Layer Duplication in LLMs*

Neo Eyal and **Nachum Dershowitz**
School of Computer Science and AI
Tel Aviv University
Israel
{neoeyal,nachum}@tau.ac.il

Kfir Bar
Efi Arazi School of Computer Science
Reichman University
Herzliya, Israel
kfir.bar@runi.ac.il

Abstract

We investigate the effect of duplicating multihead self-attention layers in large language models (LLMs) across a range of language tasks, with and without fine-tuning. The results demonstrate that duplicating the initial layers once or twice often yields a significant performance boost. Attention analysis uncovered the underlying mechanisms driving the improvement when performing layer duplication. This method enhances LLM capabilities with or without additional training or labeled data.

1 Introduction

Large language models (LLMs) have become a cornerstone of natural language processing (NLP), achieving state-of-the-art results across a wide range of tasks, such as question answering, language generation, and classification. These models, characterized by their deep architectures, large number of parameters, and attention mechanisms (Vaswani et al., 2017), show improvements with increased scale (Kaplan et al., 2020). However, this growth in model size also leads to increased computational training costs, raising challenges in optimizing performance while maintaining efficiency.

Common strategies for enhancing LLMs include data augmentation (Wei et al., 2022), prompt engineering (Reynolds and McDonell, 2021), and scaling models prior to pretraining (Kaplan et al., 2020). In addition, few-shot and in-context learning (Brown et al., 2020) have enabled models to generalize effectively without further training. Another, less explored, technique is layer duplication, a method that modifies the model architecture post-training to improve its performance by increasing its effective depth without changing its learned weights.

Layer duplication is a relatively unexplored method that has shown promise for improving LLM performance both with and without fine-tuning. At the heart of LLMs is the Transformer architecture (Vaswani et al., 2017), which uses stacked multi-head self-attention (MHSA) layers to build contextual understanding. The idea behind layer duplication is to replicate certain MHSA layers in the architecture, enabling the model to pass information through the same transformation multiple times. This reprocessing may allow the model to refine representations more deeply, enhancing its performance on various tasks. Importantly, this approach requires no additional training data and minimal architectural changes, making it especially appealing for scenarios with limited resources.

Despite significant progress in model design and training efficiency, there remains a trade-off between performance gains and computational cost. Increasing model size often results in diminishing returns (Kaplan et al., 2020). To address this, Parameter-Efficient Fine-Tuning (PEFT) techniques (Houlsby et al., 2019), such as Low-Rank Adaptation (LoRA) (Hu et al., 2022), have been developed to reduce fine-tuning costs by adapting only a subset of parameters. Layer duplication offers a complementary strategy: It alters the flow of computation to extract more value from existing weights. This study is motivated by the need to improve LLM performance, with or without fine-tuning.

We hypothesize that layer duplication is particularly effective in NLP models because language understanding may benefit from repeated contextualization over the full sequence, whereas vision tasks, which often rely on localized spatial features, are less likely to gain from repeating the same layer

*Supported in part by the European Research Council (MiDRASH, Project No. 101071829). Views and opinions expressed are, however, those of the authors only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authorities can be held responsible for them.

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 17797–17807
November 4-9, 2025

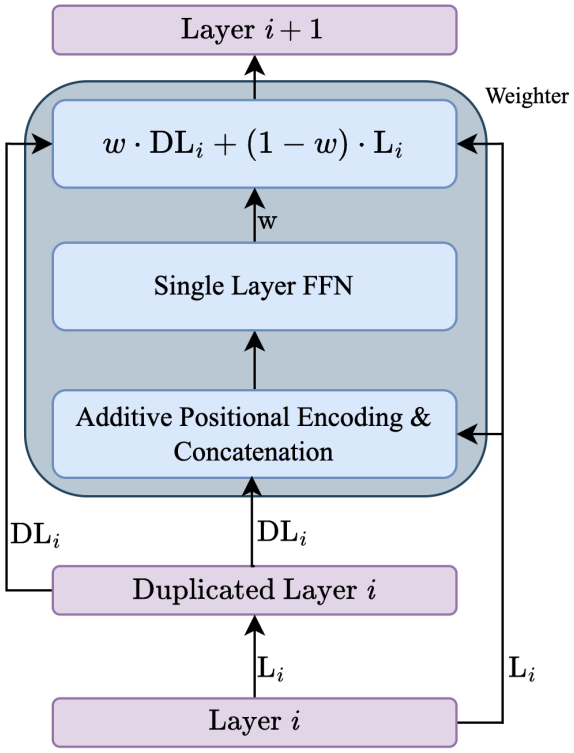

Figure 1: Model architecture of duplicating layer i one time with a Weighter.

multiple times. For this reason, our work focuses on applying layer duplication in LLMS.

Our contributions are as follows: (1) We demonstrate that duplicating one of the initial layers one or two times can be expected to significantly boost LLM performance in both few-shot and fine-tuned settings, and provide theoretical and empirical motivation for why the initial layers are particularly effective to duplicate. (2) We show that performance gains depend more on the task than on model size. (3) We analyze attention patterns and explain how duplicating the first layer alters attention dynamics, contributing to improved downstream performance.

2 Related Work

While increasing model depth and width is a well-established strategy for boosting performance (Petty et al., 2024; Kaplan et al., 2020), the idea of expanding existing pretrained models through layer duplication with or without fine-tuning remain largely unexplored.

Relaxed Recursive Transformers (Bae et al., 2025) and Dynamic Layer Operation (Tan et al., 2024) (DLO) explore layer reuse and vertical scaling, but differ from our method in motivation, complexity, and practicality. Relaxed Recursive Transformers compress models by looping over a fixed block of layers at inference and using LoRA to relax weight sharing. While effective, this adds architectural complexity and degrades performance. DLO scales depth by partitioning a model (e.g., 32 layers into 4 groups of 8) and adding 10 new layers per group, partly initialized from the group's final layer. It manages compute via a routing policy trained during supervised fine-tuning.

In contrast, we do not compress the model or rely on dynamic control. We propose a simple, static, and efficient technique, namely, layer duplication. We empirically and theoretically identify which layers to replicate and how many times, showing strong gains even without fine-tuning. Our approach avoids fixed architectural templates, offering a flexible, low-overhead way to improve existing models, and provides attention-based analyses and theoretical insights into why and where duplication helps.

Another related approach, PonderNet (Banino et al., 2021), introduces adaptive computation by dynamically adjusting the number of processing steps based on task complexity. This method focuses on optimizing computational efficiency by varying the depth of computation as needed. While PonderNet is dynamic and adapts to specific tasks, layer duplication offers a more straightforward, static alternative. By duplicating MHSA layers, the model reuses the learned representations from those layers, allowing it to process information again. This enables the model to build on the contextualized effects of the original layer, further enhancing its ability to make logical connections between words and concepts, especially in tasks requiring reasoning and deep semantic understanding.

Layer skipping is another approach to manipulating a given model's architecture. Techniques such as LayerSkip (Elhoushi et al., 2024), Unified Layer Skipping (Liu et al., 2024), and AdaInfer (Fan et al., 2024) selectively bypass certain layers to improve inference efficiency while trying to best maintain model accuracy. Unlike layer duplication, which aims to enhance model accuracy through duplication of specific layers, these methods focus on identifying and skipping layers that are of less importance.

Unlike prior work that focuses on efficiency or adaptiveness, our approach statically reuses existing layers to enhance performance. This is the first systematic demonstration showing that duplicat-

ing early layers improves LLM performance. We motivate this both empirically and theoretically.

3 Methodology

3.1 Layer Duplication Without Fine-Tuning

To replicate a MHSA layer i, n times, we create a custom model that mirrors the original architecture but replaces layer i with n consecutive copies of it. All other layers are preserved in order, and weights are transferred accordingly to maintain the model's behavior outside the replicated segment.

Layer duplication incurs additional inference-time cost. However, this increase is modest: for instance, duplicating a single layer once results in an inference time increase of up to approximately $1 + 1/\mathrm{N}$, which is often negligible in practice as LLMs have many layers.

3.2 Layer Duplication with Fine-tuning

When performing layer duplication with fine-tuning, we duplicate each layer once. This choice is motivated by results in Section 4.1, which show diminishing returns from multiple duplications. The procedure largely follows Section 3.1, with one key distinction: we insert a lightweight module we refer to as the *Weighter* after the duplicated layer. As shown in Figure 1, the Weighter takes the outputs of the original and duplicated layers and computes a convex combination of them, using either a constant override or a dynamic weight.

The Weighter is a single layer feedforward network (FFN) that first adds learned positional encodings to the original and duplicated layer outputs, then concatenates them. This combined representation is passed through the FFN to compute a dynamic weight for the convex combination. The additive encodings help the Weighter distinguish between the original and duplicated outputs.

The Weighter was used exclusively in the fine-tuning experiments, as it required tuning due to its random initialization. We adopt this approach not only to improve performance, but also to gain insight into duplication behavior by analyzing how much of the duplicated layer the Weighter learns to use.

3.3 Models

We evaluated the impact of layer duplication on models of varying sizes using four different variants of the Pythia models (Biderman et al., 2023): Pythia 70M, Pythia 160M, Pythia 410M, and Pythia 1B.[1] These models were chosen because they span a range of sizes, letting us test if duplication effects hold across scales. Using models from the same family controls for architectural differences.

For the experiments without fine-tuning, we used the models in their base generative form. In this setting, they are especially well-suited for few-shot evaluation, which requires virtually no task-specific data. Instead of relying on labeled datasets, a user can simply provide a couple of examples of the task in the prompt itself, allowing us to assess how layer duplication affects performance without any additional training or data collection.

For the experiments with fine-tuning we used the base model with a randomly initialized classification head.

3.4 Datasets

For the experiments without fine-tuning we decided to work with ten tasks from the BigBench benchmark (Srivastava et al., 2023), which are listed in Table 1.

BigBench is a widely used benchmark, released under the Apache 2.0 license, that provides a diverse set of language tasks, making it an accessible and effective tool for evaluating large language models. The ten tasks were chosen to cover a broad range of reasoning abilities, including logical deduction, causal inference, and analogical reasoning.

By selecting tasks that require different forms of abstraction and contextual understanding, we aim to evaluate whether layer duplication improves model performance across various cognitive challenges while maintaining a manageable runtime. Each of the chosen tasks is in English except *Common Morpheme*, which is in English and French.

For our fine-tuning experiments, we selected English-language tasks that are both large enough to support training and complex enough to benefit from layer duplication. We used the TriviaQA-in-SQuAD-format (Imhof, 2023) and SQuADv2 (Rajpurkar et al., 2018) datasets.[2] They consist of context paragraphs with questions whose answers are found within the given context. The former contains 15.4K English examples, and from the 141K English examples in the latter, we used a 35K sampled subset for computational efficiency.

[1] We used Pythia (Apache license 2.0) as intended by its creators.

[2] The former is released under an unknown license and the latter under a CC-BY-SA-4.0 license.

Task Name	Description	Samples
Analytic Entailment	Identify whether one sentence entails the next	70
Cause and Effect	Answer multiple-choice questions distinguishing cause and effect	153
Boolean Expressions	Evaluate the result of a random Boolean expression	428
Conceptual Combinations	Understand conceptual combinations in appropriate contexts	103
Causal Judgment	Answer questions about causal attribution	190
Analogical Similarity	Identify the type of analogy between two events	323
Common Morpheme	Determine the meaning of the shared morpheme among the given words	50
Logical Deduction	Deduce the order of a sequence of objects	1,500
Logical Sequence	Identify the correct chronological or sequential order of items in a list	39
Odd One Out	Spot the word that does not belong in the group (semantically or grammatically)	86

Table 1: BigBench tasks used in our study, with descriptions and number of samples.

TriviaQA-in-SQuAD-format is a filtered version of TriviaQA (Joshi et al., 2017) in which each answer appears within a single paragraph rather than spanning multiple paragraphs, making it more suitable for our fine-tuning experiments.

3.5 Experimental Settings

We conduct experiments both with and without fine-tuning. The experiments without fine-tuning are designed to reveal the underlying behavior of layer duplication. Specifically, which layers to replicate and how many times to replicate to best enhance model performance when no data for fine-tuning is at hand. The fine-tuning experiment's aim is to build on these findings, allowing us to validate or refute our conclusions, and to provide recommendations for how to effectively apply layer duplication to boost performance when sufficient data is available for fine-tuning.

For the experiments without fine-tuning for each base model, we tested the impact of duplicating each MHSA layer separately with different duplication factors, which determine how many times a specific layer is replicated. The duplication factors tested were 1, 2, 4, 8, and 16. To evaluate the effects of duplication, we first constructed the custom model following the procedure outlined in Section 3.1. We then computed scores for all selected tasks using the evaluation metrics specified in BigBench.[3] The evaluation is deterministic since BigBench's API computes probabilities for the given answer choices rather than generating text with a certain temperature.

For the experiments with fine-tuning we tested the impact of duplicating each MHSA layer separately as stated in Section 3.2 and fine-tuned after the duplication. In fine-tuning we only trained the classification head and the Weighter. For each layer, we fine-tuned three separate models and compared them to the original model fine-tuned without duplication. One used the Weighter, which learns how much of the duplicated layer to use. The other two used fixed weight overrides of 1.0 and 0.5, corresponding to using only the duplicated layer or an equal mix of the duplicated and original layers, respectively.

In both experimental settings, in order to quantify the impact of duplication, we measured the relative score change by comparing the performance of the duplicated models against the original model. In experiments with fine-tuning, we fine-tuned the original model as well as the duplicated model with the Weighter. The relative score change was calculated as:

$$\text{Relative Score Change} = \frac{\text{Duplicated Model Score} - \text{Original Model Score}}{\text{Original Model Score}} \quad (1)$$

This metric allowed us to assess whether duplication led to performance improvements or degradations across different layers and tasks.

Details. All experiments were run on a single A100 GPU. To manage memory effectively, we instantiated one custom model at a time, evaluated it across all tasks, and then released it before proceeding to the next. Evaluating custom Pythia 1B models across the ten tasks without fine-tuning took approximately 20 hours, while smaller models completed in less time. For the fine-tuning experiments, fine-tuning and evaluation of a single Pythia 1B model took around 20 hours per task, with smaller models requiring less time.

[3]We used BigBench's API to calculate the scores as intended by its creators.

4 Results and Analysis

4.1 Layer and Duplication Factor Impact

To better understand the impact of layer duplication, we first visualized the results of the experiments without any fine-tuning to identify key trends in how the duplicated layer and duplication factor influence model performance. The models have differing numbers of layers, so we study how a layer's relative position within the model, rather than its absolute index, impacts performance. We refer to this as the *Layer Percentile*.

From Figure 2, a few clear patterns emerge. Duplicating the initial layers consistently lead to substantial performance improvement, as indicated by a large blue bubble for duplicating layers in the 0–10% range. For example, replicating one of the layers in the that range once yielded an average relative score increase of 8.8%. This suggests that early-stage processing benefits significantly from using pretrained weights multiple times, likely due to the model refining its token representations at these stages. In contrast, middle layers show mixed results, with some layers leading to small positive or negative changes. This indicates that duplication in these layers does not have a uniform effect. Later layers tend to show more negative score changes, suggesting that duplication in these stages can disrupt learned representations rather than enhance them.

The observation that duplicating the initial layers yields the best results is backed up by (Belinkov et al., 2017), which shows that early layers of pretrained models are crucial for downstream task performance, while top layers can be pruned with minimal impact. Additionally, Li et al. (2025) demonstrate that, in architectures like Pythia, gradients tend to diminish as they propagate to deeper layers, making the early layers more effectively trained and more central to model behavior. Furthermore, Belinkov et al. (2017) state that early layers are responsible for extracting low-level features such as morphology and syntax, on which later layers build; if these representations are weak, no deeper computation can fully compensate. These observations provide theoretical motivations as to why duplicating early layers makes for the best impact.

Another key observation is the presence of rapid diminishing returns as the duplication factor increases. While duplicating an early layer one or two times often results in noticeable improvements, further duplication does not yield proportional gains. For example, on the *Odd One Out* task with Pythia 160M, duplicating the first layer once improved performance by 7%, twice by 18%, and four times by 22%, with little gain beyond that. This suggests diminishing returns from excessive duplication.

In this paper we focus on the Pythia family of models, leaving a study of larger models for future work. Nonetheless, indicative experiments on Pythia 6.9B and LLaMA 3.1 8B (Touvron et al., 2023) show that duplicating the first layer once yields average relative improvements of 8.4% and 6.8% in the non-fine-tuned setting. This suggests that our findings generalize to larger models and other families.

We also visualized the relative score change of the fine-tuning experiments. From Figure 3, several important observations become evident. The plot shows three bars for each duplicated layer percentile, representing different methods of computing the convex combination of the original and duplicated layer outputs: one using the Weighter's dynamically computed weight, one with a fixed override weight of 1, and one with a fixed override weight of 0.5. First, duplicating the initial layers before fine-tuning significantly outperforms fine-tuning the original model without any duplication. This holds true when using the Weighter's dynamic weight or a fixed weight override.

Second, fine-tuning a Weighter and using it to control how much of the duplicated layer is used during inference consistently outperforms fixed duplication, regardless of which layer is duplicated. Importantly, duplicating with the Weighter and fine-tuning never harms performance, even when duplicating deeper layers.

Finally, when the override factor is set to 1, we observe a trend consistent with the non-fine-tuned experiments that can be seen in Figure 2. Duplicating early layers improves performance, middle layers have mixed or minimal impact, and duplicating later layers often degrades performance.

Figure 4 shows that during evaluation the averaged Weighter weights are close to 1 when duplicating the initial layers, whereas duplicating deeper layers are assigned much lower average weights. This suggests that the model, in conjunction with the training data, has learned that fully duplicating early layers is most beneficial for performance.

Collectively, these observations reinforce the previous finding and theoretical rationale that duplicating initial layers is the most effective approach.

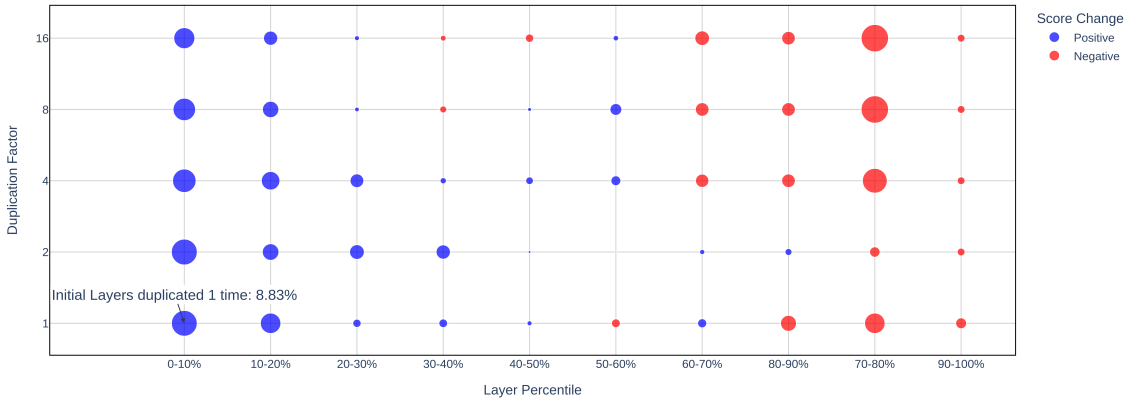

Figure 2: Results of layer duplication without fine-tuning. Bubble size indicates the average relative score change (Eq. 1). Blue represent positive score changes and red, negative ones. Averaged over all models and BigBench tasks.

4.2 Task and Model Size Impact

To further analyze the impact of layer duplication, we examined how score changes varied per task for each model in the experiments without fine-tuning. Since our previous analysis showed that duplicating the first layers yielded the highest performance gains and that duplicating once or twice was the most effective, we visualize the maximum score increase achieved by duplicating either of the first two layers once or twice for each model-task combination.

In Figure 5, one can observe significant score improvements for specific model-task combinations. While some combinations exhibited substantial performance gains, others showed minimal or even negative score changes. Additionally, the results suggest that the effectiveness of layer duplication depends more on the task than on the model size. Certain tasks consistently benefited across models of different scales; others remained largely unaffected. This pattern indicates that some tasks inherently align better with the effects of layer duplication, while others do not experience meaningful improvements, regardless of model size.

While the exact impact can vary depending on the task, this approach proved broadly effective. We recommend that practitioners consider replicating one of the early layers once or twice, as this simple change can lead to substantial improvements. When fine-tuning is possible, adding a Weighter and fine-tuning the model can further enhance results as apposed to fine-tuning without duplication. Overall, this strategy offers a practical and simple way to scale a models size and depth to boost performance.

5 Qualitative Analysis

To further investigate the impact of layer duplication, we analyzed its effects on attention mechanisms. This analyses aims to give further motivation why duplication of early layers improves performance.

5.1 Attention Analysis

Self-attention enables each token in a sequence to consider every other token and assign weights based on relevance. The model uses these weights to combine information from all tokens, generating context-rich representations.

Our hypothesis was that layer duplication would lead to systematic and explainable changes in attention patterns, helping to clarify why performance improved when applying layer duplication. Specifically, we expected that duplication would refine the attention weights distributions in a way that aligns with task-specific improvements—either strengthening relevant connections or reducing the influence of distracting elements. By analyzing the self-attention weight matrices that capture the relevance each token assigns to every other token, we aimed to uncover how the model's self-attention mechanisms adapt after duplicating a MHSA layer.

5.1.1 Experimental Setup

Layer duplication yields notable performance gains across tasks. To understand why, we analyzed attention mechanisms by comparing attention weight matrices from the original Pythia 160M model and

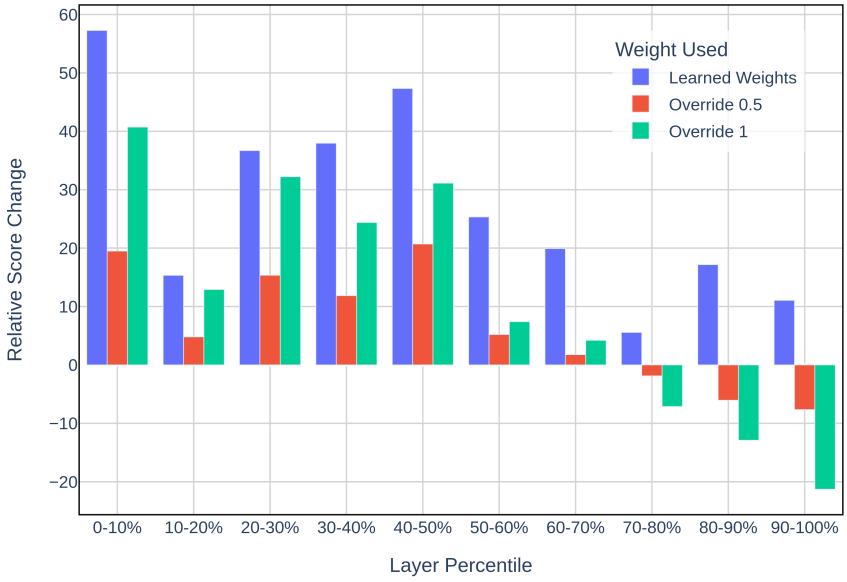

Figure 3: Results of layer duplication with fine-tuning. Averaged over all the models and both fine-tuning tasks.

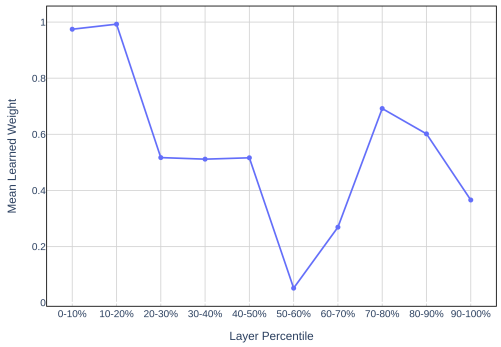

Figure 4: Average learned weights of the Weighter in evaluation of the fine-tuned duplicated model. Averaged over all models and both fine-tuning tasks.

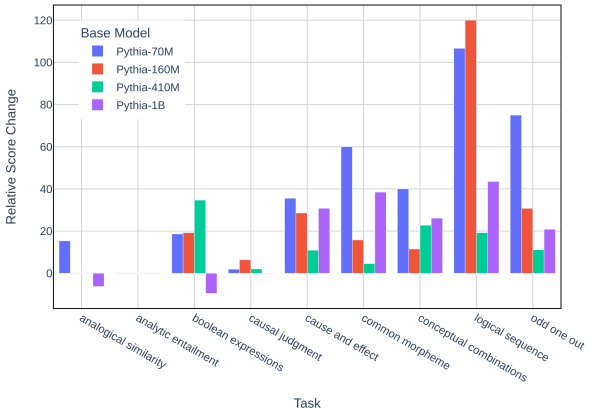

Figure 5: Maximum score improvement per BigBench task for each model, showing the highest effect of duplicating either of the first two layers once or twice.

a version with the first layer duplicated once. This revealed how duplication alters attention behavior.

We focused on the *Odd One Out* and *Common Morpheme* BigBench tasks without fine-tuning. These were selected because the original model performed reasonably well, providing a meaningful baseline. Duplication improved scores from 0.25 to 0.34 on *Odd One Out* and from 0.38 to 0.44 on *Common Morpheme*.

Both tasks have structured formats that support interpretable attention analysis. In *Odd One Out*, the model chooses an outlier from a word list, ideally showing strong intra-group attention and low attention to the odd word. In *Common Morpheme*, the model identifies a shared morpheme across four words, requiring focused attention on those inputs.

We analyzed attention patterns specifically on examples where the duplicated model succeeded but the original failed, to pinpoint how duplication improved performance.

5.1.2 Attention Heatmap Analysis

We visualized attention dynamics using heatmaps that capture the contextualized self-attention among the relevant words in chosen tasks. In these

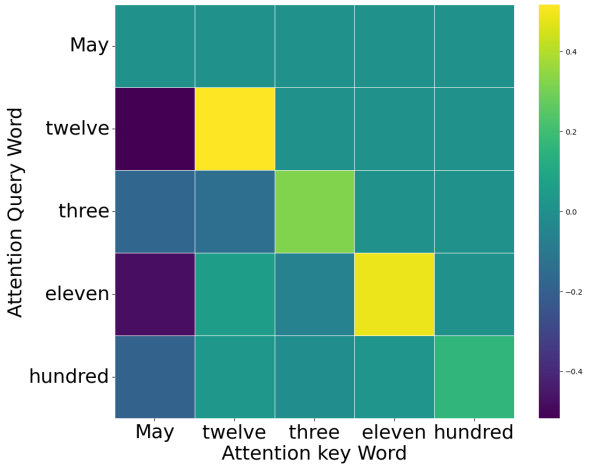

Figure 6: Attention changes on a *Odd One Out* sample where the odd word out is *May*.

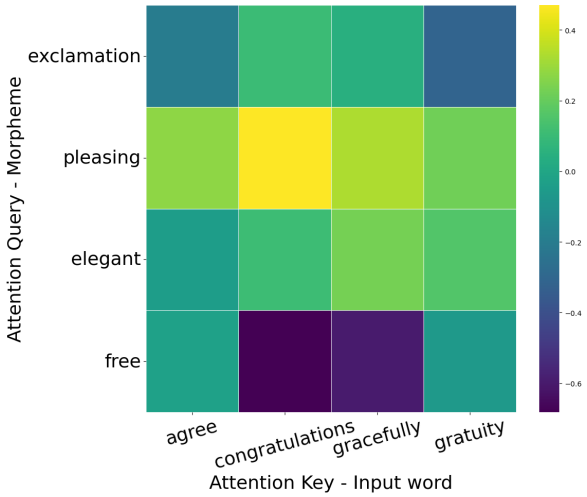

Figure 7: Attention changes on a *Common Morpheme* sample where the common morpheme is *pleasing*.

heatmaps, Heatmap$[i, j]$ quantifies the contribution of the value of word j to the representation of word i. In attention terms, word i acts as the query and word j as the key.

Intuitively, for *Odd One Out*, a well-performing model should exhibit minimal attention contributions from the odd word out to the other grouped words, resulting in a column of low values for that word. Similarly, for *Common Morpheme*, the correct morpheme should exhibit high attention values for its corresponding row.

To evaluate the effect of duplicating the first layer, we analyzed the difference in attention heatmaps between the duplicated model and the original model:

$$\text{Diff\_Heatmap}[i, j] = \text{Heatmap}_{\text{Duplicated}}[i, j] - \text{Heatmap}_{\text{Original}}[i, j]$$

The expectation was that for *Odd One Out* the odd word's column would have negative values, indicating reduced attention, while for *Common Morpheme*, the correct morpheme's row would have positive values, indicating stronger attention.

Empirical results confirmed this expectation across many observed samples. This can be seen in Figures 6 and 7, for example.

5.2 Quantifying Attention Changes

To systematically quantify the observed attention changes that can be seen in Section 5.1.2, we computed aggregated attention metrics across attention heads. For each sample in the data sets, two key values were calculated: $\textbf{Avg}_{\text{correct}}$ and $\textbf{Avg}_{\text{false}}$. The first metric, $\textbf{Avg}_{\text{correct}}$ represent the average attention value associated with the correct word in the difference heatmap. In the *Odd One Out* task, this corresponds to the average attention weight in the column corresponding to the odd word out, effectively capturing the scenario when the odd word out serves as the key. In the *Common Morpheme* task, $\textbf{Avg}_{\text{correct}}$ is computed as the average attention weight in the row corresponding to the common morpheme in the difference heatmap, effectively capturing the scenario where the common morpheme serves as the query. The second metric, $\textbf{Avg}_{\text{false}}$, captures the attention associated with incorrect predictions and is computed similarly to $\textbf{Avg}_{\text{correct}}$, but using the column or row corresponding to the falsely predicted word. These metrics allowed us to systematically analyze how attention changes contributed to improved or degraded model performance following layer duplication.

For each attention head, two bars were plotted to visualize the aggregated attention changes. The first bar represents the average of $\textbf{Avg}_{\text{correct}}$ across all samples where the duplicated model produced the correct prediction while the original model did not. The second represents the average of $\textbf{Avg}_{\text{false}}$ for the same set of samples. These visualizations, shown in Figures 8 and 9, provide insight into how layer duplication alters attention distributions across different tasks.

5.2.1 Observations and Insights

For the *Odd One Out* task, $\textbf{Avg}_{\text{correct}}$ tended to be negative across most attention heads. This indicates that duplicating the first layer reduces the attention weight of the odd word out, when it functions as the key, in forming the contextualized representation

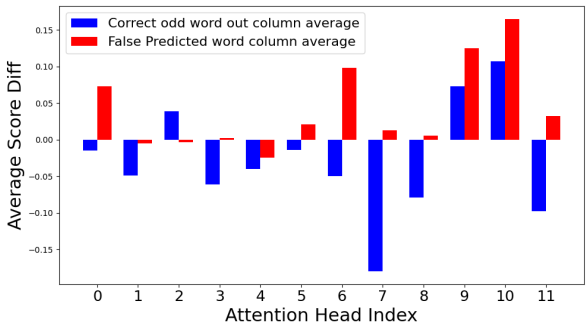

Figure 8: Attention changes on the *Odd One Out* task.

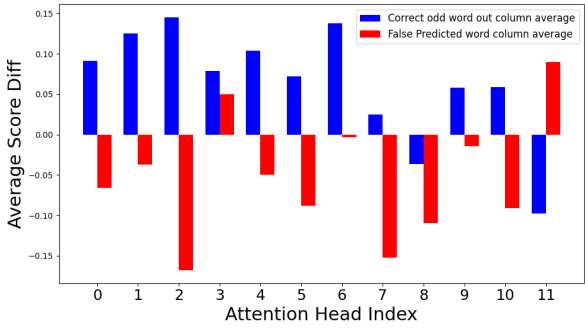

Figure 9: Attention changes on the *Common Morpheme* task.

of the grouped words, which act as the queries. This effectively pushes it farther away from their representations. Conversely, $\textbf{Avg}_{\text{false}}$ tended to be positive, meaning that duplication increased the similarity between the falsely predicted word and the grouped words. In general, $\textbf{Avg}_{\text{false}}$ was greater than $\textbf{Avg}_{\text{correct}}$, reinforcing that layer duplication improved the model's ability to separate the odd word from the grouped words.

For the *Common Morpheme* task, the trends were reversed. Here, $\textbf{Avg}_{\text{correct}}$ tended to be positive, indicating that duplicating the first layer increased the attention weights from the input words (serving as keys) when forming the representation of the common morpheme (acting as the query). Conversely, $\textbf{Avg}_{\text{false}}$ tended to be negative, suggesting that the model reduced its reliance on the input words (keys) when constructing the representation for the falsely predicted morpheme (query).

Across both tasks, the key takeaway is that duplicating the first layer one time improved attention in a task-appropriate manner: reducing similarity where necessary in *Odd One Out* and reinforcing similarity in *Common Morpheme*. This led to better contextual representations and improved performance, giving further intuition to the findings presented in Section 4.

6 Conclusion

We have demonstrated the potential of layer duplication in transformer-based LLMs across few-shot multiple-choice task settings and fine-tuned question answering settings. Our results show that duplicating any of the initial layers once or twice, whether or not additional fine-tuning is applied, can significantly improve model performance. We therefore recommend this strategy to practitioners. Our experiments suggest that this approach has a high likelihood of delivering meaningful performance gains.

Beyond empirical performance gains, we theoretically motivated why duplicating the initial layers is the best approach based upon previous work and the learned weights of the Weighter. Furthermore we showed an attention analysis that revealed that duplication of the first layer modifies attention patterns in a way that enhances contextual understanding. These findings suggest that early layer duplication effectively leverages pretrained weights, improving performance without additional training.

Limitations. Our experiments demonstrate that early layer duplication can improve model performance on certain tasks with or without fine-tuning. We also provided an experimental and theoretical explanation for why this increase occurs. However, there are limitations that should be acknowledged. The primary limitation is that performance improvements are not consistent across all tasks. This suggests that blindly applying layer duplication is not a guaranteed method for enhancing model performance. Instead, practitioners should evaluate their specific use case to determine if layer duplication is beneficial to them. Nevertheless, our findings indicate that there is strong potential for layer duplication to increase performance. In addition if practitioners have data for fine-tuning they can duplicate and fine-tune with a Weighter to boost performance.

Future research could explore the effects of duplicating multiple layers simultaneously or evaluate the application of layer duplication in very large LLMs, now that some of those models are open source.

Risk Assessment. There are no apparent potential risks in this work due to the fact that we use open source datasets and models in order to improve those models' performance.

References

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. 2025. Relaxed recursive transformers: Effective parameter sharing with layer-wise LoRA. In *Proceedings of the Thirteenth International Conference on Learning Representations*. International Conference on Learning Representations.

Andrea Banino, Jan Balaguer, and Charles Blundell. 2021. PonderNet: Learning to ponder. In *Proceedings of the 38th International Conference on Machine Learning Workshop on Automated Machine Learning*.

Yonatan Belinkov, Nadir Durrani, Fahim Dalvi, Hassan Sajjad, and James Glass. 2017. What do neural machine translation models learn about morphology? In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 861–872, Vancouver, Canada. Association for Computational Linguistics.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, Usvsn Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 2397–2430. PMLR.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. 2024. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12622–12642, Bangkok, Thailand. Association for Computational Linguistics.

Siqi Fan, Xin Jiang, Xiang Li, Xuying Meng, Peng Han, Shuo Shang, Aixin Sun, Yequan Wang, and Zhongyuan Wang. 2024. Not all layers of LLMs are necessary during inference. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024)*.

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*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

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

Timo Imhof. 2023. TriviaQA in SQuAD format. `https://huggingface.co/datasets/TimoImhof/TriviaQA-in-SQuAD-format`.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

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. *arXiv preprint arXiv:2001.08361*.

Pengxiang Li, Lu Yin, and Shiwei Liu. 2025. Mix-LN: Unleashing the power of deeper layers by combining Pre-LN and Post-LN. In *Proceedings of the International Conference on Learning Representations*.

Yijin Liu, Fandong Meng, and Jie Zhou. 2024. Accelerating inference in large language models with a unified layer skipping strategy. *arXiv preprint arXiv:2404.06954*.

Jackson Petty, Sjoerd van Steenkiste, Ishita Dasgupta, Fei Sha, Dan Garrette, and Tal Linzen. 2024. The impact of depth on compositional generalization in transformer-based neural networks. *arXiv preprint arXiv:2310.19956*.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics.

Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, New York, NY. Association for Computing Machinery.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, and Adrià Garriga-Alonso. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023(5):1–95.

Zhen Tan, Daize Dong, Xinyu Zhao, Jie Peng, Yu Cheng, and Tianlong Chen. 2024. Dlo: Dynamic layer operation for efficient vertical scaling of LLMs. *arXiv preprint arXiv:2407.11030*.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*.