NeuronMerge: Merging Models via Functional Neuron Groups

Wangyun Gu[1,2]*, Qianghua Gao[2], Lixin Zhang[3], Xu Shen[2]†, Jieping Ye[2],
[1]Zhejiang University, [2]Alibaba Cloud, [3]Zhejiang Gongshang University,
wangyungu@zju.edu.cn, shenxu.sx@alibaba-inc.com

Abstract

Model merging techniques like task arithmetic, which combines model parameters through weighted averaging, have proven effective. However, the success of task arithmetic relies on the linearity between model weight differences and output feature changes, which is often lacking in conventional fine-tuned models. In this work, we employ neuron description methods to analyze and classify neurons based on their functionalities. We theoretically demonstrate that grouping Multi-Layer Perceptron (MLP) neurons by functionality enhances model linearity. Building on this, we propose a neuron-based task arithmetic merging method that consistently improves performance across various tasks and model scales. Our approach is complementary to existing merging techniques, achieving superior results in merging models fine-tuned on fundamental tasks like Math, Code and Translation.

1 Introduction

In recent years, the scaling up in large language model (LLM) has greatly escalated data requirements and computational expenses for fine-tuning multi-task models. To address this challenge, researchers have explored methods to combine the strengths of existing single-task models through model merging techniques (Yu et al., 2024; Jin et al., 2023; Matena and Raffel, 2022; Yadav et al., 2023). A simple and efficient model merging approach, called task arithmetic (Ilharco et al., 2023) demonstrates remarkable effectiveness in creating a multi-task model through a basic weighted combination of parameters of existing models, without the need for costly retraining processes or additional data collection.

Recent studies (Ortiz-Jiménez et al., 2023; Zhou et al., 2024) introduce a concept *Linearity* which refers to the linear relationship between the differences in model weights and the differences in output features caused by fine-tuning, and reveal its connection with effectiveness of task arithmetic. Studies show that models exhibiting linearity retain their individual task performance better when merged using task arithmetic, leading to superior multi-task models (Ortiz-Jiménez et al., 2023; Tang et al., 2024; Jin et al., 2024; Liu et al., 2024). However, conventionally fine-tuned models often lack this ideal linearity (Ortiz-Jiménez et al., 2023). To address this issue, Dai et al. (2025) proposes a training-free method (SubModule Linear) intuitively by breaking models into multiple shallower submodules (e.g., layers, self-attentions, MLPs) and discovers that these submodules exhibit a level of linearity that significantly surpasses that of the overall model, thus achieves SOTA performance on task arithmetic. But still, according to their analysis, non-linearity in MLPs is much higher than in self-attentions, and intuitively naive attempts to further subdivide self-attentions (Dai et al., 2025) or MLPs Table 3 lead to performance degradation, which suggests further in-depth investigations and finer divisions for MLPs. And the non-linearity in MLPs remains under-explored within the community.

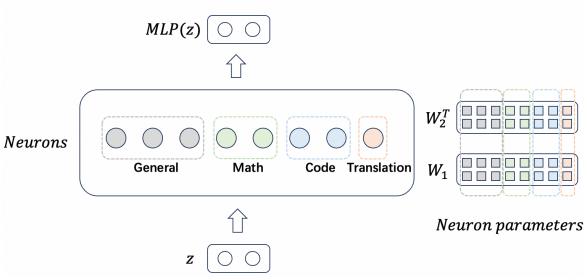


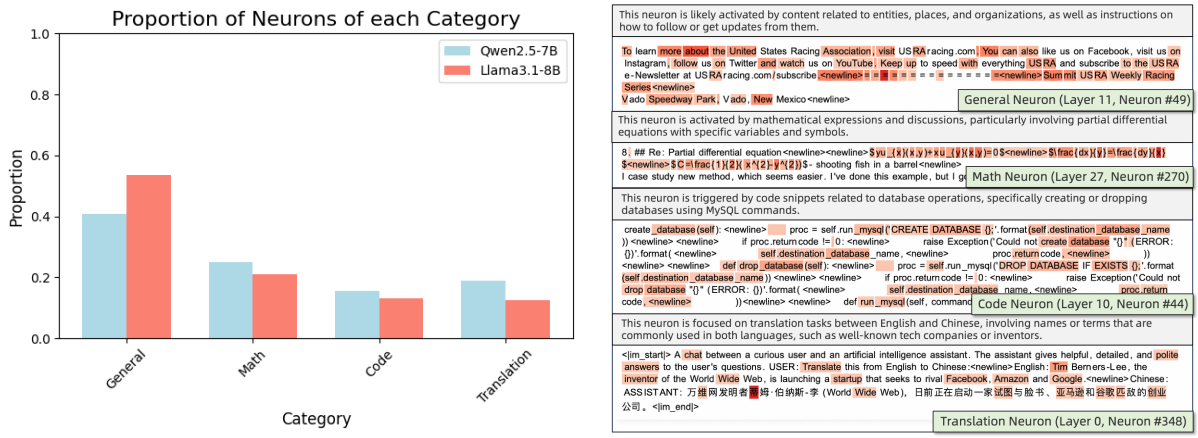
Figure 1: Functional neuron groups: grouping neurons according to neuron functionalities such as *General*, *Math*, *Code* and *Translation*.

The rapid development of automatic neuron description methods in recent years (Bills et al., 2023;

*This work was done when the author was a research intern at Alibaba Cloud.

†Corresponding author.

Findings of the Association for Computational Linguistics: ACL 2025, pages 9015–9037



(a) Proportions of *General*, *Math*, *Code* and *Translation* categories of neurons in Qwen2.5-7B and Llama3.1-8B models.

(b) Cases of functionality descriptions and activation patterns for different categories of neurons in Qwen2.5-7B.

Figure 2: Neurons exhibit a high degree of functionality in Qwen2.5 and Llama-3.1. (a) Proportions of different categories of neurons in Qwen2.5-7B and Llama3.1-8B models are shown, and exhibit similar distributions except that Qwen2.5 has more *Translation*, *Math* and *Code* neurons and less *General* neurons compared to Llama3.1. (b) We show case the a exemplars for each of the four neuron functionalities. And the darkness of the background color indicates the magnitude of neuron activation on that token. The activations demonstrate noteworthy patterns related to specific functionality.

Choi et al., 2024) provides in-depth analysis of elements in MLPs (e.g. neurons), by generating neuron functionality descriptions according to their activation patterns. One of these (Choi et al., 2024) achieves human-expert-level performance, which provides an effective way to classify neurons into groups for further study.

We observe that neurons exhibit a high degree of functionality in both LLM series (shown in Figure 2). Furthermore, theoretical derivation also demonstrates that grouping MLP neurons according to their functionality types (see Figure 1) can improve model linearity (refer to §2.4 and details in §B), and therefore improve the performance of task arithmetic model merging. Inspired by these observations and analysis, in this paper, to further improve the linearity of MLPs for task arithmetic model merging, we propose to grouping the neurons in MLPs by their functionalities (e.g. math, code, translate) and independently merging parameters within each group. However, existing neuron description methods (Bills et al., 2023; Choi et al., 2024), while providing valuable functional descriptions, face significant barriers for direct application in task arithmetic model merging: 1) They focus on natural language explanations rather than actionable neuron functionality classification, and 2) Their computational cost makes large-scale application unaffordable. So firstly, we utilize a modified and more efficient version of the method (Choi et al., 2024) to analyze and classify neurons according to neuron functionality in two SOTA open-sourced LLM series Qwen2.5 (Qwen Team, 2024) and Llama-3.1 (AI@Meta, 2024). Secondly, we merge each group of neurons independently. To be specific, we merge the *General* neurons and drop the parameter differences on other task-specific groups. Our method shows consistent improvements of performance on merging models fine-tuned at several fundamental tasks (Math, Code and Translation) and model scales (7/8/13/14B).

Our contributions are summarized as follows:

- We conduct neuron analysis on SOTA open-sourced LLM series, and find neurons keep a high consistent degree of functionalities on General, Math, Code, Translation before and after fine-tuning.
- We propose a novel model merge method by bridging automatic neuron description methods and finer-granularity task arithmetic. Our method groups neurons according to their functionalities, and conducts task arithmetic for each group independently. We demonstrate its effectiveness both theoretically and experimentally.
- Our method achieves superior performance on different types of foundation models, and is consistently effective for merging different down-stream-task fine-tuned models. More

importantly, our method is orthogonal and complementary to existing model merging methods.

2 Neuron Functionality Analysis

In this section, we systematically investigate neuron functionalities in a base model and its fine-tuning variants. We first introduce the basic definitions of neurons and activation patterns in §2.1. Building on this foundation, we present a modified data-driven framework to classify neurons into task-specific categories by analyzing their activation patterns efficiently, as detailed in §2.2. In §2.3, we validate the framework through several key observations. Finally, in §2.4, we bridge these findings to task arithmetic techniques (Ilharco et al., 2023), demonstrating how neuron-type-aware parameter composition improves *Linearity* for *task vectors*. This structured analysis provides both theoretical insights into neuron functionality and practical guidelines for optimizing task arithmetic through neuron functionality classification.

2.1 Preliminary

Introduction to Neurons. In a transformer model, neurons are part of the MLP layers (Bills et al., 2023). Each neuron computes a weighted sum of its inputs, applies an activation function, and produces an output. The activation function introduces non-linearity, allowing the model to learn more complex relationships in the data.

Neuron Activation. The activation of a neuron refers to the output value produced by the neuron in response to its inputs, and indicates the degree to which the neuron is "firing" or contributing to the overall output of the model. Mathematically, in modern decoder-only architecture LLMs like Llama3.1 and Qwen2.5 the activation of a neuron j can be represented as:

$$\phi_j(z) = \text{SiLU}\left((w_j^1)^\top z\right) \cdot (w_j^2)^\top z, \quad (1)$$

where z is the pre-MLP value of the residual stream after RMS Norm. w_j^1 and w_j^2 are the d-dimensional weight vectors for the neuron, where d is the dimension of residual stream. $SiLU$ is the Swish-like activation function, defined as $SiLU(x) = x \cdot \sigma(x)$, where $\sigma(x)$ is the logistic sigmoid function.

2.2 Classification of Neuron Functionality

Neuron Functionality. The functionality of a neuron is determined by its activation pattern across different input exemplars. By analyzing the activation values, we can identify the types of inputs that strongly activate the neuron and thus infer its functionality. This is often done by selecting the *top k* inputs where the neuron's activation is highest as exemplars and generating descriptions based on these exemplars (Bills et al., 2023).

Classification Method. Inspired by the Transluce (Choi et al., 2024) framework, we adopt a streamlined approach to perform large-scale classification of neuron functionalities. Instead of generating descriptive descriptions, our method categorizes neuron functionality based on the frequency of activation across different corpus exemplars directly.

Let $\mathcal{C} = \{c_1, c_2, \ldots, c_N\}$ denote the set of N distinct categories. For each category c_i, we curate M representative corpus exemplars, resulting in a total of $N \times M$ exemplars. Each exemplar is forwarded by a target LLM, and for each MLP neuron j, we record its activation values across all tokens in the input sentence.

For a given exemplar s belonging to category c_i, and for each token t in s, let $\phi_j(s,t)$ represent the activation value of neuron j. To summarize the neuron's activation for the entire sentence, we define:

$$a_j(s) = \max_t |\phi_j(s,t)|.$$

Here, $a_j(s)$ captures the highest absolute activation value of neuron j across all tokens in exemplar s. This process is applied uniformly across all layers of the LLM.

After computing $a_j(s)$ for all exemplars $s \in \mathcal{S}$, where $\mathcal{S}$ is the complete set of $N \times M$ exemplars, we identify the top-k exemplars that elicit the highest activations for neuron j:

$$\mathcal{S}_k(j) = \text{Top-}k\left(\{a_j(s) \mid s \in \mathcal{S}\}\right).$$

The functionality category of neuron j is assigned based on the most frequent category among the top-k activated exemplars. Formally, the functionality category $\text{Category}(j)$ is determined as:

$$\text{Category}(j) = \arg\max_{c_i \in \mathcal{C}} \left|\{s \in \mathcal{S}_k(j) \mid s \in c_i\}\right|. \quad (2)$$

In this equation, $|\cdot|$ denotes the cardinality of the set, and $\arg\max$ identifies the category c_i with the highest representation within $\mathcal{S}_k(j)$.

By aggregating the maximum activation values of each neuron across a diverse set of categorized

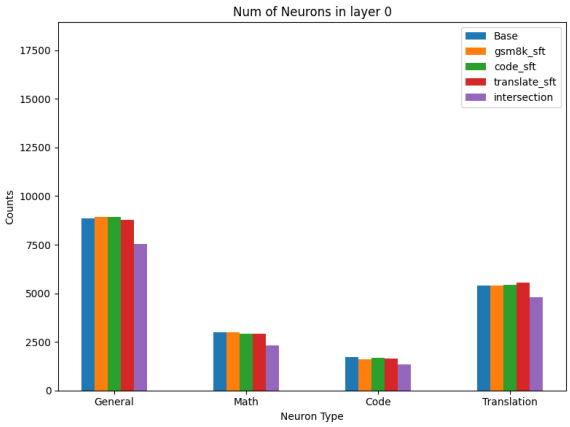


Figure 3: Neuron classification results in layer 0 for Qwen2.5-7B and three fine-tuned models, shown as bar charts for different categories of neurons. *Base* model is Qwen2.5-7B, *gsm8k_sft*, *code_sft*, *translate_sft* are Qwen2.5-7B fine-tuned variants on Math, Code, Translation dataset respectively. Purple bar is the number of intersection neurons for each functionality category.

exemplars and selecting the top-k activations, we effectively classify neuron functionalities based on the predominant category present in these high-activation instances. This method facilitates a scalable and interpretable analysis of neuron functions without relying on direct descriptive generation.

Classification Settings. In practice, we classify neuron functionalities into four distinct categories: *General*, *Math*, *Code*, and *Translation*. And the target LLMs are Qwen2.5-7B and three fine-tuned models based on Qwen2.5-7B trained on three datasets respectively: gsm8k (Cobbe et al., 2021), code alpaca (Chaudhary, 2023) and a zh↔ en translation dataset (Xu et al., 2024a) (for training details, see §4.1). To facilitate this classification, we constructed a corpus comprising a total of $1,200$ exemplars, evenly distributed across the four categories. Specifically, we randomly selected 300 exemplars from each of the following sources. For the general category, we used data from CommonCrawl in RedPajama (Weber et al., 2024). For the mathematics category, we sourced exemplars from Proof-file-2 (Azerbayev et al., 2024). The coding examples were drawn from Python dataset in StarCoder (Li et al., 2023). The translation category utilized the en↔zh dataset from Xu et al. (2024a).

Each text in these exemplars was processed with a maximum input length of 512 tokens, truncating any content that exceeded this limit to ensure uniformity across inputs. To identify sentences that most strongly activate each neuron, we selected the top-k sentences exhibiting the highest activation values, where $k = 10$.

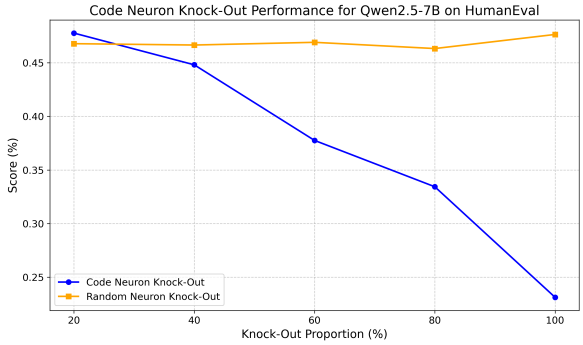


Figure 4: Knock-out code neurons in Qwen2.5-7B (layer 13) will lead to a greater decline in coding ability than random neuron knock-out. The horizontal coordinate indicates the knock-out proportion. Performances are tested on HumanEval (Chen et al., 2021).

2.3 Classification Result.

From Figure 3, we can observe that after fine-tuning, the distribution of neuron functionality categories remains largely consistent across Qwen2.5-7B and its fine-tuned variants, demonstrating high distributional uniformity. Specifically, in all four models, the proportion of neurons overlapping within each category (depicted by the purple bar) is substantially high. This suggests that most neurons retain their functionality types before and after fine-tuning regardless of down-stream tasks, which corresponding with the findings in Wang et al. (2022). The results presented correspond to the classification results of the first layer (Layer 0). Additional classification results across other layers and detailed case studies are available in the §A.1, all of which exhibit similar patterns consistently.

Cosine similarity of neurons provides another evidence that verify the consistency above. We analyzed the cosine similarity between the weight vectors of the positional corresponding neurons in base and fine-tuned models. It revealed that the corresponding neurons exhibit an exceptionally high cosine similarity of up to 0.999 (for details, see §A.3), indicating highly similar parameter patterns. This suggests why neurons preserve their functionalities after fine-tuning, as similar weights lead to similar activations (Jacot et al., 2018) and consistent maximally activated sentences.

To validate the effectiveness of our neuron functionality classification, we performed ablation experiments by selectively deactivating neurons based on their classified categories. Specifically, we knock-out the parameters of neurons associ-

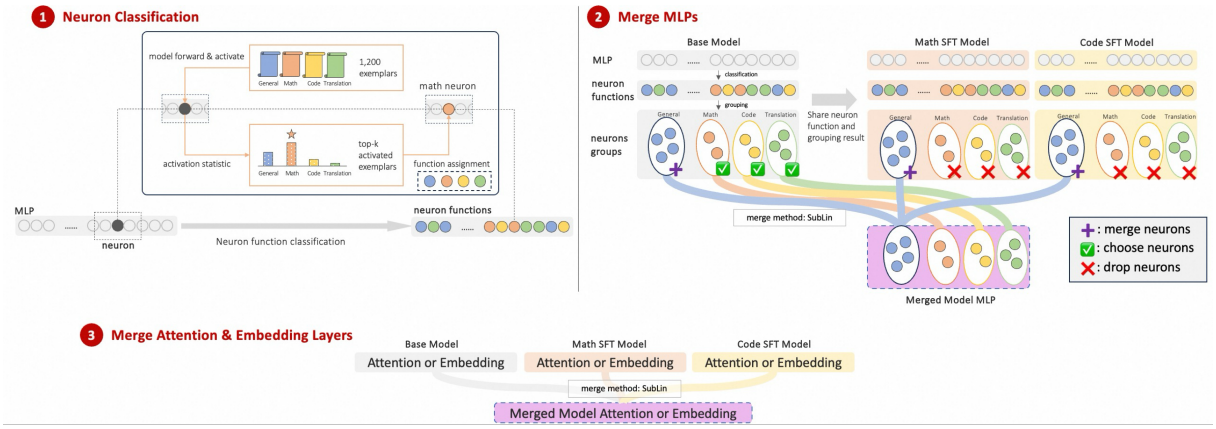


Figure 5: The framework for neuron grouping and merging in LLM. It consists of three main steps: 1) Neuron Grouping Based on Functionality, which classifies neurons into different categories; 2) Merging for MLP Layers, where *General* neurons are merged by SubModule Linearity while others are discarded to focus on the Base model's parameters; and 3) Merging for Attention and Embedding Layers, where different methods can be utilized as complementary approaches including DARE or SubModule Linearity in our main experiments.

ated with different functionalities in the base model in varying proportions $(20\%, 40\%, \cdots, 100\%)$ and evaluated the performance on corresponding tasks. As a control, we also conducted random ablations in which an equivalent number of neurons were randomly deactivated, repeating each random ablation 3 times and averaging the performance. The ablation results, depicted in Figure 4, demonstrate that neuron ablation based on the *Code* category leads to a significant degradation in model performance (tested on HumanEval (Chen et al., 2021)) compared to random ablation. For *Math* and *Translation* categories, the results exhibit consistency, see the §A.2. This stark contrast underscores the relevance of neuron classifications to the model's task performance and reinforces the validity of our classification methodology and indicates that neurons w.r.t. their functions play a key role in model performance for corresponding tasks.

2.4 Task Arithmetic and Linearity

Task arithmetic (Ilharco et al., 2023) is a simple and efficient model merging strategy. It combines model weights by a simple weighted average. This approach gives the merged model multi-task performance without needing extra training or data.

Task vector for task t is defined as the difference between the fine-tuned and the pre-trained weights, namely, $\tau_t = \theta_t - \theta_0$. Task arithmetic involves a linear combination of the task vectors added to the pre-trained model weights θ_0. The merged weights can be expressed as

$$\theta_{merge} = \theta_0 + \sum_{t=1}^{T} \alpha_t \tau_t, \qquad (3)$$

where α_t is the weight corresponding to τ_t.

Linearity (Ortiz-Jiménez et al., 2023) is beneficial for task arithmetic performance. We theoretically investigate the sources of non-linearity arising from MLP in §B. We prove a theorem to estimate the non-linearity with ReLU activation function for simplification. The theorem reveals two primary components contributing to non-linearity: firstly, the second-order terms arising from task vectors, which is negligible under task arithmetic setting; secondly, the change of neuron activation state before and after fine-tuning causes non-linearity.

If we group neurons based on their functionalities—such as *Math* neurons, which activate strongly in response to a Math dataset—their activation states will remain stable since the norm of the task vector $\|\tau\|_2$ is small under task arithmetic setting. So the non-linearity from the second source will be reduced, thereby enhancing the performance of task arithmetic.

3 Neuron Grouping and Merging

In this section, we present our approach for task arithmetic based on neuron functionalities. Our proposed framework, depicted in Figure 5 and Algorithm 1 in Appendix, involves several key steps. Initially, we classify neurons across all layers of the **base model** according to their functionalities and assign them into groups. Subsequently, for MLP layers, we perform a separate task arithmetic for each group of neurons independently. Finally, we address the merging of the remaining parameters (e.g. self-attention and embeddings). In the following sections, we provide a detailed explanation of each step in our methodology.

Neuron Grouping Based on Functionality. Let $D = \{D_t; t = 1, 2, \cdots, T\}$ be a collection of datasets for T tasks, with $\theta_1, ..., \theta_T$ representing the parameters of models fine-tuned from base model θ_0 using the corresponding datasets. The parameters resulting from the merging process are denoted by θ_{merge}.

We conduct a functionality classification of neurons within each MLP layer of the base model θ_0. The detailed methodology for this classification is elaborated in section 2, and we keep the same notation throughout. We establish a total of $N = T + 1$ categories: for each task, there is a corresponding category c_i, along with an additional category c_0 designated for task-agnostic classifications namely *General*. To enhance the robustness and generalization of the classification, all $N \times M$ exemplars in $\mathcal{S}$ are sourced from external datasets that are highly relevant for the respective tasks. Based on Equation (2), the classification result for neuron j in layer l is determined as $\text{Category}_l(j)$, where $l \in \{1, 2, ..., L\}$, and L is the number of total layers of the base model. Consequently, we group neurons for every layer:

$$g_l(t) = \{j : \text{Category}_l(j) = t\}, t = 0, \cdots, T.$$

For the fine-tuned models, we **retain the same group results** because we observed that the classification performance remained consistent before and after fine-tuning in §2.3.

Merging for MLP layers. We further merge the MLP layers of the base model and fine-tuned models based on the established neuron groupings. Specifically, in accordance with the task arithmetic framework, we utilize the task vectors to merge neuron groups. For the task-agnostic group $g_l(0)$, we adopt the approach proposed by Dai et al. (2025), calculating T weights α_t for each layer's $g_l(0)$ neuron group to merge the task vectors from the fine-tuned models. For more details of SubModeule Linear, please refer to §C.3. In contrast, for the other groups $g_l(i)$ where $i \in \{1, 2, ..., T\}$, all task vectors are discarded, which means all the coefficients are setting to zero. Formally, denote the operation of SubModule Linear as $\alpha = SubLin(f)$, which means calculating the merge coefficient α from the submodule f. Then for a neuron j, we can write

$$\theta_{merge,l}(j) = \theta_{0,l}(j) + \sum_{t=1}^{T} \alpha_{t,l}(j)\tau_{t,l}(j),$$

where $\theta_{0,l}(j)$ is the weight of neuron j in layer l of the base model, $\tau_{t,l}(j)$ is the task vector of neuron j in layer l of the t-th fine-tuned model and $\alpha_{t,l}(j)$ is the coefficient for merging with

$$\alpha_{t,l}(j) = \begin{cases} 0 & \text{if } j \notin g_l(0), \\ SubLin(g_l(0)) & \text{if } j \in g_l(0). \end{cases}$$

Merging for Attention and Embedding layers. For parameters unrelated to neuron groupings, such as those in the attention and embedding layers, various integration strategies are applicable. In our study, we evaluated both DARE (Yu et al., 2024) and the Submodule Linear method (Dai et al., 2025) (denote by NeuronMerge_1 and NauronMerge_2 respectively) to merge these parameters.

4 Experiments

4.1 Experiments Setup

Basic Settings. We adopt Qwen2.5-7B (Qwen Team, 2024) and Llama-3.1-8B (AI@Meta, 2024) as our backbone models. We follow the settings outlined in Dai et al. (2025) and fine-tune the models on three tasks: math, coding, and translation. We utilize the GSM8K dataset (Cobbe et al., 2021) for the math task, the Code Alpaca dataset (Chaudhary, 2023) for the coding task, and the zh↔ en dataset from (Xu et al., 2024a) for the translation task. During the training phase, we implement the FastChat template (Zheng et al., 2023) for prompt design, performing fine-tuning over 2 epochs with a batch size of 128 and a learning rate of 2×10^{-6}. For the evaluation phase, we employ the GSM8K test set for math, HumanEval (Chen et al., 2021) for coding, and the resources provided in Xu et al. (2024a) for translation. Besides merging neurons in MLP layers as proposed in this paper, we tested both the "Submodule Linearity" technique (Dai et al., 2025) and the "DARE" (Yu et al., 2024) for attention layers.

Algorithm Implementation Details. In practice, the classification settings for neurons are based on the methodology outlined in section 2. We have classified all layers of the base model, and the classification results for each layer can be found in §C.1 and we also show some cases of neurons in §D.10. In the setting of merging the two models, we only select the corresponding two categories, merging the category of the third task into the general category c_0. When applying DARE to the attention layers, we use the same hyperparameters as those specified for DARE in §C.3.

Methods	Math & Coding	Math & Translate	Coding & Translate	Math & Coding & Translate
Fine-tuned Model	71.48	81.43	77.02	76.64
Task Arithmetic	69.73	81.71	74.81	75.36
DARE	69.84	82.31	<u>75.11</u>	76.10
NeuronMerge$_1$	**71.12** ± 0.36	<u>82.62</u> ± 0.16	**75.44** ± 0.21	**76.82** ± 0.11
SubModule Linearity	69.18	82.19	74.77	75.42
NeuronMerge$_2$	<u>70.80</u> ± 0.33	**82.70** ± 0.20	74.90 ± 0.16	<u>76.21</u> ± 0.23

Table 1: Results of Qwen2.5-7B. Math&Coding represents the result of merging the Math SFT model and the Coding SFT model and other columns are like-wise. Here the values in each column are the means of the merged model on the corresponding tasks. For example, in the first column, the value is the average of the scores on GSM8K and HumanEval for the merged model, and so on. For each setting, we replicated for 5 times with different sample seeds and compute the mean value and standard deviation of five results. The best and second-best results are highlighted in **bold** and <u>underlined</u>, respectively.

Methods	Math & Coding	Math & Translate	Coding & Translate	Math & Coding & Translate
Fine-tuned Model	47.71	71.55	62.25	60.50
Task Arithmetic	<u>47.41</u>	<u>70.45</u>	61.93	59.04
DARE	46.81	70.27	61.96	58.26
NeuronMerge$_1$	47.24 ± 0.14	70.39 ± 0.21	<u>62.89</u> ± 0.26	**59.94** ± 0.20
Submodule Linearity	47.13	70.43	62.65	59.37
NeuronMerge$_2$	**47.73** ± 0.28	**70.69** ± 0.29	**63.00** ± 0.16	<u>59.88</u> ± 0.22

Table 2: Results of Llama-3.1-8B. The details of this table are the same as Table 1. For more detailed results, please refer to the Appendix C.

4.2 Main Results

We compare our method with several baseline methods. **Task Arithmetic** (Ilharco et al., 2023), involves a straightforward weighted combination of *task vectors* and weights to merge models. **DARE** (Yu et al., 2024), builds upon the Task Arithmetic framework by incorporating random dropout of parameters within the *task vectors*. This mechanism aims to mitigate conflicts between different task vectors during the merging process. **Submodule Linearity** (Dai et al., 2025), leverage the linear properties at the submodule level when integrating fine-tuned models with task arithmetic.

We present the results of fine-tuned models merged in different approaches, with Qwen2.5-7B and Llama-3.1-8B serving as the foundational models. The results are shown in Tables 1 and 2, respectively. Each entry reflects the average evaluation metrics obtained in related tasks. It is evident that our method outperforms the baselines in most settings, both in the two SOTA open-sourced foundation models in Qwen2.5-7B and Llama-3.1-8B, the proposed method achieves an improvement of around 1%. And also in the case of Math-Translate merge for Qwen2.5-7B, and Math-Coding merge, Coding-Translate merge for Llama-3.1-8B, the performances surpass the original fine-tuned models. For more results of Qwen2.5-14B and Llama-2-13B (Touvron et al., 2023), please refer to §C.2. From these results, we have three observations: 1) The proposed Neuron Merge method are superior on different types of foundation models. 2) Merging models based on corresponding functional groups of neurons are consistently effective for both closed-QA (math) and open-QA tasks (code/translate), especially in the case of merging Math and Code models. 3) The performance gains on both "NeuronMerge$_1$" vs. "DARE" and

Math neuron \ Code neuron	Choose Code Model	Merge	Drop
Choose Math Model	70.42	69.95	69.94
Merge	70.21	<u>69.89</u>	70.63
Drop	70.52	70.64	**70.80**

Table 3: Comparison of Math and Code Neurons across different merging options. For each type of neuron (Math and Code), three strategies are considered: (1) Choose Model — retain parameters from one specific model, (2) Merge — combine models using SubModule Linearity, and (3) Drop — retain Base model parameters. The numbers represent the average scores for GSM8K and HumanEval. The best performance was achieved when both task-specific neurons were set to "Drop", as indicated in **bold**. The <u>underlined</u> number is the performance of naively merging with finer granularity, which is insufficient to improve the task arithmetic further.

	Task Arithmetic	Random grouping	Our Method
Math	78.77	75.73	78.80
Coding	61.59	60.98	62.80
Avg	69.73	68.36	70.80

Table 4: Comparison of model performance between random neuron grouping and our proposed method across Math and Coding tasks. The performance metrics illustrate the advantages of targeted neuron classification over random assignment.

"NeuronMerge$_2$" vs. "SubModule Linear" indicating that our method is orthogonal and complementary to the merging methods for other components in LLMs (i.e. attention and embedding layers).

4.3 Ablation Study

To validate the effectiveness of our neuron merging method, we compare our approach with other options for the grouped neurons as well as random neuron grouping. Furthermore, we examine the stability of our neuron functionality classification method by testing the hyper-parameter *top-k*, which determines the number of top-activated sentences selected for neuron functionality classification.

Different neuron merging approaches In the experiment of merging math and code models, our approach drops the task vectors for Math neurons and Code neurons. Actually, there are several alternative options for each neuron functionality categories, including weighted merging with weights calculated by SubModule Linear and direct replacement using the neurons in corresponding fine-tuned models. We evaluated math and code metrics on Qwen2.5-7B while exploring these different options, as shown in Table 3. Our findings indicate that, although all options show competitive scores, the highest performance metrics are achieved when the "task vector" for both neuron classes completely dropped. This results align with the concept of "spurious forgetting" identified in Zheng et al. (2025), where performance drops in models are attributed to misalignment in task-specific adaptations rather than true knowledge loss. The observed improvement when discarding task-specific neurons suggests conflicts of "task vectors" (Yu et al., 2024), and resemblance to the "Freeze" strategy, e.g. freezing certain parameters during training could preserve their knowledge while mitigating alignment issues (Zheng et al., 2025). Please refer to §D.8 for more discussion.

Random neuron grouping. We randomly grouped the neurons of Qwen2.5-7B into three categories: *General*, *Math*, and *Code*, ensuring that each group size is the same as the result of neuron functionality classification. We discarded the parameter deltas from the Math and Code groups. The results are presented in Table 4. It can be seen that the performance of the merged model was inferior compared to our method. This indicates that merging neurons based on their functionalities leads to a more coherent and effective integration of their contributions within the model.

The number of Top-Activated sentences. Since we classify the neuron functionality with top k highest-activated sentences where $k = 10$. We ablate k on the Qwen2.5-7B model on math and coding tasks, with the results presented in the Table 9 in §D.1. In these experiments, we varied the top k parameter from 10 to 40 to evaluate its impact on the performance of merged models. The results indicate that, regardless of k, the performance of the merged model remains quite stable. And $k = 10$ is a reasonable choice while keeping a lower computation and storage overhead.

5 Related Works

Task Arithmetic and Linearity Recent advances in large language models have spurred interest in efficient model merging techniques. Weight interpolation methods (Frankle et al., 2020; Izmailov et al., 2018) improve generalization and multi-task performance by averaging parameters, while task arithmetic (Ilharco et al., 2023) integrates models through weighted parameter differences, inspiring variants like (Yang et al., 2024; Yu et al., 2024; Yadav et al., 2023). Though fine-tuned models deviate from NTK theory predictions (Jacot et al., 2018), preserved parameter linearity remains critical for effective task arithmetic (Ortiz-Jiménez et al., 2023). Recent work enhances linearity via constrained parameter updates or selective linearization (Jin et al., 2024), with submodule-level linearity enabling superior block-wise task arithmetic performance (Dai et al., 2025).

Neuron and Interpretability Growing interest in large model interpretability has evolved from analyzing attention mechanisms (Elhage et al., 2021) to probing MLP modules. Early work studied MLP activations through key-value memory frameworks (Geva et al., 2021), while later efforts addressed neuron superposition challenges (Black et al., 2022) via sparse autoencoders (Bricken et al., 2023; Gao et al., 2024). Despite high training costs for autoencoders, neuron-centric approaches (Choi et al., 2024) remain prevalent, with interpretability insights directly enhancing task performance (Nikankin et al., 2024).

6 Conclusion

In conclusion, this work advances the understanding of neuron functionalities in state-of-the-art LLMs. We demonstrate that neurons maintain consistent performance across various tasks, both before and after fine-tuning. Our novel model merging method, which integrates neuron analysis with task arithmetic, showcases significant effectiveness and superior performance across different foundational models. Notably, our approach is orthogonal and complementary to existing methods, highlighting its potential for broader applicability in future research.

Limitations

This work has several limitations. First, the classification of neurons may lack precision, as we relied on top-activated examples and categorized them by data type, potentially overlooking nuanced functionalities. Additionally, we did not analyze the role of attention mechanisms in conjunction with neuron functionalities, highlighting a critical area for further exploration in future research.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China under Grant U23A2064 and in part by the National Natural Science Foundation of China under Grant 12031005.

References

AI@Meta. 2024. The llama 3 herd of models. Preprint, arXiv:2407.21783.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2024. Llemma: An open language model for mathematics. Preprint, arXiv:2310.10631.

Steven Bills, Nick Cammarata, Dan Mossing, Henk Tillman, Leo Gao, Gabriel Goh, Ilya Sutskever, Jan Leike, Jeff Wu, and William Saunders. 2023. Language models can explain neurons in language models. https://openaipublic.blob.core.windows.net/neuron-explainer/paper/index.html.

Sid Black, Lee Sharkey, Leo Grinsztajn, Eric Winsor, Dan Braun, Jacob Merizian, Kip Parker, Carlos Ramón Guevara, Beren Millidge, Gabriel Alfour, and Connor Leahy. 2022. Interpreting neural networks through the polytope lens. Preprint, arXiv:2211.12312.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E Burke, Tristan Hume, Shan Carter, Tom Henighan, and Christopher Olah. 2023. Towards monosemanticity: Decomposing language models with dictionary learning. Transformer Circuits Thread. Https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter,

Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. CoRR, abs/2107.03374.

Dami Choi, Vincent Huang, Kevin Meng, Daniel D Johnson, Jacob Steinhardt, and Sarah Schwettmann. 2024. Scaling automatic neuron description. https://transluce.org/neuron-descriptions.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. CoRR, abs/2110.14168.

Rui Dai, Sile Hu, Xu Shren, Yonggang Zhang, XinMei Tian, and Jieping Ye. 2025. Leveraging submodule linearity enhances task arithmetic performance in llms. In The Thirteenth International Conference on Learning Representations.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. 2021. A mathematical framework for transformer circuits. Transformer Circuits Thread. Https://transformer-circuits.pub/2021/framework/index.html.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. 2020. Linear mode connectivity and the lottery ticket hypothesis. In Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, volume 119 of Proceedings of Machine Learning Research, pages 3259–3269. PMLR.

Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. 2024. Scaling and evaluating sparse autoencoders. Preprint, arXiv:2406.04093.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer feed-forward layers are key-value memories. Preprint, arXiv:2012.14913.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. 2024. Qwen2. 5-coder technical report. arXiv preprint arXiv:2409.12186.

Gabriel Ilharco, Marco Túlio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2023. Editing models with task arithmetic. In The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry P. Vetrov, and Andrew Gordon Wilson. 2018. Averaging weights leads to wider optima and better generalization. In Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018, pages 876–885. AUAI Press.

Arthur Jacot, Clément Hongler, and Franck Gabriel. 2018. Neural tangent kernel: Convergence and generalization in neural networks. In Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, pages 8580–8589.

Ruochen Jin, Bojian Hou, Jiancong Xiao, Weijie J. Su, and Li Shen. 2024. Fine-tuning linear layers only is a simple yet effective way for task arithmetic. CoRR, abs/2407.07089.

Xisen Jin, Xiang Ren, Daniel Preotiuc-Pietro, and Pengxiang Cheng. 2023. Dataless knowledge fusion by merging weights of language models. In The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. Starcoder: may the source be with you! Preprint, arXiv:2305.06161.

Tian Yu Liu, Aditya Golatkar, and Stefano Soatto. 2024. Tangent transformers for composition, privacy and removal. In The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net.

Michael Matena and Colin Raffel. 2022. Merging models with fisher-weighted averaging. In Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022.

Yaniv Nikankin, Anja Reusch, Aaron Mueller, and Yonatan Belinkov. 2024. Arithmetic without algorithms: Language models solve math with a bag of heuristics. Preprint, arXiv:2410.21272.

Guillermo Ortiz-Jiménez, Alessandro Favero, and Pascal Frossard. 2023. Task arithmetic in the tangent space: Improved editing of pre-trained models. In Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

George Stoica, Daniel Bolya, Jakob Bjorner, Pratik Ramesh, Taylor Hearn, and Judy Hoffman. 2023. Zipit! merging models from different tasks without training. arXiv preprint arXiv:2305.03053.

Anke Tang, Li Shen, Yong Luo, Yibing Zhan, Han Hu, Bo Du, Yixin Chen, and Dacheng Tao. 2024. Parameter-efficient multi-task model fusion with partial linearization. In The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. CoRR, abs/2307.09288.

Xiaozhi Wang, Kaiyue Wen, Zhengyan Zhang, Lei Hou, Zhiyuan Liu, and Juanzi Li. 2022. Finding skill neurons in pre-trained transformer-based language models. In Proceedings of EMNLP.

Maurice Weber, Daniel Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, Ben Athiwaratkun, Rahul Chalamala, Kezhen Chen, Max Ryabinin, Tri Dao, Percy Liang, Christopher Ré, Irina Rish, and Ce Zhang. 2024. Redpajama: an open dataset for training large language models. Preprint, arXiv:2411.12372.

Haoran Xu, Young Jin Kim, Amr Sharaf, and Hany Hassan Awadalla. 2024a. A paradigm shift in machine translation: Boosting translation performance of large language models. In The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net.

Zhengqi Xu, Ke Yuan, Huiqiong Wang, Yong Wang, Mingli Song, and Jie Song. 2024b. Training-free pretrained model merging. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 5915–5925.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A. Raffel, and Mohit Bansal. 2023. Ties-merging: Resolving interference when merging models. In Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023.

Enneng Yang, Zhenyi Wang, Li Shen, Shiwei Liu, Guibing Guo, Xingwei Wang, and Dacheng Tao. 2024. Adamerging: Adaptive model merging for multi-task learning. In The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. 2024. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024. OpenReview.net.

Junhao Zheng, Xidi Cai, Shengjie Qiu, and Qianli Ma. 2025. Spurious forgetting in continual learning of language models. In The Thirteenth International Conference on Learning Representations.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. Preprint, arXiv:2306.05685.

Zhanpeng Zhou, Zijun Chen, Yilan Chen, Bo Zhang, and Junchi Yan. 2024. On the emergence of cross-task linearity in pretraining-finetuning paradigm. In Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024. OpenReview.net.

A More Results of Neuron Analysis

A.1 Additional Neuron Functionality Classification Results

In Figure 6, results for every 5 layers and the last layer (layer 27) are shown. The distribution of neuron categories remains largely consistent across base and fine-tuned models, demonstrating high distributional uniformity. In addition, the proportion of neurons that overlap within each category (depicted by the purple bar) is substantially high for almost all layers. This findings highlight that most neurons retain their functionalities after fine-tuning for all layers with both closed-QA (math) and open-QA tasks (code/translate).

A.2 Additional Neuron Knock-out Results

In Figure 7, knocking-out *Code*, *Math*, *Translation* neuron categories respectively in Qwen2.5-7B (layer 13) will consistently lead to a greater decline in corresponding abilities compared with random knock-out. This result indicates that grouping neurons with respect to their functions plays a key role in model performance for corresponding tasks.

A.3 Neuron Activation Cosine-Similarity Analysis

To quantify the high consistency of neurons in base and different fine-tuned models in the same position, we analyzed the cosine similarity between the weight vectors of corresponding neurons. Specifically, fix layer L, for the i-*th* neuron in the base model and the i-*th* neuron in the fine-tuned model , the cosine similarity $cos\_sim(w_i^{base}, w_i^{ft})$ is calculated as:

$$cos\_sim(w_i^{base}, w_i^{ft}) = \frac{w_i^{base} \cdot w_i^{ft}}{\|w_i^{base}\|_2 \|w_i^{ft}\|_2},$$

where w_i^{base} and w_i^{ft} are the weight vectors of base neuron and fine-tuned neuron, respectively (each representing a row or column of the weight matrix in MLP).

To better illustrate the closeness of the fine-tuned model to the base model, we compared the cosine similarities between the base-ft models and the base-CPT(continue pre-trained) models across all three matrices in the MLP. We selected Qwen2.5-Coder-7B (Hui et al., 2024) as the CPT model corresponding to Qwen2.5-7B. Our analysis revealed that the corresponding neurons exhibit an exceptionally high cosine similarity of up to 0.999 across all layers between Qwen2.5-7B and our code fine-tuned models (Figure 8), indicating near-identical parameter patterns. whereas the similarity between the CPT model and the base model is only around 0.4. This shows that the neuron parameters change very little after the fine-tuning, and also explains why the neuron functionality remains largely unchanged.

Moreover, due to the extremely high cosine similarity, methods for adjusting the positions of neurons such as Zipit (Stoica et al., 2023) and MuDSC (Xu et al., 2024b) are unnecessary in our setting.

B Analysis of Non-linearity for ReLU MLPs

In this section, we discuss the definition of linearity and the sources of non-linearity of MLP layer with $ReLU$ activations. From this, we conclude that grouping based on maximum activation values can help enhance linearity.

Definition 1. *(Linearity). Let f be a submodule of LLM, θ_0 and θ be parameters of f before and after fine-tuning respectively. We call θ exhibits linearity, if the differences in model weights caused by fine-tuning are linearly related to the differences in output features caused by fine-tuning for any input $x \in X$, i.e.*

$$f(x; \theta_0 + \alpha\tau) \approx f(x; \theta_0) + \alpha \Delta f(x; \theta_0 + \tau) \tag{4}$$

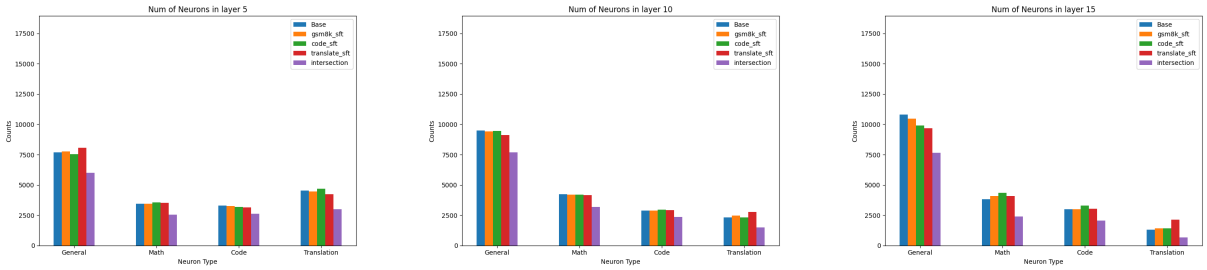
where $\tau = \theta - \theta_0$ is the differences in model weights before and after fine-tuning, and $\Delta f(x; \theta_0 + \tau) = f(x; \theta_0 + \tau) - f(x; \theta_0)$ is the differences in model output features before and after fine-tuning.

Remark 1. *If we define $g(\alpha) := f(x; \theta_0 + \alpha\tau) - f(x; \theta_0)$, the definition of linearity in equation (4) is equivalent to $g(\alpha)$ is a linear function for $\alpha \in [0, 1]$.*

We consider the MLP module with ReLU activation function for simplicity. The analyze for SwiGLU MLP is similar. Let d be the dimension of residual stream, n be the number of neurons in a MLP layer. A MLP layer of the base model can be expressed as

$$\begin{aligned} f(x; \theta_0) &= W_2 ReLU(W_1 x) \\ &= \sum_{i=1}^{n} ReLU(w_{1i} \cdot x) w_{2i}, \end{aligned} \tag{5}$$

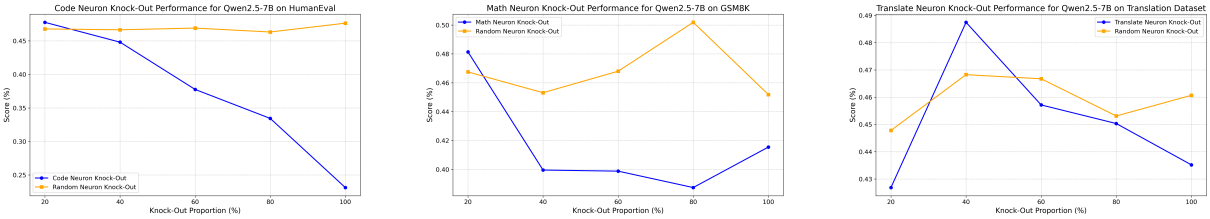
where $W_1 \in \mathbb{R}^{n \times d}, W_2 \in \mathbb{R}^{d \times n}$ are up and down projection matrix, respectively, and $w_{1i} \in$



(a) Layer 5 (b) Layer 10 (c) Layer 15

(d) Layer 20 (e) Layer 25 (f) Layer 27

Figure 6: Neuron functionality classification results for Qwen2.5-7B across different layers, shown as bar charts for different categories of neurons. *Base* model is Qwen2.5-7B. *gsm8k_sft*, *code_sft*, *translate_sft* are Qwen2.5-7B fine-tuned variants on Math, Code, Translation dataset respectively. Purple bars are the number of intersection neurons for each functionality category. Results for every 5 layers and the last layer (layer 27) are shown.



(a) Code neuron knock-out (b) Math neuron knock-out (c) Translation neuron knock-out

Figure 7: Knock-out neuron categories in Qwen2.5-7B (layer 13 shown) will lead to a greater decline in corresponding abilities compared with random knock-out. The horizontal coordinate indicates the knock-out proportion.

$\mathbb{R}^d, w_{2i} \in \mathbb{R}^d$(*i-th* row/column of W_1, W_2) are the weight of neuron i. The output of $f(x;\theta_0)$ can be viewed as a linear combination of $w_{2i}, i = 1, \cdots, n$ and the coefficient are the activation of the neuron $ReLU(w_{1i} \cdot x)$.

Under the aforementioned notation, we can derive the following theorem.

Theorem 1. *Consider a MLP layer with $ReLU$ activation function in (5), and further assume that*

$$sign(w_{1i} \cdot x + \tau_{1i} \cdot x) = sign(w_{1i} \cdot x), \forall x \in X, \tag{6}$$

where τ_{1i} is the task vector with respect to w_{1i}. Then we have

$$g(\alpha) = C_1\alpha + C_2\alpha^2, \tag{7}$$

where C_1, C_2 are constants only depending on x, W_1, W_2 and τ and independent of α.

Remark 2. *The assumption (6) is reasonable due to the extremely high cosine similarity between w_{1i} and $w_{1i} + \tau_{1i}$ observed in A.3.*

Remark 3. *By Remark 1, we expect $g(\alpha)$ to be a linear function of α. So we find two sources of non-linearity.*

(1) The first comes from (7), which is a quadratic term $C_2\alpha^2$. As shown in the proof below, C_2 is the product term of τ_1 and τ_2. Under the setting of task arithmetic, τ_1, τ_2 are small. Thus C_2 is negligible.

(2) The second comes from our assumption. Since the inputs x in the base model and the fine-tuned model are not perfectly identical, the

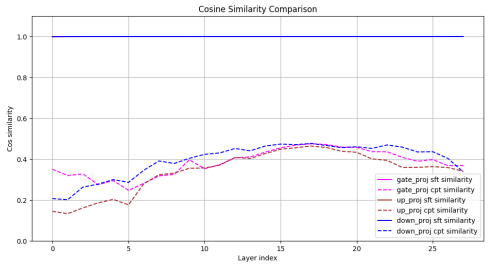


Figure 8: Mean Cosine similarity of neurons across different layers in the Qwen2.5-7B model. The solid lines represent the similarity between the Qwen2.5-7B model and our fine-tuned model, while the dashed lines indicate the cosine similarity between the Qwen2.5-7B and the Qwen2.5-Coder-7B models. Notably, the cosine similarity between our fine-tuned model and the base model reaches as high as 0.999, whereas the similarity between the CPT model and the base model is only around 0.4.

pre-activation after fine-tuning may not share the same sign as base model.

Remark 4. *To suppress sign-flip non-linearity, we prioritize neurons with persistently high activations on a specific dataset D_t, where $w_{1i} \cdot x \gg 0$ ensures*

$$w_{1i} \cdot x + \tau_{1i} \cdot x > 0, \forall x \in D_t,$$

through the norm dominance $\|w_{1i}\| \gg \|\tau_{1i}\|$. As a result, these neurons

$$g(t) = \{i : w_{1i} \cdot x \gg 0, \forall x \in D_t\}$$

form a group that exhibits high linearity on the datasets D_t.

Now we start to prove Theorem 1.

Proof of Theorem 1. Denote $x^+ := ReLU(x)$ for ease of notation, now consider function $g(\alpha) := f(x; \theta_0 + \alpha\tau) - f(x; \theta_0)$, we have

$$\begin{aligned}
g(\alpha) &= \sum_{i=1}^{n} (w_{1i} \cdot x + \alpha\tau_{1i} \cdot x)^+ (w_{2i} + \alpha\tau_{2i}) \\
&\quad - \sum_{i=1}^{n} (w_{1i} \cdot x)^+ w_{2i} \\
&= \sum_{i=1}^{n} [(w_{1i} \cdot x + \alpha\tau_{1i} \cdot x)^+ (w_{2i} + \alpha\tau_{2i}) \\
&\quad - (w_{1i} \cdot x + \alpha\tau_{1i} \cdot x)^+ w_{2i}] \\
&\quad + \sum_{i=1}^{n} [(w_{1i} \cdot x + \alpha\tau_{1i} \cdot x)^+ w_{2i} - (w_{1i} \cdot x)^+ w_{2i}] \\
&= \alpha \sum_{i=1}^{n} (w_{1i} \cdot x + \alpha\tau_{1i} \cdot x)^+ \tau_{2i} \\
&\quad + \sum_{i=1}^{n} \left((w_{1i} \cdot x + \alpha\tau_{1i} \cdot x)^+ - (w_{1i} \cdot x)^+ \right) w_{2i},
\end{aligned}$$

where τ_{1i}, τ_{2i} are the task vectors with respect to w_{1i}, w_{2i}.

Note that $\alpha \in [0, 1]$ and the assumption (6), we can infer that $w_{1i} \cdot x + \alpha\tau_{1i} \cdot x$ and $w_{1i} \cdot x$ have the same sign. Then the first term of $g(\alpha)$ can be written as

$$\begin{aligned}
I_1 &:= \alpha \sum_{i=1}^{n} (w_{1i} \cdot x + \alpha\tau_{1i} \cdot x)^+ \tau_{2i} \\
&= \alpha \sum_{i=1}^{n} (w_{1i} \cdot x + \alpha\tau_{1i} \cdot x) \mathbf{1}_{\{w_{1i} \cdot x + \alpha\tau_{1i} \cdot x > 0\}} \tau_{2i} \\
&= \alpha \sum_{i=1}^{n} (w_{1i} \cdot x + \alpha\tau_{1i} \cdot x) \mathbf{1}_{\{w_{1i} \cdot x > 0\}} \tau_{2i} \\
&= \alpha \sum_{i=1}^{n} (w_{1i} \cdot x)^+ \tau_{2i} + \alpha^2 \sum_{i=1}^{n} (\tau_{1i} \cdot x) \mathbf{1}_{\{w_{1i} \cdot x > 0\}} \tau_{2i} \\
&=: \alpha C_1(x; W_1, \tau_2) + \alpha^2 C_2(x; W_1, \tau_1, \tau_2),
\end{aligned}$$

where C_1, C_2 are constants independent of α and $\mathbf{1}_{\{A\}}$ is the indicator function, which equals 1 if event A happens and equals 0 otherwise.

For the second term, we have

$$\begin{aligned}
I_2 &:= \sum_{i=1}^{n} \left((w_{1i} \cdot x + \alpha\tau_{1i} \cdot x)^+ - (w_{1i} \cdot x)^+ \right) w_{2i} \\
&= \alpha \sum_{i=1}^{n} (\tau_{1i} \cdot x) \mathbf{1}_{\{w_{1i} \cdot x > 0\}} w_{2i} \\
&=: \alpha C_3(x; W_1, W_2, \tau_1),
\end{aligned}$$

where C_3 is a constant independent of α.

In conclusion, we have

$$g(\alpha) = I_1 + I_2 = (C_1 + C_3)\alpha + C_2\alpha^2.$$

□

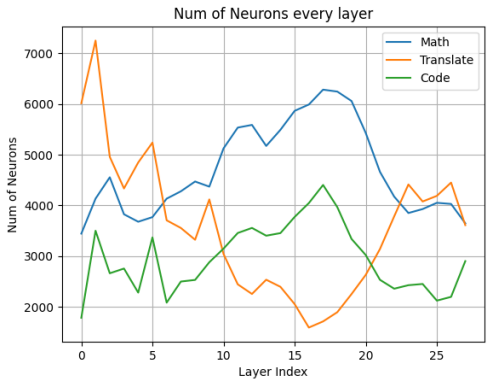
C More Results in Merging Models

C.1 The Number of Neurons of each Category

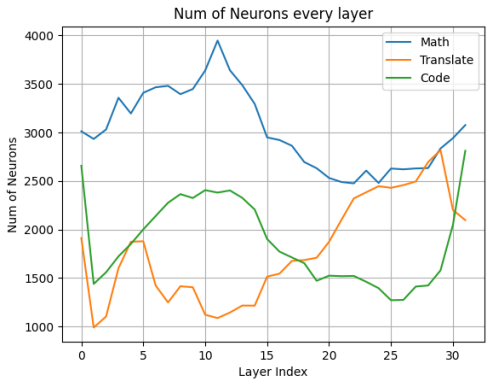
In this section, we represent the number of *Math*, *Code* and *Translate* neurons in each layer of Qwen2.5-7B and Llama3.1-8B in Figure 9a and Figure 9b, respectively. The pattern of Qwen2.5-7B differs significantly from that of Llama3.1-8B. Qwen2.5-7B contains more *Translate* neurons in the shallow layers, which may be attributed to its training on a larger amount of Chinese data.

C.2 Detailed Results in Experiments

In this section, we represent the detailed results of Tables 1 and 2 in Tables 5 and 6. We also present the detailed results on larger models Qwen2.5-14B and Llama2-13B for Math and Coding tasks in Table 7. For each setting, we replicated for 5 times



(a) Qwen2.5-7B.



(b) Llama3.1-8B.

Figure 9: The number of neurons in each layer.

with different sample seeds and compute the mean value of five results. The best and second-best results are highlighted in **bold** and underlined, respectively. For the hyperparameters of DARE and Task Arithmetic, we search for the best setting and the specific values are listed in Appendix C.3.

C.3 More Details About The Baseline

For Task arithmetic, We explored the hyperparameter merging weights $\alpha \in [0.1, 0.2, \ldots, 0.9]$ and selected the best results for reporting in the table.

For DARE, we fixed the merging weights $\alpha = 1$ and explored the dropout probability *drop_ratio* $\in [0.6, \ldots, 0.9]$, reporting the best results in the table.

For the Submodule Linearity, we utilize the Attn/MLP Level approach. This means that each Attention Layer and MLP Layer has its own coefficients when it comes to merging the models. Following the recommendations of Dai et al. (2025), we sample 30 data for each task.

The optimal hyperparameters corresponding to the best results in Table 1 and 2 obtained in practice are reported in Table 8.

D More Ablation and Discussion on Different Settings

D.1 Ablation on The number of Top-Activated sentences

We varied the top k parameter from 10 to 40 to evaluate its impact on the performance of merged models. The results in Table 3 indicate that, regardless of k, the performance of the merged model remains quite stable. And $k = 10$ is a reasonable choice while keeping a lower computation and storage overhead.

D.2 Ablation on more Combinations of Model Merge Methods

We attempt more combinations of model merging methods for both Attention parameters and general neurons.

In Table 10, we apply Task Arithmetic to Attention and SubModule Linearity to general neurons on Qwen2.5-7B, comparing with Task Arithmetic as baseline. Our approach, when combined with Task Arithmetic in the attention layers, consistently outperforms the baseline method.

In Table 11, we explore different methods for general neurons. All experiments involve merging the Math and Code models of Qwen2.5-7B. Our approaches consistently outperform the baseline methods, irrespective of the techniques employed. As illustrated in the table, the scores for experiments 1 to 4 range from 70.92 to 71.54, demonstrating consistent improvements over the three baseline methods (Task Arithmetic, DARE, and Sub-Lin), which score between 69.18 and 69.84. Additionally, our main methods (NeuronMerge$_1$ and NeuronMerge$_2$) show comparable performance, yielding scores of 71.12 and 70.80, respectively.

In Table 12, we explore various options similar to those presented in Table 3 for the Llama3.1-8B model. The results indicate a significant performance gap between the "drop" merging strategy and the other approaches.

D.3 Ablation on Neuron Grouping with Threshold

In this ablation experiment, we restrict that the task specific neurons have to be statistically significant based on top activated sentences. We define the

Method Qwen2.5-7B	Merging Two Models Math	Coding	Avg	Math	Translate	Avg
Fine-tuned Model	75.89	67.07	71.48	75.89	86.96	81.43
Task Arithmetic	78.77	61.59	69.73	76.65	86.77	81.71
DARE	77.48	62.20	69.84	77.79	**86.82**	82.31
NeuronMerge$_1$	**78.95**$_{\pm 0.33}$	**63.29**$_{\pm 0.51}$	**71.12**$_{\pm 0.36}$	78.73$_{\pm 0.29}$	86.51$_{\pm 0.05}$	82.62$_{\pm 0.16}$
Submodule Linearity	77.14	61.22	69.18	77.57	86.80	82.19
NeuronMerge$_2$	78.80$_{\pm 0.33}$	62.80$_{\pm 0.43}$	70.80$_{\pm 0.33}$	**78.89**$_{\pm 0.36}$	86.57$_{\pm 0.03}$	**82.70**$_{\pm 0.20}$

Method Qwen2.5-7B	Merging Two Models Coding	Translate	Avg	Merging Three Models Math	Coding	Translate	Avg
Fine-tuned Model	67.07	86.96	77.02	75.89	67.07	86.96	76.64
Task Arithmetic	62.80	86.62	74.81	78.84	60.97	86.28	75.36
DARE	63.41	86.81	75.11	78.17	63.41	86.72	76.10
NeuronMerge$_1$	**64.26**$_{\pm 0.41}$	86.61$_{\pm 0.02}$	**75.44**$_{\pm 0.21}$	**79.91**$_{\pm 0.42}$	**63.79**$_{\pm 0.02}$	86.76$_{\pm 0.51}$	**76.82**$_{\pm 0.11}$
Submodule Linearity	62.68	**86.86**	74.77	79.53	60.73	86.00	75.42
NeuronMerge$_2$	63.17$_{\pm 0.33}$	86.63$_{\pm 0.02}$	74.90$_{\pm 0.16}$	79.19$_{\pm 0.27}$	62.67$_{\pm 0.70}$	**86.77**$_{\pm 0.04}$	76.21$_{\pm 0.23}$

Table 5: Detailed experimental results on Qwen2.5-7B.

category of neurons as following instead of (2):

$$Category(j) = \begin{cases} \arg\max_{c_i \in \mathcal{C}} & \{s \in S_k(j) \mid s \in c_i\} \\ & \text{if } \max(S_k(j)) - \max_2(S_k(j)) \geq m \\ \text{General} & \text{otherwise} \end{cases}$$

where $\max_i$ means the i-th maximum element.

We test this framework using three different threshold values for $m : 1, 2, \cdots 8$. Table 13 summarizes the impact of these thresholds on neuron grouping and the corresponding performance metrics. Despite some fluctuation, the Math scores generally decrease while the Coding scores gradually increase as m increases from 1 to 8.

D.4 Automatically Neuron Classification Based on LLM

We also attempted to leverage LLM to automatically classify all the neurons, as demonstrated by Choi et al. (2024). For each neuron, we selected the top 10 activated sentences, as described in section 2. Additionally, we calculated the 0.9 percentile of the absolute activation values across all $1,200$ exemplars. We then utilized Qwen2.5-72B-Instruct to identify tokens from these top 10 samples that exceeded this percentile, which aided us in classifying the neurons. For each sentence, we asked the LLM to determine its category and provide a relevance score from 1 to 5. Finally, we aggregated the scores of the 10 sentences across the various categories and selected the category with the highest score as the classification for that neuron. For a detailed prompt, please refer to Appendix E.

Due to the substantial costs associated with analyzing neurons in each layer using this approach, we opted not to employ LLM-based classification in our main experiment.

In this section, we randomly sampled 100 neurons from Qwen2.5-7B and compared the results with our sample-type-based classification. As shown in Figure 10, 72 out of the 100 neurons were classified into the same category.

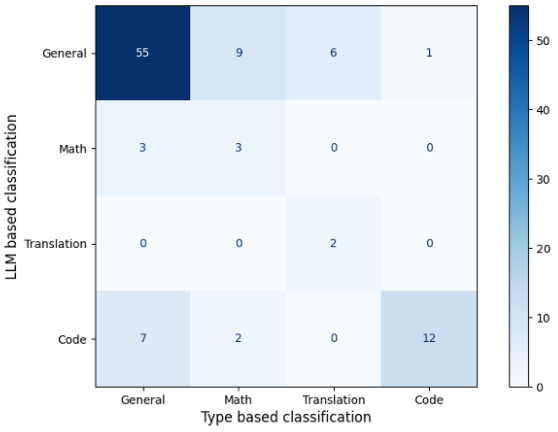


Figure 10: Classification results on 100 random sampled neurons in Qwen2.5-7B with LLM based method and sample type based method.

Method Llama3.1-8B	Merging Two Models Math	Coding	Avg	Math	Translate	Avg
Fine-tuned Model	57.01	38.41	47.71	57.01	86.09	71.55
Task Arithmetic	55.19	39.63	47.41	55.04	**85.85**	70.45
DARE	53.37	40.24	46.81	54.73	85.82	85.70
NeuronMerge$_1$	52.77$_{\pm 0.15}$	**41.70**$_{\pm 0.34}$	47.24$_{\pm 0.14}$	55.10 $_{\pm 0.44}$	85.68 $_{\pm 0.02}$	70.39 $_{\pm 0.21}$
Submodule Linearity	**55.37**	38.90	47.13	55.02	85.84	70.43
NeuronMerge$_2$	54.50$_{\pm 0.54}$	40.97$_{\pm 0.51}$	**47.73**$_{\pm 0.28}$	**55.68**$_{\pm 0.58}$	85.71$_{\pm 0.02}$	**70.69**$_{\pm 0.29}$

Method Llama3.1-8B	Merging Two Models Coding	Translate	Avg	Merging Three Models Math	Coding	Translate	Avg
Fine-tuned Model	38.41	86.09	62.25	57.01	38.41	86.09	60.50
Task Arithmetic	37.80	86.06	61.93	50.72	40.85	85.54	59.04
DARE	37.80	**86.12**	61.96	48.29	40.85	85.63	58.26
NeuronMerge$_1$	39.75$_{\pm 0.51}$	86.04$_{\pm 0.03}$	62.89$_{\pm 0.26}$	**50.99** $_{\pm 0.42}$	42.57$_{\pm 0.40}$	**86.27** $_{\pm 0.06}$	**59.94**$_{\pm 0.20}$
Submodule Linearity	39.26	86.04	62.65	50.52	42.68	84.90	59.37
NeuronMerge$_2$	**40.00**$_{\pm 0.33}$	86.01$_{\pm 0.03}$	**63.00**$_{\pm 0.16}$	50.93$_{\pm 0.51}$	**42.88**$_{\pm 0.51}$	85.84$_{\pm 0.03}$	59.88$_{\pm 0.22}$

Table 6: Detailed experimental results on Llama3.1-8B.

Method	Qwen2.5-14B Math	Coding	Avg	Llama2-13B Math	Coding	Avg
Fine-tuned Model	77.71	67.07	72.39	47.91	20.12	34.02
Task Arithmetic	83.55	65.24	74.40	45.41	23.78	34.60
DARE	84.15	64.63	74.39	41.55	24.39	32.97
NeuronMerge$_1$	**84.99**	65.65	75.32	43.05	**26.95**	35.00
Submodule Linearity	83.52	65.65	74.59	**46.85**	23.37	35.11
NeuronMerge$_2$	83.67	**67.07**	**75.37**	45.17	25.12	**35.15**

Table 7: Detailed experimental results on Qwen2.5-14B and Llama2-13B for Math and Coding tasks.

D.5 Discussion on Potential Data Leakage

We provide a detailed list of datasets used in our experiments in Table 14. To prevent potential data leakage, we replace the translation dataset with WMT24 and calculate the intersection-over-union ratio between the set of translation neurons S_{WMT24} and the previously identified neruon set $S_{WMT17-20}$, which is found to be 0.71. This demonstrates that our neuron classification does not rely on the fine-tuning dataset.

D.6 Comparison with more Baselines

In the context of Qwen2.5-7B for mathematics and code tasks, we have added performance metrics for TIES-Merging and PCB-Merging to our comparisons in Table 15, and our methods outperform these baselines.

D.7 Discussion on Cost-Performance Tradeoff

We provide a quantification of computational and storage costs. Taking our experiments on Qwen2.5-7B as an example, we detail the computational and storage costs for neuron classification as follows: 1) **Computational Cost:** For classifying neurons, it takes approximately 65s (1min) inference time on a single A100 GPU to get complete 28-layer model activations on 1,200 samples. 2) **Storage Cost:** We need to compute activation values for 4 categories $\times$ 300 sentences $\times$ 512 tokens. However, for each neuron, we only store the maximum activations of the top 10 sentences, leading to a total storage requirement of: $10 \times$ number of neurons $\times$ number of layers $= 10 \times 18944 \times 28 = 5.3M$ data points. At BF16 precision, this amounts to approximately 10.6 MB of storage space.

To further analyze the performance gain versus

Hyper-parameters	Math & Code	Math & Translate	Coding & Translate	Math & Coding & Translate
		Qwen2.5-7B		
Task Arithmetic	0.4	0.6	0.6	0.5
DARE	0.6	0.7	0.6	0.8
		Llama3.1-8B		
Task Arithmetic	0.5	0.6	0.6	0.5
DARE	0.6	0.6	0.6	0.7

Table 8: The optimal hyperparameters corresponding to the best results obtained in practice for baslines.

Top k	10	20	30	40
Avg	70.80	70.79	70.87	70.70

Table 9: Ablation result of Top-k sentences used to classify the neruon functionalities. The numbers in the table are the average scores on GSM8K and HumanEval.

time cost of each method compared to the baseline (i.e. Task Arithmetic), we calculate the 'performance gain' / 'normalized time cost' of each method and list the results in the following tables. Specifically, 'performance gain' is computed by subtraction of performance on each method and that of Task Arithmetic, and 'normalized time cost' is computed by division of the time cost of each method and that of Task Arithmetic. All experiments are conducted on Qwen2.5-7B with a single A100 GPU. Since DARE, Task Arithmetic, and TIES-Merging require a grid search of hyperparameters, we multiplied the time cost for a single run by the number of hyperparameter searched. The results of Math & Translate and Math & Code are listed in the Tables 16 and 17.

From these two tables, we have three observations: 1) From the view of performance, the proposed method achieves more consistent gains than the most efficient method SubModule Linearity (NeuronMerge$_2$ vs. SubModule Linearity). 2) From the view of time cost, the newly added 'neuron classify' step in the proposed method introduces an affordable fraction of additional time cost (e.g. 65s vs. 1020s/375s). 3) From the view of performance gain vs. time cost, the proposed method achieves a good balance between performance and cost compared with other methods (e.g. 1.187/2.626 vs. 0.116/-1.584).

D.8 Discussion on the Merging Strategy

Our initial approach aimed to optimize merging by adopting parameters from task-specific SFT models for neurons with specific functions—such as employing Math SFT model parameters for math neurons. However, experimental results indicated that this strategy, while promising, did not yield the anticipated optimal performance. We hypothesize this arises from the phenomenon of "spurious forgetting" (Zheng et al., 2025) and the effects of orthogonal updates in the model parameters. This suggests that while functional neurons are important, their coexistence can lead to performance conflicts across tasks due to misalignment, thus necessitating their removal to maintain overall model coherence. Our findings demonstrate that general neurons, which represent shared patterns across tasks, can effectively harmonize the alignment of different SFT models. Additionally, adopting a freeze strategy by merging general neurons and discarding updates for neurons with predominant orthogonal interactions has proven to enhance performance. This analysis highlights the intricate dynamics of neuron functionality in SFT model merging and underscores the need for a nuanced approach to balancing the contributions of both specific and general neurons in optimizing model performance.

D.9 Discussion on Dynamic Tasks

In our paper, we assumed that all tasks are known and fixed in advance, following the assumption commonly applied in related research areas like Task Arithmetic, etc. If the introduction of dynamic tasks is necessary, we only need to compute the activations for additional 300 data corresponding to each new task. These values can then be combined with the previously cached activations

Method	Merging Two Models								
Qwen2.5-7B	Math	Coding	Avg	Math	Translate	Avg	Coding	Translate	Avg
Task Arithmetic	78.77	61.59	69.73	76.65	**86.77**	81.71	62.80	86.62	74.81
Task Arithmetic + NeuronMerge	**79.06**	**62.20**	**70.62**	**79.20**	86.35	**82.78**	**63.78**	**86.53**	**75.15**

Table 10: Results of applying Task Arithmetic and the combined approach of Task Arithmetic with NeuronMerge on the Qwen2.5-7B model, showcasing performance across various tasks including Math, Coding, and Translation. Higher scores are highlighted in **bold**. Our approach, when combined with Task Arithmetic in the attention layers, consistently outperforms the baseline method.

Method	Attention Operation	General Neurons Operation	Average Performance
Task Arithmetic	-	-	69.73
DARE	-	-	69.84
SubLin	-	-	69.18
NeuronMerge$_1$	DARE	SubLin	71.12
NeuronMerge$_2$	SubLin	SubLin	70.80
experiment 1	DARE	Task Arithmetic	71.05
experiment 2	SubLin	Task Arithmetic	71.14
experiment 3	DARE	DARE	71.54
experiment 4	SubLin	DARE	70.92

Table 11: To further analyze the effectiveness of the proposed method, we conducted 4 experiments on Math&Code model merging task with different merge operations on general neuron group on Qwen2.5-7B. For example, experiment 1 adopted DARE on attention layers, and Task Arithmetic on general neurons in MLP layers. From the table, we can observe that experiments 1-4 scores range from 70.92-71.54, with consistent improvements compare to 3 baselines (Task Arithmetic, DARE, SubLin) which range from 69.18-69.84, and comparable to our main methods (NeuronMerge1 and NeuronMerge2, with scores 71.12 and 70.80 respectively).

to determine the Top-K sentences, which will facilitate the reclassification of neurons. And the refreshed classification results can be used further to merge the fine-tuned models. And also, if real data is unavailable in dynamic task scenarios, we suggest using LLM to synthesize task-relevant data with acceptable cost. This helps to further improve the generalizability of our method.

D.10 Cases for Different Categories of Neurons

In this section, we show some cases of activation pattern for different categories of neurons in Qwen2.5-7B. For a given neuron, we calculated the 0.9 percentile of the absolute activation values across all $1,200$ exemplars and displayed the tokens from the top 10 highly activated samples that exceed this percentile. The tokens in the same quotes represent consecutive activation values greater than the 0.9 percentile. We found that neurons in shallow layers tend to activate on a single, discrete token while neurons in deep layers tend to activate on a consecutive context. We show some cases of neurons in Figure 11.

E All the prompts used in our method

In this section we provide the prompt we used to classify the neurons based on LLM in Figure 12.

Layer 0, Neuron #1381: A neuron activates on some symbols in LaTeX.

{\partial u^j}-\frac{\partial g_{ij}}{\partial u^l}\right).$$ Check that $\Gamma^k_{ij}=\mathbf{g}^k\cdot\frac{\partial\mathbf{g}_i}{\partial u^j}$ (I really don't want to derive this identity right now)

Layer 27, Neuron #270: A neuron activates on mathematical formulas in LaTeX.

k}\frac{\partial x^l}{\partial u_j}\delta_{kl}=\sum_k\frac{\partial u^i}{\partial x^k}\frac{\partial x^k}{\partial u_j}=\delta^i_j.$$<newline><newline> We further introduce a matrix g_{ij} given as $g_{ij}=\mathbf{g}_i\cdot\mathbf{g}_j$. Because of orthogonality, we have $g_{ij}=h_i^2\delta _{ij}$. We also introduce $g^{ij}=\mathbf{g}^i\cdot\mathbf{g}^j$, feel free to check that $g^{ij}=\frac{1}{h_i^2}\delta_{ij}$, thus g_{ij} and $ g^{ij}$ are inverse matrices.

(a) Math Neuron.

Layer 0, Neuron #0: A neuron activates on "import" and "datetime".

```
os<newline>
import sys<newline>
import json<newline>
import random<newline>
import logging<newline>
import urllib.request<newline>
from datetime import datetime, timedelta<newline><newline># import 3rd party modules<newline>
sys.path.append('./lib')<newline><newline>import slackweb<newline>
import feedparser<newline>
from bs4 import BeautifulSoup
```

Layer 27, Neuron #64: The neuron is activated on a Django ListView.

```
class PostListView(ListView):<newline>    model = Post<newline>
    template_name = 'blog/home.html'<newline>    context_object_name = 'posts'<newline>    ordering = ['-date_posted']<newline>    paginate_by = 5
<newline><newline>    def get_context_data(self, **kwargs):<newline>        context = super().get_context_data(**kwargs)<newline>        <newline>
        if Post.objects.all().count():<newline>            context['latest'] = Post.objects.order_by('-date_posted')[0]<newline><newline>        if self.request
.user.is_authenticated:<newline>            context['user'] = User.objects.filter(username=self.request.user.username)[0]<newline>        <newline>
return context
```

(b) Code Neuron.

Layer 0, Neuron #348: A neuron assists in Translation and activates on technology companies.

A chat between a curious user and an artificial intelligence assistant. The assistant gives helpful, detailed, and polite answers to the user's questions. USER: Translate this from English to Chinese:<newline>English: Tim Berners-Lee, the inventor of the World Wide Web, is launching a startup that seeks to rival Facebook, Amazon and Google.<newline>Chinese: ASSISTANT: 万维网发明者蒂姆·伯纳斯-李（World Wide Web），日前正在启动一家试图与脸书、亚马逊和谷歌匹敌的创业公司。

Layer 27, Neuron #77: A neuron activates in the translation of names and place names.

The assistant gives helpful, detailed, and polite answers to the user's questions. USER: Translate this from English to Chinese:<newline>English: The museum will come up at Vidyasagar's residence at Baduria Bagan area where pictures and models, chronicling his life from his birth place at Birsinghapur village in Paschim Medinipur district to the house in north Kolkata, will be put on display, he said on Thursday.<newline>Chinese: ASSISTANT: 他星期四说，博物馆将建立在维迪亚萨加位于巴杜尔巴甘地区的住所内，将展出照片和模型，这些照片和模型记录了维迪亚萨加的一生，覆盖从出生在加斯奇姆麦地尼普尔区比尔辛格普尔村到搬迁至加尔各答北部居所的各个时期。<|im_end|>

(c) Translation Neuron.

Figure 11: More activation cases for different categories of neurons.

Math neuron \ Code neuron	Choose Code Model	Merge	Drop
Choose Math Model	44.71	45.17	46.66
Merge	44.01	44.40	45.64
Drop	44.00	45.45	**47.73**

Table 12: Comparison of Math and Code Neurons across different merging options. For each type of neuron (Math and Code), three strategies are considered: (1) Choose Model — retain parameters from one specific model, (2) Merge — combine models using SubModule Linearity, and (3) Drop — retain Base model parameters. The numbers represent the average scores for GSM8K and HumanEval. The best performance was achieved when both task-specific neurons were set to "Drop", as indicated in **bold**.

m	1	2	3	4	5	6	7	8
Math	78.80	77.85	77.15	77.56	77.05	76.67	76.56	76.66
Coding	62.80	62.32	62.68	63.29	63.41	63.29	64.26	63.65
Avg	70.80	70.08	69.92	70.42	70.23	69.98	70.41	70.16

Table 13: Impact of different thresholds m on model performance across Math and Coding tasks. The numbers in the table are the average scores of GSM8K and HumanEval.

	General	**Math**	**Code**	**Translation**
Neuron Classification	RedPajama CommonCrawl (train)	Proof-file-2 (train)	Python in StarCoder (train)	WMT17-WMT20
Fine-tuning	–	GSM8k (train)	CodeAlpaca (train)	WMT17-WMT20
Test	–	GSM8k (test)	HumanEval (test)	WMT22

Table 14: Dataset Splits used in our main experiments.

Method	**Math**	**Coding**	**Average**
Fine-tuned Model	75.89	67.07	71.48
Task Arithmetic	78.77	61.59	69.73
TIES-Merging	77.18	62.80	69.99
PCB-Merging	76.88	**63.41**	70.15
DARE	77.48	62.20	69.84
NeuronMerge$_1$	**78.95**	<u>63.29</u>	**71.12**
Submodule Linearity	77.14	61.22	69.18
NeuronMerge$_2$	<u>78.80</u>	62.80	<u>70.80</u>

Table 15: Qwen2.5-7B Performance Metrics. In above tables, Math means merged model performance on GSM8k and Coding means merged model performance on HumanEval. The best and second-best results are highlighted in **bold** and <u>underlined</u>, respectively.

Method	**Time Cost**	**Average Performance (Δ)**	**Δ / Normalized Time Cost**
Task Arithmetic	0s + 120s * 9 = 1080s	81.71%	-
TIES-Merging	0s + 1020s * 4 = 4080s	82.29% (+0.58%)	0.154
DARE	0s + 255s * 4 = 1020s	82.31% (+0.60%)	0.635
NeuronMerge$_1$	65s + 300s * 4 = 1265s	82.62% (+0.91%)	0.777
SubModule Linearity	0s + 375s = 375s	82.19% (+0.48%)	1.382
NeuronMerge$_2$	65s + 375s = 440s	82.7% (+0.99%)	2.43

Table 16: Performance-Cost tradeoff on Math & Translate tasks for merged models on Qwen2.5-7B. The Time Cost consists of two components: neuron classification and merging, which includes the time taken for a single merge multiplied by the time required for grid search hyperparameters.

Method	Time Cost	Average Performance (Δ)	Δ / Normalized Time Cost
Task Arithmetic	0s + 120s * 9 = 1080s	69.73%	-
TIES-Merging	0s + 1020s * 4 = 4080s	69.99% (+0.26%)	0.069
DARE	0s + 255s * 4 = 1020s	69.84% (+0.11%)	0.116
NeuronMerge$_1$	65s + 300s * 4 = 1265s	71.12% (+1.39%)	1.187
SubModule Linearity	0s + 375s = 375s	69.18% (-0.55%)	-1.584
NeuronMerge$_2$	65s + 375s = 440s	70.80% (+1.07%)	2.626

Table 17: Performance-Cost tradeoff on Math & Coding tasks for merged models on Qwen2.5-7B.

Algorithm 1 Merging Modules with Neuron Merge

Input:

T fine-tuned models with L MLP layers $\theta_i = \{\theta_{i,1}, \theta_{i,2}, \cdots, \theta_{i,L}\}$, $i = 1, 2, \cdots, T$;

Pre-trained model MLP layers $\theta_0 = \{\theta_{0,1}, \theta_{0,2}, \cdots, \theta_{0,L}\}$;

Datasets for neuron classification $d_0, d_1, \cdots, d_T$.

Output:

Merged model MLP layers $\theta_{\text{merge}} = \{\theta_{\text{merge},1}, \theta_{\text{merge},2}, \cdots, \theta_{\text{merge},L}\}$.

for $l = 1$ to L **do**
  $n =$ number of neurons
  /* Step 1: Classify neurons in each layer */
  $categories_l \leftarrow$ ClassifyNeurons$(\theta_{0,l}, d_0, d_1, \ldots, d_T)$ {Classification done by Equation (2)}
  /* Step 2: Group neurons by category */
  for $c = 0$ to T **do**
    $Neurons_c \leftarrow \{j \in [n] : categories_l(j) = c\}$
    $g_l(c) \leftarrow (\{\theta_{0,l}(j) : j \in Neurons_c\}, \{\theta_{1,l}(j) : j \in Neurons_c\}, \ldots, \{\theta_{T,l}(j) : j \in Neurons_c\})$
  end for
  /* Step 3: Merge neurons based on groups */
  for $j = 1$ to n **do**
    if $j \in Neurons_0$ **then**
      $\alpha_{t,l}(j) \leftarrow$ SubLin$(g_l(0))$
    else
      $\alpha_{t,l}(j) \leftarrow 0$
    end if
    /* Update the weights of the merged model */
    $\theta_{\text{merge},l}(j) \leftarrow \theta_{0,l}(j) + \sum_{t=1}^{T} \alpha_{t,l}(j) \cdot [\theta_{t,l}(j) - \theta_{0,l}(j)]$
  end for
end for
return θ_{merge}

Prompt for LLM based Neuron Classification

```
Evaluation Instructions:
# Role
You are an AI researcher specializing in analyzing specific neurons within neural networks and their responses to certain text fragments.
      Your primary task is to identify which domain the neuron is most related to based on its activation patterns.

# Skills
1. **Neuron Analysis**: Meticulously examine each neuron's information, to determine its strongest association with a particular domain.
2. **Contextual Assessment**: Evaluate the neuron's response based on activation patterns, such as trigger keywords (e.g., {{
      highly_activated_token}}).
3. **Domain Identification**: Select the most relevant domain from the following options:
   - A. Mathematics: Concepts, terminology, and operations related to mathematics.
   - B. Programming: Code snippets, programming languages, and computational algorithms.
   - C. Translation: Activate on cross-linguistic tokens with the same semantics and on translation instructions.
   - D. Others: Domains not covered by the above categories, such as general knowledge, pronouns and other common words, etc.

# Constraints
1. **Sequential Dependency**: Neuron activations are influenced solely by the sequence of words preceding the activation point. Your judgment
       must only consider words before the activation and disregard any subsequent words.
2. **Token-Based Analysis**: The analysis must focus on the activation patterns of individual tokens and their impact on the neuron's
      functionality, rather than analyzing the overall semantics of entire sentences.
3. **Activation Pattern Interpretation**: Infer the neuron's characteristics based on activation patterns to determine its associated domain.
4. **Scoring Mechanism**: For each neuron, provide:
   - **Selected Domain**: [Choose from A-D]
   - **Relevance Score**: [1-5] based on the following guidelines:
     - **1**: Minimal relevance
     - **2**: Low relevance
     - **3**: Moderate relevance
     - **4**: High relevance
     - **5**: Very high relevance
   - **Explanation**: Brief reasoning supporting the score.
5. **Reply Requirements**: Only output 'Selected Domain', 'Relevance Score', and 'Explanation'. Do **NOT** reply with any other information
      or copy any examples text in Prompt.

Use these insights to discern pattern strongly associated with particular domains.
## Neuron Analysis
**Excerpt and Activation Example**:
- Excerpt 1: " boto", ".Session", "=tf", "_letters", "igits", "added", ".valid", "added", "added", "import", "from time import time", "import
      ", ".pyplot", " =", "(sys", " =", "for i", " range", "Num", "Num", " i", " range", "Num", "'", " i", " range", ".pyplot", ".pyplot", ".
      pyplot"
**Conclusion**:
```

Figure 12: This is the prompt we used in the LLM based classification.