Transplanting Knowledge: A Study on Layer-Specific Grafting in LLMs

Bastien Zimmermann[1], **Matthieu Boussard**[1]

[1]craft ai
{bastien.zimmermann, matthieu.boussard}@craft.ai

Abstract

This paper introduces *layer-specific grafting*, a novel approach for transferring specialized capabilities between Large Language Models (LLMs). We specialize individual layers by fine-tuning on a task and grafting them into a host model. Using modular addition as a test case, we demonstrate the feasibility of grafting task-specific functionality while evaluating the preservation of the host's general language generation capabilities. To ensure smooth integration, we employ translator modules to align residual streams between models. Experiments reveal key factors influencing grafting success, including layer selection, dataset alignment, and model size. While the method effectively transfers task adherence, challenges persist in achieving strict format retention and fluency. These findings establish layer grafting as a promising tool for modular and efficient AI system development.

Introduction

Model grafting, the integration of components from one Large Language Model (LLM) into another to transfer specific capabilities, presents a promising approach for enhancing model capabilities and advancing our understanding of neural network modularity and interpretability (Yang et al. 2024). By selectively transplanting layers of an LLM into another, we can gain deeper insights into how individual components contribute to the model's overall behaviour.

Furthermore, layer-grafting seeks to facilitate the reuse of existing model capacities without extensive retraining, which is crucial for efficient AI system development and deployment. Transformer-based architectures, despite their outstanding performance across various fields, often incur significant computational costs (Vaswani et al. 2023; Brown et al. 2020). Model grafting presents a solution by allowing the incorporation of pre-trained neural network components to improve performance and accelerate training processes.

However, the success of model grafting depends on multiple factors which have not been yet clearly identified, such as host-donor model compatibility, integration methods or the characteristics of the data used during grafting. A systematic investigation into these variables is critical to making grafting procedures viable and understanding their potential limitations. This paper aims to identify and analyse key factors influencing the success of model grafting. Through this study, we expect to:

1. Demonstrate the feasibility of transferring specific capabilities between models through layer grafting.
2. Identify viable strategies for model grafting, including the selection of host models, loss computation techniques, and translator module configurations.
3. Provide insights into the modularity of neural networks, contributing to the broader understanding of model interpretability and efficient AI system design.

Related Work

Several fields of research explore the potential to combine existing models to improve the overall performance.

Model Stitching

Lenc and Vedaldi (2015) introduce the stitching method to assess whether different models capture the same visual information. By combining representations from one model with those of another, stitching the beginning of a model to the end of another, the stitched model's performance on tasks such as object classification or detection can be evaluated. If performance remains consistent, it suggests that both models encode similar information. This evaluation identifies interchangeable neural network layers and the visual properties encoded by each. The stitching method provides a framework for comparing how representations from different models align.

Pan, Cai, and Zhuang (2023) propose a modular stitching approach to combine pre-trained models, optimizing networks for varying tasks or computational constraints without sacrificing performance. Similarly, Yang et al. (2022) develop a framework to reassemble pre-trained layers or modules into task-specific networks. This reduces training time, enhances flexibility, and improves performance, especially for resource-constrained applications.

Model Merging

Model merging combines multiple pre-trained models to enhance capabilities, leveraging their complementary strengths. However, instead of concatenating parts of models, as in stitching, this field merges model weights.

Matena and Raffel (2022) introduced techniques using Fisher information matrices to prioritize critical parameters during merging, preserving essential learned knowledge while consolidating models. Yang et al. (2024) proposed LLM-Augmented LLMs, which create versatile models by combining specialized components, enabling modular AI development for specific tasks or domains. Liu et al. (2023) presented composable language models, selectively integrating pre-trained model components to build task-specific architectures, maximizing resource use and flexibility.

These studies demonstrate the potential of model merging to create efficient, adaptable, domain-specific architectures. Current efforts aim to refine merging strategies and enhance model compatibility to broaden applications.

Experimental Design

Modular Addition as a Fine-Tuning Task and Metrics

We select *modular addition*, defined as computing $a + b \mod n$, as the fine-tuning task due to its simplicity and the ease of evaluating correctness. For the sake of simplicity, we fix $n = 4$. This task provides a clear framework for assessing the model's ability to learn new, well-defined information. Fine-tuning large language models on modular addition enables us to:

1. **Evaluate Response Format Learning**: Determine if the model can produce answers in the correct syntactic format required for modular arithmetic problems.
2. **Assess Factual Knowledge Acquisition**: Quantitatively evaluate the model's ability to compute accurate modular sums, providing a clear metric for fine-tuning effectiveness.
3. **Monitor Language Generation Integrity**: Observe any degradation in the model's general language capabilities due to specialized fine-tuning by comparing performance on standard language tasks before and after training.

To evaluate the outputs of modular addition fine-tuning, we use three key metrics:

1. **Factual**: Checks whether the first number in the output (positive or negative) matches the exact answer. 2. **Format Hard**: Verifies strict adherence to the modular addition dataset format: *"The result is x"*. 3. **Format Soft**: Ensures the output contains a single numerical value.

Those three metrics reflect different levels of learning: Conforming to the task (outputting a number), meeting the expectations (format hard) and achieving the task (factual). They are evaluated as % of success over 1000 points of the fine-tuning test dataset. Higher scores indicate better performance. Modular addition provides a controlled environment to investigate the effects of fine-tuning at a specific layer while exploring the feasibility and implications of layer-specific learning for model grafting.

Layer-Specific Learning for Grafting

To enable model grafting, we fine-tune a specific layer within the LLM by restricting weight updates exclusively to that layer. This approach localizes the modular addition capability within a single layer, making it possible to extract and transfer it independently to another model for studying task transferability. For a model *pythia-70m* (Biderman et al. 2023) all but the last layer were able to reach good performance through LoRA fine-tuning (Hu et al. 2021). Achieving perfect format-hard and format-soft metrics, with a factual accuracy being $\sim 25\%$ which is random given the modulo 4 task. This imperfection is not a problem, we can see fine-tuning as a mean to steer the model's behaviour. The learned capacity being recognizing a pattern and correctly adapting (adopting the expected format) to it.

Grafting Procedure

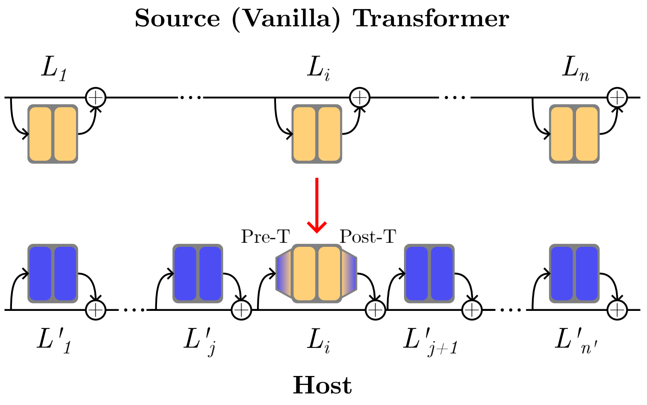

Figure 1: Grafting procedure between a Vanilla model and a host model

The objective of the grafting procedure is to transfer a learned task contained in a layer, while preserving the host model's original capabilities.

We first take a vanilla model (M_{vanilla}) and fine-tune layer L_i on modular addition, which becomes L_{tuned}. Our grafting procedure first integrates L_i from the M_{vanilla} into a host model M_{host} at position $j+1$. To ensure seamless integration, we introduce translator modules before and after the inserted block (Fig.1). The translators are modules composed of a single linear layer, initialized as an identity transformation. This component is inspired from Belrose et al. (2023) which "translates" representations from the basis used at one layer of the network to the basis expected at the final layer. Here, we translate from one layer of one network to one of another network. With the translators, we call this model M_{grafted}.

The pre-translator *Pre-T* adapts the activations from M_{host} to match the input space of L_i, while the post-translator *Post-T* aligns the output of L_i with the subsequent layers of M_{host}. The training is done over the Fineweb dataset (Penedo et al. 2024). In other words, we ensure for the grafted block to receive a similar residual stream through the pre-translator as it was in the vanilla model, as the post-translator with the residual expected in M_{host}.

1. **Pre-Translator Training**: Freeze *Post-T* and all layers of M_{grafted}. Train *Pre-T* to minimize the discrepancy between the inputs of L_i in M_{vanilla} and the corresponding outputs in *Pre-T* in M_{grafted}.

2. **Post-Translator Training**: Freeze $Pre\text{-}T$ and all preceding layers. Train $Post\text{-}T$ to minimize the difference between the outputs of $Post\text{-}T$ and the inputs of block L'_{j+1} in M_{host}.

The loss is expressed using the KL divergence between the two streams. The KL divergence measures how much information is lost when approximating the reference distribution with the new distribution.

After training, M_{grafted} effectively incorporates the modular addition capability by exchanging L_i with L_{tuned} the fine-tuned block. For our experiments, we use *pythia-70m* as vanilla model and *pythia-70m-deduped* as host model. They share the same architecture but with different weights. The small scale of those models allows iterating the different experiments without a surge in computation cost.

Experiments

In this section, we evaluate the grafting procedure across multiple scenarios. Each subsection details the experimental setup, methodology, and results. We evaluate the grafting procedure using two key metrics: perplexity and masked language model predictability.

Perplexity Metric Perplexity measures lexical diversity and unpredictability in text. For a sequence $\mathbf{S} = (w_1, w_2, \ldots, w_N)$ with N words, the empirical probability of each word w is $p(w) = \frac{\text{count}(w)}{N}$, where $\text{count}(w)$ is the frequency of w. The entropy H of the text is then computed over the text sequence:

$$H = -\sum_{w \in S} p(w) \log_2 p(w).$$

and perplexity $P = 2^H$.

This metric represents the average uncertainty per word in the text, considering both word frequency and distribution. A higher perplexity indicates greater lexical diversity and unpredictability within the text sample.

Masked Language Model (MLM) Metric We assess text naturalness and predictability using a pre-trained MLM (BERT-base-uncased). Given a text sequence $\mathbf{S} = (w_1, w_2, \ldots, w_N)$, we compute the predictability score as:

1. **Masking**: Replace each w_i with `[MASK]` to form $\mathbf{w}^{(i)} = (w_1, \ldots, w_{i-1}, \texttt{[MASK]}, w_{i+1}, \ldots, w_N)$. 2. **Prediction**: Use the MLM to predict w_i and record the probability $p_\theta(w_i \mid \mathbf{S}^{(i)})$. 3. **Scoring**: Average probabilities where the original word is among top predictions:

$$P = \frac{1}{M} \sum_{i=1}^{N} \delta_i \, p_\theta(w_i \mid \mathbf{w}^{(i)}),$$

where $\delta_i = 1$ if w_i is retrieved, and M is the total number of successful predictions. A higher P indicates greater fluency and alignment with the MLM's patterns. The higher, the better. Those metrics are evaluated over 1000 points of the fineweb dataset, truncated to generate the end of its sentences.

Acronyms: f_H (format hard mean), f_S (format soft mean), $fact$ (factual mean), P (mean perplexity), MLM (mean Masked Language Model metric).

Loss Computation Techniques

This experiment examines the impact of computing grafting loss on model ouptu logits versus residuals (cf). Instead of carefully aligning the proper residual streams, the logits approach intends to match the logits of M_{grafted} and M_{vanilla} for the training of $Pre\text{-}T$ and those of M_{grafted} and M_{host} for the training of $Post\text{-}T$.

task	Tuned	residual	logits	Vanilla
f_H	**1.000**	0.000	0.000	0.000
f_S	**1.000**	0.645	0.012	0.000
$fact$	0.227	**0.233**	0.046	0.000

Table 1: Modular addition evaluation

task	Tuned	residual	logits	Vanilla
P	1.284	1.236	1.370	**1.230**
MLM	0.359	0.022	0.012	**0.749**

Table 2: Language generation evaluation

Training translators using residual streams is more efficient, it avoids relying on downstream model components.

In the modular addition evaluation (Tab. 1), the tuned model excels in both strict and soft format adherence, while residual-based grafting retains soft format capability but fails in strict format adherence. Logits-based grafting and the vanilla model perform poorly across both metrics. Residual-based grafting slightly outperforms the tuned model in factual accuracy, effectively transferring task-specific knowledge.

For language generation (Tab. 2), the vanilla model demonstrates the best fluency and predictability, followed closely by residual-based grafting in fluency. Logits-based grafting and the tuned model underperform across all metrics.

Influence of Destination Layer

Does the graft destination layer (position where the block is placed) impact the resulting performance?

task	L_1	L_2	L_3	L_4	L_5
f_H	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
f_S	0.000	0.001	0.000	**0.723**	0.000
$fact$	0.000	0.038	0.000	**0.233**	0.000

Table 3: Modular addition evaluation

task	L_1	L_2	L_3	L_4	L_5
P	**1.006**	1.290	1.301	1.281	1.055
MLM	0.000	**0.121**	0.054	0.042	0.010

Table 4: Language generation evaluation

For modular addition (Tab. 3), destination layer L_4 stands out, achieving the best soft format adherence and factual accuracy, while all other layers show negligible performance. Strict format adherence is not retained across any layer.

In language generation (Tab. 4), destination layer L_1 (second *pythia-70m* layer) exhibits the best fluency with the lowest perplexity, followed by layer L_5 the last layer. However, destination layer L_2 achieves the highest predictability, indicating better alignment with natural language patterns.

Role of Translator Training Data

How does the choice of dataset for training translator models affect grafting efficacy? The alternatives are: the fine-tuning dataset (Mod), the Fineweb dataset (Fineweb) and a mix of boths: (Mixed).

task	Tuned	Mod	Mixed	Fineweb	Vanilla
f_H	**1.000**	0.000	0.000	0.000	0.000
f_S	**1.000**	0.910	0.706	0.639	0.000
$fact$	0.227	**0.233**	**0.233**	**0.233**	0.000

Table 5: Modular addition evaluation

task	Tuned	Mod	Mixed	Fineweb	Vanilla
P	1.284	1.343	1.305	1.337	**1.231**
MLM	0.359	0.029	0.104	0.024	**0.749**

Table 6: Language generation evaluation

All dataloaders allow some transfer of the fine-tuned task (Tab. 5, Mod dataloader performs slightly better. In language generation (Tab. 6), none of the dataloaders approach the performance of the vanilla model.

Comparison of Grafting Strategies

What would be the effect of directly merging the tuned block instead of first merging the vanilla block? We named this alternative strategy $direct$ and the original $original$.

task	Tuned	original	direct	Vanilla
f_H	**1.000**	0.000	0.000	0.000
f_S	**1.000**	0.706	0.000	0.000
$fact$	0.227	**0.233**	0.000	0.000

Table 7: Modular addition evaluation

task	Tuned	original	direct	Vanilla
P	1.284	1.305	**1.087**	1.231
MLM	0.359	0.104	0.138	**0.749**

Table 8: Language generation evaluation

The direct grafting approach fails to transfer both strict and soft format adherence effectively (Tab. 7), as well as failing factuality. In language generation, the direct grafting strategy achieves the lowest perplexity (Tab. 8), indicating better fluency compared to the original strategy, though neither comes close to the predictability of the vanilla model.

Comparison of model sizes

	Tuned	70m	160m	Vanilla	410m
f_H	**1.000**	0.000	0.000	0.000	0.000
f_S	**1.000**	0.645	0.245	0.000	0.000
$fact$	0.229	**0.233**	0.027	0.002	0.000

Table 9: Modular addition evaluation

	Tuned	70m	160m	Vanilla	410m
P	1.222	1.236	1.271	**1.198**	1.338
MLM	0.619	0.022	0.111	**0.722**	0.000

Table 10: Language generation evaluation

For modular addition, the 70m model retains moderate soft format adherence and the highest factual accuracy among the grafted models (Tab. 9). Larger models (160m and 410m) fail to transfer strict or soft format capabilities effectively, showing minimal performance across metrics. In language generation (Tab. 10), 70m maintains closer fluency to the vanilla model, while 160m shows slightly better predictability. The 410m model performs poorly in both metrics, highlighting challenges in grafting larger models.

Conclusion

We proposed and systematically analyzed a novel approach for transferring specific capabilities between large language models (LLMs) through layer-specific grafting. By isolating and fine-tuning individual layers for a modular addition task and then integrating these layers into a host model, we demonstrated the feasibility of task-specific transfer while maintaining the host model's broader capabilities. However, the transfer is limited to conforming to the task, no iteration was able to transfer the format-hard capabilities of M_{tuned}. In the evaluation of language generation (conserving the original model capabilities), our metrics are limited. For instance, texts with repetitive patterns (low diversity) will have lower perplexity, which might be interpreted as more predictable or "better", even when such repetition is undesirable in natural language. Better benchmarks exist such as MMLU but are not relevant in our case, as we evaluated *pythia-70m* our baseline as 0 the lowest possible score.

These findings highlight the potential of model grafting as a tool for modular AI development, allowing researchers and practitioners to repurpose and integrate capabilities from pre-trained models. Future work should explore applying this technique to a wider range of tasks and model architectures, while investigating methods to enhance grafting robustness and generalization across diverse domains.

By deepening our understanding of neural network modularity, this study establishes a foundation for more efficient, interpretable, and reusable AI systems, paving the way for scalable and task-specific model development.

References

Belrose, N.; Furman, Z.; Smith, L.; Halawi, D.; Ostrovsky, I.; McKinney, L.; Biderman, S.; and Steinhardt, J. 2023. Eliciting Latent Predictions from Transformers with the Tuned Lens. arXiv:2303.08112.

Biderman, S.; Schoelkopf, H.; Anthony, Q.; Bradley, H.; O'Brien, K.; Hallahan, E.; Khan, M. A.; Purohit, S.; Prashanth, U. S.; Raff, E.; Skowron, A.; Sutawika, L.; and van der Wal, O. 2023. Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling. arXiv:2304.01373.

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D. M.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2021. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685.

Lenc, K.; and Vedaldi, A. 2015. Understanding Image Representations by Measuring their Equivariance and Equivalence. *arXiv preprint arXiv:1411.5908*.

Liu, N.; Li, S.; Du, Y.; Torralba, A.; and Tenenbaum, J. B. 2023. Compositional Visual Generation with Composable Diffusion Models. arXiv:2206.01714.

Matena, M.; and Raffel, C. 2022. Merging Models with Fisher-Weighted Averaging. arXiv:2111.09832.

Pan, Z.; Cai, J.; and Zhuang, B. 2023. Stitchable Neural Networks. *arXiv preprint arXiv:2302.06586*.

Penedo, G.; Kydlíček, H.; allal, L. B.; Lozhkov, A.; Mitchell, M.; Raffel, C.; Werra, L. V.; and Wolf, T. 2024. The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale. arXiv:2406.17557.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2023. Attention Is All You Need. arXiv:1706.03762.

Yang, E.; Shen, L.; Guo, G.; Wang, X.; Cao, X.; Zhang, J.; and Tao, D. 2024. Model Merging in LLMs, MLLMs, and Beyond: Methods, Theories, Applications and Opportunities. arXiv:2408.07666.

Yang, X.; Zhou, D.; Liu, S.; Ye, J.; and Wang, X. 2022. Deep Model Reassembly. *arXiv preprint arXiv:2210.17409*.