Allocation):** This dynamic baseline adapts the rank of each module based on importance scores computed via SVD. AdaLoRA prunes the rank of less sensitive modules during training, leveraging SVD to ensure that parameters are allocated efficiently. This approach favors parameter efficiency by reducing unnecessary computed values, potentially leading to improved performance. For fair comparison, AdaLoRA is configured with an initial rank of $r = 64$ and a target average rank of $r = 8$. This comparison evaluates whether AdaLoRA is superior to LoRA-NAS.
- **LoNAS (Rank-Search NAS):** This NAS-based method employs an evolutionary algorithm to optimize the adapter configuration. LoNAS operates on a distinct search space, searching for the optimal rank $r_l \in \{4, \ldots, 64\}$ for each specific layer l, while typically maintaining a dense topology with adapters present in all layers. Comparison with LoNAS isolates the impact of determining where to adapt against determining how much capacity to allocate, as in LoNAS.

The evolutionary search is conducted with a population size of $P = 20$ individuals evolved over $G = 30$ generations. To accelerate fitness estimation, candidates are trained for 1 epoch on a sub-sampled dataset containing 1% of the training data. We set the crossover probability to 1.00 and the mutation probability to 0.20. The search explores any combination of the 32 layers. The parametric search explores ranks $r \in \{4, \ldots, 64\}$ and scaling factors $\alpha \in \{4, \ldots, 64\}$.

The best architecture discovered by the search is fully fine-tuned on the complete dataset. All experiments were conducted on workstations equipped with an NVIDIA A4000 GPU. The Hugging Face *peft* library was used to implement LoRA and AdaLoRA baselines.

V. Experimental Results and Discussion

In this section, we present the quantitative evaluation of LoRA-NAS against the baselines. We analyze the impact of structural search on model performance, efficiency, and architectural interpretability.

A. Performance Comparison on GLUE Benchmark

Table I summarizes the accuracy of LoRA-NAS compared to Zero-Shot, Standard LoRA, AdaLoRA, and LoNAS across five NLU tasks. To ensure statistical robustness, the results for LoRA-NAS and the trainable baselines are reported as the mean and standard deviation over five independent runs.

Standard LoRA, when fine-tuned uniformly, significantly improves performance compared to the Zero-Shot baseline, confirming the necessity of adaptation. However, LoRA-NAS consistently surpasses the Standard LoRA baseline across all datasets, achieving an average accuracy improvement of **+3.34%**.

LoRA-NAS outperforms the dynamic baselines. Although AdaLoRA improves upon standard LoRA by dynamically allocating the parameter budget using SVD, it does not match the performance of LoRA-NAS (e.g., 91.40% vs. 89.13% on MNLI). These results support the core hypothesis that optimizing the spatial placement of adapters leads to better generalization than optimizing only the rank matrix dimensions. Furthermore, compared to LoNAS, which utilizes a similar evolutionary search but focuses on rank allocation, LoRA-NAS achieves superior results, particularly on smaller datasets such as RTE (+1.39%). This finding indicates that pruning entire layers serves as a more effective regularizer than rank reduction and helps prevent overfitting in low-resource scenarios.

B. Stability and Statistical Significance

To address concerns regarding the stochastic nature of NAS, we analyzed the variance of the search process. As shown in the standard deviations reported in Table I, LoRA-NAS exhibits high stability, noted by a low variance. This indicates that the evolutionary algorithm consistently converges to high-performing configurations.

C. Architectural Efficiency

Beyond accuracy, we evaluated the computational efficiency of the resulting architectures. Table II presents the percentage of active layers and the corresponding parameter reduction relative to the standard uniform configuration.

LoRA-NAS discovers highly sparse architectures. For simpler tasks like SST-2, the search converged to utilizing only 12.5% of the available layers, resulting in a significant reduction in trainable parameters. Unlike AdaLoRA, which typically maintains small rank matrices across all layers, our method only considered LoRA adapters in active layers.

VI. Conclusion

This paper introduced LoRA-NAS, a novel framework that reconceptualizes Parameter-Efficient Fine-Tuning as a search problem. Unlike existing methods that rely on uniform placement heuristics or that exclusively emphasize rank optimization, LoRA-NAS uses an evolutionary algorithm to autonomously determine the optimal subset of layers for adaptation.

Extensive empirical evaluation on the GLUE benchmark demonstrates that the location of adaptation is more influential than the total capacity allocated. LoRA-NAS consistently outperformed standard baselines and achieved state-of-the-art results relative to dynamic methods such as AdaLoRA and LoNAS. These findings suggest that structural sparsity, achieved by selectively adapting specific modules while freezing others, yields superior regularization, particularly in low-resource tasks.

While our low-fidelity proxy search achieved state-of-the-art results, it depended on the empirical assumption of rank

TABLE I
COMPARATIVE PERFORMANCE (ACCURACY) ON GLUE TASKS. WE REPORT MEAN ± STANDARD DEVIATION OVER FIVE INDEPENDENT RUNS. **BOLD** INDICATES THE BEST PERFORMANCE. THE *Strategy* COLUMN DENOTES THE PRIMARY OPTIMIZATION MECHANISM.

Method	SST-2	MNLI	QQP	RTE	MRPC
LLaMA-ZSC	90.14	47.13	40.61	52.71	68.40
Standard LoRA	$96.25_{\pm 0.2}$	$86.99_{\pm 0.3}$	$88.02_{\pm 0.1}$	$85.20_{\pm 0.5}$	$87.50_{\pm 0.4}$
AdaLoRA	$96.44_{\pm 0.3}$	$89.13_{\pm 0.1}$	$89.75_{\pm 0.3}$	$88.15_{\pm 0.4}$	$88.25_{\pm 0.3}$
LoNAS	$96.85_{\pm 0.2}$	$90.50_{\pm 0.3}$	$90.10_{\pm 0.2}$	$89.95_{\pm 0.3}$	$89.10_{\pm 0.4}$
LoRA-NAS	$\mathbf{97.25}_{\pm 0.2}$	$\mathbf{91.40}_{\pm 0.3}$	$\mathbf{90.69}_{\pm 0.1}$	$\mathbf{91.34}_{\pm 0.3}$	$\mathbf{89.95}_{\pm 0.3}$

TABLE II
ARCHITECTURAL SPARSITY AND EFFICIENCY OF LORA-NAS COMPARED TO STANDARD LORA.

Dataset	Active Layers	Param. Reduction
SST-2	4 / 32 (12.5%)	87.5%
MNLI	18 / 32 (56.2%)	43.8%
RTE	6 / 32 (18.7%)	81.3%

preservation between the 1% proxy and the full dataset. In future work, we will rigorously quantify this relationship using Kendall's rank correlation coefficients to identify the minimal data fraction needed for reliable topological search. This analysis may further reduce search costs. Moreover, we focused on the LLaMA-3 model; however, it remains an open question whether the obtained configurations transfer across different model architectures, such as BERT. Future research will also examine whether optimal LoRA structures identified for high-resource tasks (e.g., MNLI) can be effectively transferred to low-resource tasks (e.g., RTE) without repeating the search process, thereby reducing computational overhead for related downstream applications.

ACKNOWLEDGMENT

This work was partially supported by project CBF-2023-2024-2797 from the Secretaría de Ciencia, Tecnología e Innovación.

REFERENCES

[1] A. M. Llama Team, "The llama 3 herd of models," 2024. [Online]. Available: https://arxiv.org/abs/2407.21783

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

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 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)*, J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423/

[4] N. Ding, Y. Qin, G. Yang, F. Wei, Z. Yang, Y. Su, S. Hu, Y. Chen, C.-M. Chan, W. Chen, J. Yi, W. Zhao, X. Wang, Z. Liu, H.-T. Zheng, J. Chen, Y. Liu, J. Tang, J. Li, and M. Sun, "Parameter-efficient fine-tuning of large-scale pre-trained language models," *Nature Machine Intelligence*, vol. 5, no. 3, pp. 220–235, Mar 2023. [Online]. Available: https://doi.org/10.1038/s42256-023-00626-4

[5] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "LoRA: Low-rank adaptation of large language models." *ICLR*, vol. 1, no. 2, p. 3, 2022.

[6] A. I. Nowak, O.-B. Mercea, A. Arnab, J. Pfeiffer, Y. Dauphin, and U. Evci, "Towards optimal adapter placement for efficient transfer learning," *arXiv preprint arXiv:2410.15858*, 2024.

[7] Q. Zhang, M. Chen, A. Bukharin, P. He, Y. Cheng, W. Chen, and T. Zhao, "Adaptive budget allocation for parameter-efficient fine-tuning," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=lq62uWRJjiY

[8] J. Munoz, J. Yuan, and N. Jain, "Shears: Unstructured sparsity with neural low-rank adapter search," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, 2024, pp. 395–405.

[9] J. P. Munoz, J. Yuan, and N. Jain, "Low-rank adapters meet neural architecture search for llm compression," *ArXiv*, vol. abs/2501.16372, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusID:275932012

[10] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 10 088–10 115.

[11] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for NLP," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 2790–2799. [Online]. Available: https://proceedings.mlr.press/v97/houlsby19a.html

[12] M. Valipour, M. Rezagholizadeh, I. Kobyzev, and A. Ghodsi, "DyLoRA: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation," in *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, A. Vlachos and I. Augenstein, Eds. Dubrovnik, Croatia: Association for Computational Linguistics, May 2023, pp. 3274–3287. [Online]. Available: https://aclanthology.org/2023.eacl-main.239/

[13] N. Lawton, A. Kumar, G. Thattai, A. Galstyan, and G. Ver Steeg, "Neural architecture search for parameter-efficient fine-tuning of large pre-trained language models," in *Findings of the Association for Computational Linguistics: ACL 2023*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 8506–8515.