# EFFICIENT MODEL EDITING WITH TASK-LOCALIZED SPARSE FINE-TUNING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Pre-trained models are stepping stones for modern machine learning systems, but how to efficiently extract, reuse, and steer their knowledge for new tasks is an area of research with still several open questions. State-of-the-art Task Arithmetic solutions are strongly tied to model linearization which leads to computational bottlenecks during training and inference, and potentially neglect essential task dependencies. In this work, we focus on the fine-tuning stage that defines task vectors and propose `TaLoS`, a new approach based on sparse fine-tuning that strategically updates only parameters expected to provide functional task localization. This efficiently yields weight-disentangled models without the need for explicit linearization. We present a thorough experimental analysis showing how our approach significantly improves in training and inference efficiency while outperforming state-of-the-art approaches in task addition and task negation. Our work offers a principled solution to pre-trained model editing and paves the way to more cost-effective and scalable machine learning systems for real-world applications.

## 1 INTRODUCTION

Pretrained models (Radford et al., 2021; Raffel et al., 2020; Brown et al., 2020) have become the cornerstone of modern machine learning, demonstrating impressive capabilities in solving general tasks and offering a wealth of reusable knowledge for downstream applications (Kirillov et al., 2023; Touvron et al., 2023). However, while these models excel in general domains, they often require fine-tuning to achieve optimal performance on specialized tasks or to align with user preferences.

While the high computational cost limits the development of large pre-training models only to a few privileged research groups, fine-tuning is becoming increasingly democratized, thanks to efficient techniques enabling model customization on more affordable consumer GPUs. Consequently, a new paradigm has emerged where large companies and institutions publicly release pretrained models, empowering users to leverage, parameter-efficient fine-tuning (PEFT) (Hu et al., 2022; Liu et al., 2022; 2024), sparsity (Ansell et al., 2022; 2024), and quantization (Dettmers et al., 2024) to efficiently adapt these models to their own data. This recent trend has fueled the growth of a rich ecosystem of task-specific models, readily available on open platforms (Pfeiffer et al., 2020; Poth et al., 2023), fostering collaborative knowledge building by enabling users to share, reuse, and combine specialized modules for personal use (Raffel, 2023).

In this context, *Task Arithmetic* (Ilharco et al., 2023) has emerged as a promising framework for scalable and cost-effective model editing. By directly manipulating the model's parameters, or *task vectors*, task arithmetic can induce functional changes such as combining functionalities, enhancing performance on specific tasks, or even suppressing undesired behaviors. However, performing task arithmetic in a decentralized and collaborative setting, where independently fine-tuned modules are combined, presents significant challenges. One major challenge is the potential for unintended interference between tasks (Yadav et al., 2023; Wang et al., 2024), where the addition or deletion of a functionality disrupts performance on previously learned tasks. This interference can arise when the fine-tuning process modifies parameters that are crucial for other tasks, leading to unexpected and undesirable changes in the model's behavior.

To address this, Ortiz-Jimenez et al. (2023) formalized the notion of task arithmetic and demonstrated that *weight disentanglement*, where the model behaves as a composition of independently activated *localized components*, is crucial for preventing interference. They further argued that explicit linearization of the model during fine-tuning can preserve and enhance weight disentanglement, albeit at the cost of increased computational overhead.

In this work, we propose a novel approach that avoids the computational burden of explicit linearization while still achieving robust task arithmetic. Our key insight is that **sparse fine-tuning**, that selectively updates a small subset of the model's parameters, can inherently induce linearized behavior and promote weight disentanglement. We derive conditions for identifying the parameters that can be safely updated while maintaining the model's insensitivity to their changes on other tasks. Through extensive empirical analyses and theoretical justification, we demonstrate that our approach *effectively promotes function localization*, preventing interference between tasks and ensuring compatibility between task vectors. This enables efficient and robust model editing through the simple addition and subtraction of sparse task vectors, facilitating a more modular and collaborative approach.

**We can summarize our main contributions as follows.**

- We advance the field of task arithmetic by deriving a novel set of function localization constraints that provide exact guarantees of weight disentanglement on linearized networks.
- We empirically observed that least sensitive parameters in transformer-based architectures pre-trained on large-scale datasets can be consistently identified regardless of the task. We exploit this regularity to satisfy the localization constraints under strict individual training assumptions.
- We introduce *Task-Localized Sparse Fine-Tuning* (TaLoS) that enables task arithmetic by jointly implementing the localization constraints and inducing a linear regime during fine-tuning, without incurring in the overheads of explicit network linearization.

Overall, our work addresses a critical gap in task arithmetic, providing a more complete and practical framework for parameter-space model editing, targeting real-world applications.

## 2 Related Works

**Sparsity & Parameter-Efficient Fine-Tuning.** Model pruning and quantization are compression strategies generally applied after training for efficient storage and inference of large models (Liang et al., 2021). Research on pruning at initialization has also highlighted that model sparsification can be performed before training by searching for subnetworks (lottery tickets, Frankle & Carbin (2019)) that once trained can match the test accuracy of the original dense networks with a largely reduced learning cost. More recently the attention has moved towards parameter-efficient fine-tuning (PEFT) methods that reduce the number of trainable parameters that should be updated for adaptation. Adapter layers (Houlsby et al., 2019) and prefix tuning (Li & Liang, 2021) are among the most used techniques together with low-rank adaptation (LoRA, Hu et al. (2022)). The latter optimizes rank decomposition matrices of the dense layers' change during adaptation: it approximates model updates while keeping the pre-trained weights frozen, thus fine-tuning the dense layers in a neural network indirectly. Sparse masking approaches (Wortsman et al., 2020; Mallya et al., 2018; Mallya & Lazebnik, 2018; Havasi et al., 2020) employ subnetworks for continual and multi-task learning by leveraging sparsity. Other works, (Guo et al., 2021; Xu et al., 2021), explore sparse fine-tuning strategies to improve training efficiency, often using the Fisher information matrix (Fisher, 1922; Amari, 1996) for selecting important weights, as further explored by Sung et al. (2021); Ben Zaken et al. (2022). In contrast, Liao et al. (2023); Ansell et al. (2024) focus on fine-tuning only the least important parameters, aiming for minimal disruption of the original model. In the realm of sparse weight addition, Ansell et al. (2022); Panda et al. (2024) investigate adding sparse weights, providing a flexible and complementary approach to model composition.

**Model Merging.** The goal of model merging is to combine multiple task-specific models into a single multitask model without performing additional training. This can be obtained by merging techniques that avoid negative interferences among the separately learned parameters. Even when dealing with fine-tuned models initialized from the same pre-trained model, simple parameter averaging is not enough: existing approaches search for tailored re-weighting schemes that however tend to be computationally demanding. RegMean (Jin et al., 2023) solves a local linear regression problem for each individual linear layer in the model that requires transmitting extra data statistics of the same size as the model and additional inference steps. Fisher Merging (Matena & Raffel, 2022) exploits the Fisher information matrix that requires computing gradients with high memory costs. A recent approach exploits extra unlabeled data to learn the model merging weights (Yang et al., 2024).

**Task Arithmetic.** Task arithmetic (Ilharco et al., 2023) has been recently introduced as a new paradigm for editing models based on arithmetic operations over task-specific vectors obtained by fine-tuning a fixed pre-trained model and then subtracting the pre-trained weights from the fine-tuned ones. Such a definition of task vectors has been used also in the model merging literature with

approaches that rely on various post-hoc heuristics to resolve overlap among redundant parameter values and sign disagreements when merging (TIES, Yadav et al. (2023)), or after merging by deactivating irrelevant model parts with binary masks (TALL-mask, Wang et al. (2024)). Other approaches proposed to sparsify task vectors with either a random drop and re-scaling (Yu et al., 2024) or with masks that eliminate weight outliers (Davari & Belilovsky, 2024). However, task arithmetic goes beyond model merging as it aims at *adding to* or *deleting* knowledge and capabilities from a model in a modular and efficient manner. The goal is retaining and possibly improving separate task performance while also performing unlearning when needed. Theoretical foundations about the inner functioning of task arithmetic were offered by Ortiz-Jimenez et al. (2023), revealing how the weight disentanglement that arises during pre-training is the crucial factor that makes it effective. The authors of that work focused on the fine-tuning stage. Specifically, they proposed to preserve the disentanglement property by linearizing the models and fine-tuning in the tangent space. However, this solution is computationally expensive and may cause single-task performance drops (Ortiz-Jiménez et al., 2021). To reduce the costs of linearization, Tang et al. (2024) proposed to use linearized low-rank adapters during fine-tuning.

Our work fits in the context of task arithmetic and targets the fine-tuning stage that determines task vectors by leveraging strategies from the pruning and sparse fine-tuning literature with the design of a tailored parameter update selection criterion that promotes weight disentanglement.

## 3 Background

Consider a neural network $f$ with parameters $\boldsymbol{\theta} \in \mathbb{R}^m$, pre-trained on a mixture of tasks $\mathcal{P}$ to obtain parameters $\boldsymbol{\theta}_0$. We are interested in fine-tuning the pre-trained model $f(\cdot, \boldsymbol{\theta}_0)$ on a set of $T$ distinct tasks, with associated non-intersecting task supports $\mathcal{D} = \{\bigcup_{t=1}^{T} \mathcal{D}_t\} \subseteq \mathcal{D}_{\mathcal{P}}$ (*i.e.* $\forall t, t'$ if $t \neq t'$ then $\mathcal{D}_t \cap \mathcal{D}_{t'} = \varnothing$).

In this setting, the core idea behind task arithmetic introduced in Ilharco et al. (2023), is to represent the knowledge acquired for each task $t$ as a *task vector* $\boldsymbol{\tau}_t = \boldsymbol{\theta}_t^* - \boldsymbol{\theta}_0$, obtained by subtracting the initial parameters from the fine-tuned parameters. Intuitively, this vector captures the direction and magnitude of change in the model's weight space induced by learning task $t$. By manipulating tasks via task arithmetic operations we can effectively add, combine, or remove knowledge in the pre-trained model producing actual functional behaviors directly in the parameters space.

As formalized by Ortiz-Jimenez et al. (2023), a network $f$ is said to satisfy the task arithmetic property around $\boldsymbol{\theta}_0$ if it holds

$$f\left(\boldsymbol{x}, \boldsymbol{\theta}_0 + \sum_{t=1}^{T} \alpha_t \boldsymbol{\tau}_t\right) = \begin{cases} f(\boldsymbol{x}, \boldsymbol{\theta}_0 + \alpha_t \boldsymbol{\tau}_t) & \boldsymbol{x} \in \mathcal{D}_t \\ f(\boldsymbol{x}, \boldsymbol{\theta}_0) & \boldsymbol{x} \notin \bigcup_{t=1}^{T} \mathcal{D}_t \end{cases} \tag{1}$$

with scaling factors $(\alpha_1, ..., \alpha_T) \in \mathcal{A} \subseteq \mathbb{R}^T$. This equation essentially states that adding a linear combination of task vectors to the initial parameters $\boldsymbol{\theta}_0$ is equivalent to selectively applying each task-specific modification to the model. In other words, the performance of the pre-trained model on different tasks can be modified independently if the task vector $\boldsymbol{\tau}_t$ does not modify the output of the model outside $\mathcal{D}_t$.

To satisfy the task arithmetic property, Ortiz-Jimenez et al. (2023) states that the model $f$ must exhibit a form of *weight disentanglement* with respect to the set of fine-tuning tasks, *i.e.*, $f$ should behave as a composition of spatially localized components corresponding to functions that vanish outside the task's data support. Equation 1 can be re-written as

$$f\left(\boldsymbol{x}, \boldsymbol{\theta}_0 + \sum_{t=1}^{T} \alpha_t \boldsymbol{\tau}_t\right) = f(\boldsymbol{x}, \boldsymbol{\theta}_0)\mathbb{1}\left(x \notin \bigcup_{t=1}^{T} \mathcal{D}_t\right) + \sum_{t=1}^{T} f(\boldsymbol{x}, \boldsymbol{\theta}_0 + \alpha_t \boldsymbol{\tau}_t)\mathbb{1}(\boldsymbol{x} \in \mathcal{D}_t) \tag{2}$$

$$= g_0(\boldsymbol{x}) + \sum_{t=1}^{T} g_t(\boldsymbol{x}; \alpha_t \boldsymbol{\tau}_t)\ . \tag{3}$$

where $g_t(\boldsymbol{x}, \alpha_t \boldsymbol{\tau}_t) = \mathbf{0}$ for $\boldsymbol{x} \notin \mathcal{D}_t$ and $t = 1, ..., T$, and $g_0(\boldsymbol{x}) = 0$ for $\boldsymbol{x} \in \bigcup_{t=1}^{T} \mathcal{D}_t$, capturing the base behavior of the pre-trained model on inputs outside any of the task support.

Previous works (Tang et al., 2024; Ortiz-Jimenez et al., 2023) have sought to achieve task arithmetic by focusing on linearized neural networks (Ortiz-Jiménez et al., 2021), as they explicitly constrain

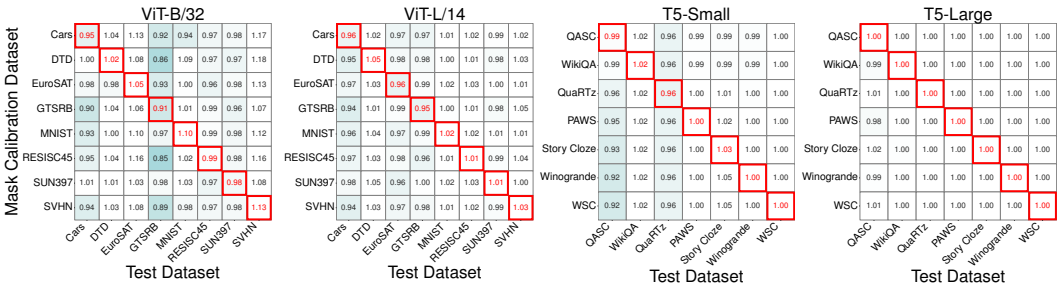


Figure 1: **Pruning parameters with low sensitivity.** The heatmaps illustrate the effect of pruning the parameters with the lowest sensitivity (measured by $\mathbb{E}_{\boldsymbol{x}\in\mathcal{D}_t}[|\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x},\boldsymbol{\theta}_0)|]$) on different tasks across various pre-trained models. Each grid compares the accuracy ratios for models after pruning, with the rows representing the task $\mathcal{D}_t$ used to identify the parameters with the lowest sensitivity and the columns showing the model's performance on each task after pruning those parameters. The accuracy ratios are normalized by the model's performance before pruning. The sparsity ratio (10%) was found as the maximal sparsity that minimally influenced the model's output on the mask calibration dataset.

$f$ to be represented as a linear combination of functions. Specifically, the linearization of $f$ can be achieved by its first-order Taylor expansion centered around $\boldsymbol{\theta}_0$:

$$f(\boldsymbol{x},\boldsymbol{\theta}_0+\alpha_t\boldsymbol{\tau}_t) \approx f_{\text{lin}}(\boldsymbol{x},\boldsymbol{\theta}_0+\alpha_t\boldsymbol{\tau}_t) = f(\boldsymbol{x},\boldsymbol{\theta}_0)+\alpha_t\boldsymbol{\tau}_t^\top\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x},\boldsymbol{\theta}_0)\,. \tag{4}$$

The model $f_{\text{lin}}(\boldsymbol{x},\boldsymbol{\theta}_0+\boldsymbol{\tau}_t)$ represents a linearized neural network. For this type of networks, when combining together multiple task vectors, it holds

$$f_{\text{lin}}\left(\boldsymbol{x},\boldsymbol{\theta}_0+\sum_{t=1}^{T}\alpha_t\boldsymbol{\tau}_t\right) = f(\boldsymbol{x},\boldsymbol{\theta}_0)+\sum_{t=1}^{T}\alpha_t\boldsymbol{\tau}_t^\top\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x},\boldsymbol{\theta}_0)\,. \tag{5}$$

While Equation 5 appears to closely resemble the weight disentanglement condition presented in Equation 3, this similarity is superficial unless each term $\alpha_t\boldsymbol{\tau}_t^\top\nabla\boldsymbol{\theta} f(\boldsymbol{x},\boldsymbol{\theta}_0)$ corresponds to a function that vanishes outside its task data support (*i.e.* it is *localized* within $\mathcal{D}_t$). In the following, we will demonstrate how to efficiently impose a condition of function localization.

## 4 Task-Localized Sparse Fine-Tuning

To formalize the condition of function localization for task arithmetic, we begin by revisiting the linear approximation of $f$ used in linearized fine-tuning. For Equation 5 to satisfy the weight disentanglement conditions in Equation 3, we must ensure that $\boldsymbol{\tau}_t^\top\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x},\boldsymbol{\theta}_0)$ is active (non-zero) only for inputs within the corresponding task support, *i.e.*, $\boldsymbol{x}\in\mathcal{D}_t$. This requirement can be expressed as a set of constraints:

$$\forall\boldsymbol{x}\in\mathcal{D}_{t'\neq t},\quad \boldsymbol{\tau}_t^\top\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x},\boldsymbol{\theta}_0)=0\,. \tag{6}$$

They ensure that when updating the model's weights to learn task $t$, this does not affect how the model processes data from other tasks. To understand this intuitively, remember that $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x},\boldsymbol{\theta}_0)$ measures how much each parameter influences the model's output for a given input $\boldsymbol{x}$. Parameters that have a strong impact on the model's output should remain close to their initial values, so to prevent them from affecting other tasks when combined linearly.

A direct implementation of Equation 6 poses a significant practical challenge. Enforcing the constraint $\forall\boldsymbol{x}\in\mathcal{D}_{t'}$ requires access to data from all other tasks ($t'\neq t$) during the fine-tuning process for task $t$. However, this data is typically unavailable in realistic scenarios.

To solve this issue, we exploit the fact that the pre-trained model has already been exposed to a diverse set of tasks, including those similar to the $T$ tasks we are considering. This is consistent with our initial assumption that $\mathcal{D}=\{\bigcup_{t=1}^{T}\mathcal{D}_t\}\subseteq\mathcal{D}_{\mathcal{P}}$. Under this assumption, we hypothesize that it is possible to efficiently constrain the fine-tuning of the parameters on task $t$ and prepare the model for task arithmetic by using $\mathcal{D}_t$ alone.

**Function localization under individual training constraints.** We hypothesize that pre-trained models obtained on a large corpus of data exhibit shared structure across seemingly distinct tasks and this is reflected in the sensitivity of certain parameters to multiple tasks. We empirically confirm this observation by analyzing the pre-trained model's gradients on each task $t$ and found that parameters

that have low impact on the network's output, where $\forall \boldsymbol{x} \in \mathcal{D}_t$, $\nabla_{\boldsymbol{\theta}_j} f(\boldsymbol{x}, \boldsymbol{\theta}_0) \approx 0$ for $j \in 1, \ldots, m$, are consistently shared across tasks.

To demonstrate this phenomenon, we conduct a pruning experiment, illustrated in Figure 1. We first identify the parameters with the lowest $\nabla_{\boldsymbol{\theta}_j} f(\boldsymbol{x}, \boldsymbol{\theta}_0)$ using data only from task $t$. We then prune these parameters from the network and evaluate its performance on all tasks, including $t$ and other tasks $t' \neq t$. The results show that the pruned model retains its *zero-shot* performance across all tasks, indicating that seemingly unimportant parameters for task $t$ are also not crucial for other tasks and they can be effectively identified independently on the specific task (further validation and discussion of these findings in Appendix A.7).

This observation has important implications for achieving function localization as it allows us to estimate the parameters satisfying constraints for task arithmetic in Equation 6 by simply using data from the available task $t$. Hence, to prevent interference between tasks and enable task arithmetic, we propose a selective *Task-Localized Sparse Fine-Tuning*, (TaLoS) a procedure through which we constrain the parameters with the largest $\nabla_{\boldsymbol{\theta}_j} f(\boldsymbol{x}, \boldsymbol{\theta}_0)$ to remain constant and update the only the ones where $\nabla_{\boldsymbol{\theta}_j} f(\boldsymbol{x}, \boldsymbol{\theta}_0) \approx 0$. This effectively prevents significant changes to the parameters that are highly sensitive to multiple tasks. By fine-tuning only the remaining parameters, we can adapt the model to the new task while minimizing the risk of interference when performing task arithmetic.

To implement our sparse fine-tuning process, we introduce a binary mask $\boldsymbol{c} \in \{0, 1\}^m$ that controls which parameters are updated during fine-tuning. This mask allows us to modify the update rule of any gradient-based optimization algorithm (*e.g.* SGD) to selectively update only certain parameters. Specifically, at each $i$-th iteration, the update rule becomes:

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \gamma[\boldsymbol{c} \odot \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{x}, \boldsymbol{\theta}^{(i-1)}), y)] \,, \tag{7}$$

where $\gamma$ is the learning rate, $\mathcal{L}$ is the loss function, and $\odot$ represents the element-wise product.

To achieve function localization, we want to selectively update the elements of $\boldsymbol{\tau}_t$ corresponding to parameters that have minimal impact on the model's output for other tasks. As discussed earlier, these are the parameters with gradient components close to zero in $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}_0)$. Therefore, we calibrate the mask $\boldsymbol{c}$ using the score $\boldsymbol{s} = \mathbb{E}_{\boldsymbol{x} \in \mathcal{D}_t}[|\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}_0)|] \in \mathbb{R}^m$, which reflects the average sensitivity of each parameter to the input data. Using the indices of the $k$ least sensitive parameters according to $\boldsymbol{s}$, we set the corresponding elements in $\boldsymbol{c}$ to 1, allowing these parameters to be updated during fine-tuning. The remaining elements in $\boldsymbol{c}$ are set to 0, effectively freezing those parameters. Note that the estimation of $\boldsymbol{c}$ may be susceptible to gradient noise (Tanaka et al., 2020). Thus, we follow standard Pruning-at-Initialization practices (Tanaka et al., 2020) and iteratively refine $\boldsymbol{c}$ in multiple rounds (we provide full details of TaLoS, alongside its pseudocode in Appendix A.2). This strategy ensures that we fine-tune the parameters that are least likely to cause interference between tasks, promoting function localization and enabling task arithmetic.

This construction ensures that the contribution of the fine-tuned parameters to the change in the model's output is bounded. Specifically, we have

$$\max_{\boldsymbol{x} \in \mathcal{D}_t} |\boldsymbol{c} \odot (\boldsymbol{\tau}_t^\top \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}_0)| \leq \|\boldsymbol{c} \odot \boldsymbol{\tau}_t\| \cdot \max_{\boldsymbol{x} \in \mathcal{D}_t} \|\boldsymbol{c} \odot \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}_0)\| \leq k^2 \cdot \mu \cdot \eta \,. \tag{8}$$

Here, $\eta = \max_{\boldsymbol{x}} |\nabla_{\boldsymbol{\theta}_k} f(\boldsymbol{x}, \boldsymbol{\theta}_0)|$ represents the magnitude of the $k$-th largest gradient component, effectively capturing the maximum sensitivity of the fine-tuned parameters to input data, and $\mu = \max_j |\boldsymbol{c}_j \odot \boldsymbol{\tau}_{t_j}|$ represents the maximum change in any of the updated parameters during fine-tuning. The inequality 8 provides an upper bound on the quantity that Equation 6 aims to cancel to avoid task interference, thus indirectly controlling the degree of function localization. By selecting a smaller value of $k$, we reduce the number of parameters being updated and consequently lower this bound. As detailed in Appendix A.1, we tuned $k$ at the task level, resulting in optimal sparsity ratios between 90% and 99%. Further ablations on the effect of $k$ are reported in Appendix A.5.

**Sparsity promotes linear behavior.** As shown in Section 3, the constraints in Equation 6 are derived under the assumption of linearized behavior, which is often achieved through explicit network linearization Ortiz-Jimenez et al. (2023) and can be computationally expensive. We highlight that our sparse fine-tuning formulation *inherently promotes linearized behavior* without requiring explicitly linearizing the network. This is observed when the change in network output post-training can be accurately approximated by its first-order Taylor expansion (Malladi et al., 2023b; Ortiz-Jimenez et al., 2023). Thus, explicit network linearization becomes unnecessary, reducing computational overhead while maintaining the benefits of linearized fine-tuning. Mathematically, as $\|\boldsymbol{\theta}_t - \boldsymbol{\theta}_0\|^2 = \|\boldsymbol{\tau}_t\|^2 \to 0$

$$f(\boldsymbol{x}, \boldsymbol{\theta}_0 + \boldsymbol{\tau}_t) = f(\boldsymbol{x}, \boldsymbol{\theta}_0) + \boldsymbol{\tau}_t^\top \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}_0) + \mathcal{O}(\|\boldsymbol{\tau}_t\|^2), \tag{9}$$

Provided that the gradient updates of the unmasked parameters are similar in magnitude to the ones of full fine-tuning, sparse fine-tuning naturally leads to a smaller $\|\boldsymbol{\tau}_t\|^2$ as it involves updating a limited number of parameters relative to the overall size of the model (Yu et al., 2024; Yadav et al., 2023; Ortiz-Jimenez et al., 2023). This inherent property of sparse fine-tuning increases the likelihood that the linearization condition will hold, potentially rendering explicit network linearization unnecessary. Indeed, we follow Ortiz-Jimenez et al. (2023) to experimentally confirm this claim, as discussed in Appendix A.4 (see also Figure 6). As a result, the set of constraints in Equation 6 are applicable to our method, without incurring the computational overhead of explicit linearization. Note that the quality of the linear approximation can also be theoretically analyzed, as was demonstrated by several studies in the neural tangent kernel literature Jacot et al. (2018); Lee et al. (2019); Arora et al. (2019); Mei et al. (2019).

# 5 Experiments

Our experimental evaluation focuses on the established Task Arithmetic framework outlined by Ilharco et al. (2022; 2023), specifically targeting Task Addition and Task Negation, encompassing both language and vision domains. In the following we describe the baselines we compared our `TaLoS` against. Further details regarding the experimental setups, the relevant metrics, the implementation of the experiments, as well as the data and architectures used, are deferred to Appendix A.1.

**Baselines.** We consider three families of methods as references. (i) **Full fine-tuning** methods aim to produce task vectors $\boldsymbol{\tau}_t$ by fine-tuning all the parameters of the network. Specifically, *Non-linear fine-tuning* (FT) (Ilharco et al., 2022; 2023) minimizes a standard cross-entropy loss, while *Linearized FT* fine-tunes the linearized counterpart of the network, as in Ortiz-Jimenez et al. (2023). (ii) **Post-hoc** methods refine $\boldsymbol{\tau}_t$ after it has been obtained via fine-tuning (as prescribed by the respective methods, we apply these post-hoc approaches on non-linear FT checkpoints). *TIES-Merging* (Yadav et al., 2023) reduces redundancy in $\boldsymbol{\tau}_t$ by magnitude pruning, keeping only the top-$k$ highest magnitude parameters, and addressing sign conflicts when merging task vectors. *TALL Mask / Consensus* (Wang et al., 2024) identifies task-specific parameters in $\boldsymbol{\tau}_t$ by comparing them to the sum of task vectors. It then merges multiple task vectors by using an element-wise OR operation between masks to further identify and remove conflicting parameters. *DARE* (Yu et al., 2024) randomly sparsifies $\boldsymbol{\tau}_t$ to eliminate redundancy and upweights the remaining parameters based on the percentage that was removed. *Breadcrumbs* (Davari & Belilovsky, 2024) reduces redundancy using magnitude pruning and eliminates weight outliers within the retained top-$k$ parameters. Although these methods have been presented for task addition, we also test their ability of handling task negation. (iii) **Parameter-efficient fine-tuning (PEFT)** methods aim to obtain task vectors by efficiently fine-tuning the network, using far fewer resources compared to full fine-tuning. We compare against *L-LoRA* (Tang et al., 2024), which applies linearized low-rank adapters to the $\boldsymbol{Q}$ and $\boldsymbol{V}$ matrices in self-attention layers. This approach was specifically designed for Task Arithmetic and offers superior performance over standard LoRA without adding extra computational costs. For sparse fine-tuning, we use *LoTA* (Panda et al., 2024), a method that leverages the Lottery Ticket hypothesis (Frankle & Carbin, 2019) to select the top-$k$ parameters when sparsely fine-tuning the network, making it suitable for model merging.

## 5.1 Task Arithmetic Results

We thoroughly evaluate `TaLoS` on its ability to derive task vectors that enable model editing through simple arithmetic operations on model parameters.

**Task Addition.** In this benchmark, the sum of the task vectors $\sum_t \alpha_t \boldsymbol{\tau}_t$ is added to a pre-trained checkpoint to produce a multi-task model $f(\cdot, \boldsymbol{\theta}_0 + \sum_t \alpha_t \boldsymbol{\tau}_t)$. The success is measured in terms of the maximum average accuracy over the different tasks. As done by Ortiz-Jimenez et al. (2023); Tang et al. (2024), we also report the average normalized accuracy over the tasks. The normalization is performed with respect to the single-task accuracies achieved by the model fine-tuned on each task (see Appendix A.1). The results in Table 1 demonstrate the effectiveness of our proposed method across various model scales and modalities. `TaLoS` consistently outperforms existing approaches, with evident improvements in normalized accuracy of 1.88% to 4.65% over the second best method across all model variants. Such a metric provides insights into the outstanding ability of `TaLoS` to maximize the benefits of model combination while mitigating interference.

For vision models, `TaLoS` exhibits strong performance across all scales, with absolute accuracy gains of up to 2.61% over the closest competitor. In NLP, `TaLoS` maintains its leading position,

| Method | ViT-B/32 | | ViT-B/16 | | ViT-L/14 | | T5-Small | | T5-Base | | T5-Large | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Abs. (↑) | Norm. (↑) | Abs. (↑) | Norm. (↑) | Abs. (↑) | Norm. (↑) | Abs. (↑) | Norm. (↑) | Abs. (↑) | Norm. (↑) | Abs. (↑) | Norm. (↑) |
| Pre-trained (Zero-shot) | 47.72 | - | 55.83 | - | 65.47 | - | 55.70 | - | 53.51 | - | 51.71 | - |
| **Full Fine-tuning Methods** | | | | | | | | | | | | |
| Non-linear FT (Ilharco et al., 2023) | 71.25 | 76.94 | 72.85 | 77.17 | 86.09 | 90.14 | **65.04** | 87.98 | 74.20 | 90.63 | 75.37 | 85.25 |
| Linearized FT (Ortiz-Jimenez et al., 2023) | 76.70 | 85.86 | 80.01 | 87.29 | 88.29 | 93.01 | 64.13 | 86.62 | 74.69 | 92.12 | 69.38 | 78.95 |
| **Post-hoc Methods** | | | | | | | | | | | | |
| TIES-Merging (Yadav et al., 2023) | 74.79 | 82.84 | 77.09 | 82.13 | 88.16 | 92.56 | 62.53 | 94.83 | 70.74 | 92.37 | 74.30 | 86.36 |
| TALL Mask / Consensus (Wang et al., 2024) | 74.55 | 80.27 | 74.92 | 79.12 | 86.89 | 90.81 | 63.61 | 95.34 | 73.31 | 91.60 | 77.31 | 87.84 |
| DARE (Yu et al., 2024) | 70.88 | 76.59 | 73.08 | 77.51 | 85.95 | 90.04 | 63.89 | 89.09 | 74.26 | 91.49 | 76.20 | 86.51 |
| Breadcrumbs (Davari & Belilovsky, 2024) | 69.39 | 79.51 | 71.93 | 78.94 | 84.78 | 92.97 | 61.19 | 92.23 | 73.89 | 92.70 | 73.41 | 87.07 |
| **Parameter-efficient Fine-tuning Methods** | | | | | | | | | | | | |
| L-LoRA (Tang et al., 2024) | 78.00 | 86.08 | 80.61 | 85.83 | 87.77 | 91.87 | 60.29 | 94.46 | 68.76 | 91.98 | 72.10 | 87.78 |
| LoTA (Panda et al., 2024) | 64.94 | 74.37 | 79.11 | 83.97 | 87.66 | 91.69 | 64.21 | 87.92 | 74.31 | 92.25 | 75.84 | 88.14 |
| **TaLoS (Ours)** | **79.67** [+1.67] | **90.73** [+4.65] | **82.60** [+1.99] | **91.41** [+4.12] | **88.37** [+0.08] | **95.20** [+2.19] | **65.04** [ 0.00] | **97.22** [+1.88] | **75.93** [+1.24] | **95.87** [+3.17] | **79.07** [+1.76] | **90.61** [+2.47] |

Table 1: **Task Addition results.** Average absolute accuracies (%) and normalized accuracies (%) of different CLIP ViTs and T5 pre-trained models edited by adding task vectors on each of the downstream tasks. We normalize performance of each method by their single-task accuracy. **Bold** indicates the best results. Underline the second best.

| Method | ViT-B/32 | | ViT-B/16 | | ViT-L/14 | | T5-Small | | T5-Base | | T5-Large | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Targ. (↓) | Cont. (↑) | Targ. (↓) | Cont. (↑) | Targ. (↓) | Cont. (↑) | Targ. (↓) | Cont. (↑) | Targ. (↓) | Cont. (↑) | Targ. (↓) | Cont. (↑) |
| Pre-trained (Zero-shot) | 47.72 | 63.26 | 55.83 | 68.37 | 65.47 | 75.53 | 55.70 | 45.70 | 53.51 | 45.30 | 51.71 | 45.70 |
| **Full Fine-tuning Methods** | | | | | | | | | | | | |
| Non-linear FT (Ilharco et al., 2023) | 24.04 | 60.36 | 20.36 | 64.79 | 20.61 | 72.72 | 43.06 | 45.47 | 40.06 | 45.16 | 41.54 | 45.49 |
| Linearized FT (Ortiz-Jimenez et al., 2023) | 11.20 | 60.74 | 10.97 | 65.55 | 10.86 | 72.43 | 44.47 | 44.94 | 40.16 | 45.27 | 41.37 | **45.70** |
| **Post-hoc Methods** | | | | | | | | | | | | |
| TIES-Merging (Yadav et al., 2023) | 21.94 | **61.49** | 19.72 | 65.69 | 24.50 | 73.41 | 55.01 | 45.30 | 40.30 | 45.13 | 46.19 | 45.56 |
| TALL Mask / Consensus (Wang et al., 2024) | 23.31 | 60.54 | 20.71 | 65.17 | 22.33 | 73.30 | 43.43 | 45.41 | 40.14 | 45.20 | 41.26 | 45.59 |
| DARE (Yu et al., 2024) | 25.04 | 60.60 | 22.22 | 64.98 | 20.94 | 72.66 | 42.53 | 45.36 | 40.24 | 45.16 | 41.29 | **45.70** |
| Breadcrumbs (Davari & Belilovsky, 2024) | 24.27 | 60.58 | 21.60 | 65.22 | 20.69 | 72.95 | 53.03 | 45.19 | 40.46 | 45.14 | 41.49 | 45.51 |
| **Parameter-efficient Fine-tuning Methods** | | | | | | | | | | | | |
| L-LoRA (Tang et al., 2024) | 17.29 | 60.75 | 19.33 | 65.69 | 19.39 | 73.14 | 55.30 | 45.24 | 51.33 | 45.10 | 48.37 | 45.51 |
| LoTA (Panda et al., 2024) | 21.09 | 61.01 | 17.76 | 65.60 | 22.11 | 73.21 | 54.70 | 45.13 | 40.50 | 45.24 | 44.33 | 45.47 |
| **TaLoS (Ours)** | **11.03** [+0.17] | 60.69 [-0.80] | **10.58** [+0.39] | **66.11** [+0.42] | **10.68** [+0.18] | **73.63** [+0.22] | **39.64** [+2.89] | **45.67** [+0.20] | **38.49** [+1.57] | **45.28** [+0.01] | **37.20** [+4.06] | **45.70** [ 0.00] |

Table 2: **Task Negation results.** Average minimal accuracy (%) of different CLIP ViTs and T5 pre-trained models edited by subtracting a task vector from a target task while retaining at least 95% of their performance on the control task. We average the minimal accuracy over each of the downstream tasks. **Bold** indicates the best results. Underline the second best.

although the gains are less striking than in vision experiments. Nevertheless, the improvements are particularly pronounced in larger models, suggesting that TaLoS scales well with model size. Notably, TaLoS's performance surpasses both full fine-tuning and post-hoc methods across the board. This suggests that our parameter-efficient approach can achieve superior results while potentially reducing computational costs, a crucial factor when working with large-scale models.

**Task Negation.** In this benchmark a task vector $\boldsymbol{\tau}_t$ is subtracted from the pre-trained checkpoint to reduce the performance on task $t$, producing the model $f(\cdot, \boldsymbol{\theta}_0 - \alpha_t \boldsymbol{\tau}_t)$. By following Ortiz-Jimenez et al. (2023), the success is measured in terms of the maximum drop in accuracy on the forgetting task that retains at least 95% of the accuracy on the control task. Results are averaged over tasks and presented in Table 2. For vision models, TaLoS achieves the lowest target task accuracies while maintaining high control task performance, indicating superior ability to selectively remove targeted task information. For T5 models, all methods, including TaLoS, face significant challenges in Task Negation. The results show a much tighter clustering of performance across different approaches. This suggests that negating specific language tasks without substantially impacting the control task accuracy is inherently more difficult than in vision models. Despite this challenge, TaLoS still manages to achieve the best balance between target and control task performance.

### 5.2 Weight Disentanglement and Localization

The improved localization provided by TaLoS seems to play a crucial role in driving effective task arithmetic. Here we delve deeper into this aspect with tailored analyses. First, we assess how well the weight disentanglement property holds. Then, for each training recipe, we evaluate the degree of task component localization on each task.

**Weight disentanglement error visualization.** Ortiz-Jimenez et al. (2023); Tang et al. (2024) proposed to evaluate the *disentanglement error* defined as

$$\xi(\alpha_1, \alpha_2) = \sum_{t=1}^{2} \mathbb{E}_{\boldsymbol{x} \in \mathcal{D}_t} [\text{dist}(f(\boldsymbol{x}, \boldsymbol{\theta}_0 + \alpha_1 \boldsymbol{\tau}_1), f(\boldsymbol{x}, \boldsymbol{\theta}_0 + \alpha_1 \boldsymbol{\tau}_1 + \alpha_2 \boldsymbol{\tau}_2))] \tag{10}$$

where the *prediction error* $\text{dist}(y_1, y_2) = \mathbb{1}(y_1 \neq y_2)$ is taken as the distance metric. Generally, given a pair $(\alpha_1, \alpha_2)$, the smaller the value of $\xi(\alpha_1, \alpha_2)$ the more weight disentangled a model is.

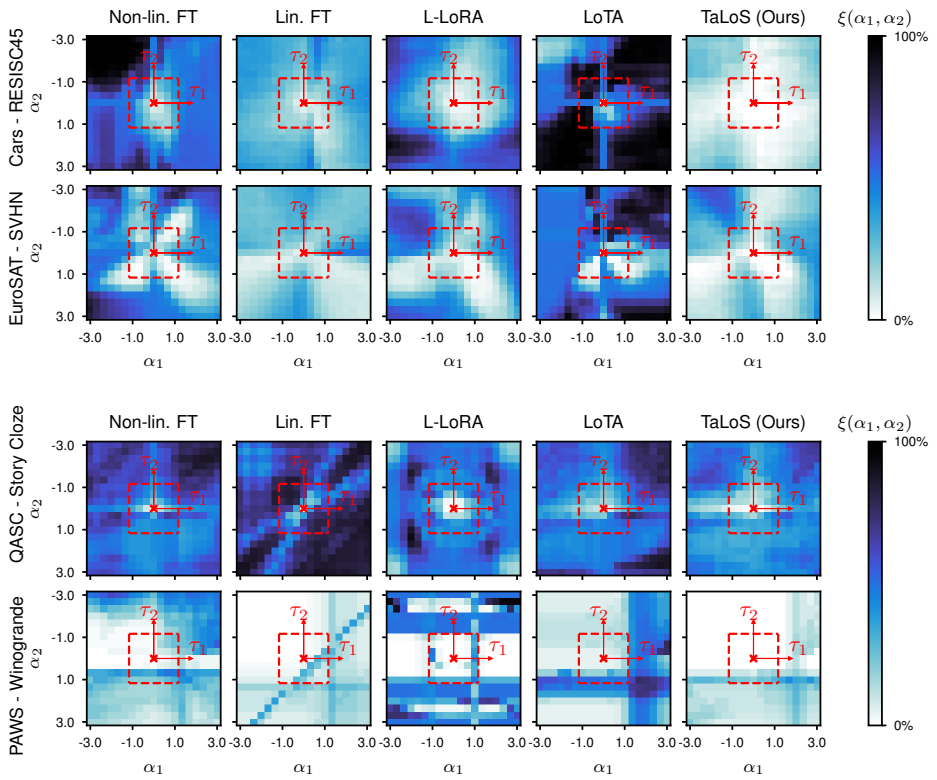


Figure 2: **Visualizing weight disentanglement error.** The heatmaps illustrate the disentanglement error $\xi(\alpha_1, \alpha_2)$ of each fine-tuning strategy on both a CLIP ViT-B/32 model (top) and a T5-Small model (bottom) across two task pairs. Lighter areas highlight regions of the weight space where disentanglement is more pronounced. The red box indicates the search space within which the optimal $\alpha$ values were searched (refer to Appendix A.1). We chose the task pairs to visualize by following Ortiz-Jimenez et al. (2023) for vision and a criterion akin to the one used in Tang et al. (2024) for language.

Maintaining a low disentanglement error as $\alpha_1$ and $\alpha_2$ increase provides an even stronger evidence of the weight disentanglement property.

In Figure 2, we report $\xi(\alpha_1, \alpha_2)$ across different fine-tuning strategies for both the CLIP ViT-B/32 and T5-Small models on two task pairs. Overall there is a clear difference in disentanglement patterns between vision and language models. For the latter, the patterns are more consistent across strategies, which may explain why the differences in task arithmetic performance are notable in vision experiments and less pronounced in language experiments (ref. to Tables 1, 2).

By focusing on vision models we observe that Linearized FT, L-LoRA, and our approach demonstrate improved disentanglement (indicated by lighter regions) than non-linear fine-tuning, with our method performing the best overall. We remind that L-LoRA approximate the behavior of Linearized FT via adapters but still lacks to optimize the task localization property. Interestingly, LoTA shows a much lower degree of disentanglement. We remark that this approach selects and updates task-specific parameters while `TaLoS` focuses on task-generic ones and this difference accounts for the observed behavior.

For language, Linearized FT and L-LoRA yield mixed results depending on the pairs of considered tasks. LoTA seems able to improve over non-linearized FT but with different extents across tasks and it is consistently outperformed by `TaLoS`.

**Function localization.** We experimentally assess the function localization property of `TaLoS` by comparing it with other fine-tuning methods. From the definition in Equation 6, we know that when this property holds, each task activates only for its specific data support. Thus, we should observe an advantage in the prediction output when testing on that task, and the same performance of the pre-trained model for all the others tasks. Figure 3 confirms the expected behavior for `TaLoS` in

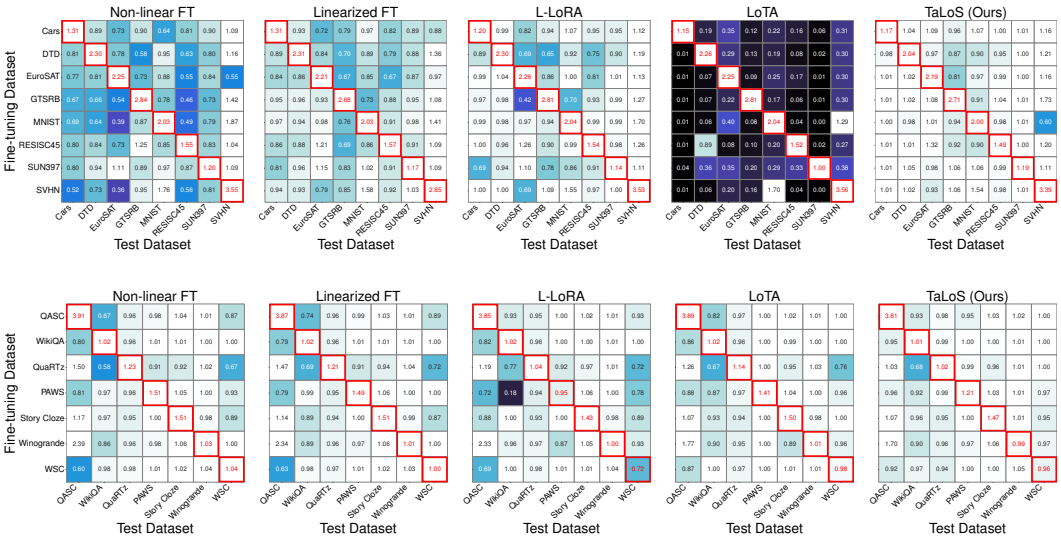


Figure 3: **Function localization.** The heatmaps present the accuracy ratios for fine-tuned models across tasks for CLIP ViT-B/32 (top) and T5-Small (bottom) models. Each row indicates a model fine-tuned on a specific task, with columns representing its performance on different test datasets. Accuracy ratios are normalized by the pre-trained model's performance. Lighter colors indicate better performance, suggesting minimal interference between the fine-tuned model and other tasks' input spaces. The red diagonal highlights each model's test performance on its specific fine-tuning task.


Figure 4: **Visualization of mask calibration.** Percentage of parameters selected for sparse fine-tuning in a transformer block of a ViT-B/32 (left) and a T5-Small (right) models, after our method's mask calibration vs. LoTA's mask calibration, at 90% sparsity. On ViT-B/32, we calibrate the masks on the Cars dataset (Krause et al., 2013), while on T5-Small we use QASC (Khot et al., 2020). Full visualizations of all masked layers are reported in Appendix A.3.

vision, while the competitors display more interference between tasks, as indicated by darker hues off the diagonal. Interestingly, for NLP tasks all methods exhibit natural function localization, as reflected by the lighter regions in the figure. This provides us the opportunity to remark the importance of extensive model analysis as conclusions drawn from a single domain where linearization is sufficient might be misleading.

### 5.3 Weight Sparsity Structure and Efficiency

**Visualizing task vector masks.** To understand the nature of our sparse fine-tuning approach, we analyze the structure of the masks $\boldsymbol{c}$ calibrated using `TaLoS` and compare it with the ones produced by LoTA. Figure 4 provides a visualization of the layer-wise percentage of parameters selected for sparse fine-tuning in a transformer block of a ViT-B/32 and a T5-Small models. The results reveal distinct patterns in parameter selection between `TaLoS` and LoTA across both models. `TaLoS` exhibits a highly structured selection, predominantly preserving parameters in the multihead self-attention layer, particularly in the $\boldsymbol{Q}$ and $\boldsymbol{K}$ components. In contrast, LoTA's selection appears more distributed across different layers of the transformer block. Interestingly, our analysis reveals some notable contrasts with L-LoRA (Tang et al., 2024), a method specifically designed for task arithmetic. While L-LoRA arbitrarily fine-tunes the $\boldsymbol{Q}$ and $\boldsymbol{V}$ components, our findings suggest that, generally, $\boldsymbol{Q}$ and $\boldsymbol{K}$ play a more significant role in task arithmetic than $\boldsymbol{V}$ in the multihead self-attention layers. Additionally, for CLIP ViT-B/32 biases also seemingly play a crucial role for function localization.

| Method | Effective Cost of Fine-tuning | | | | Task Addition | | Task Negation | |
|---|---|---|---|---|---|---|---|---|
| | Forward-Backward Pass Time (s) | Optim. Step Time (s) | Tot. Iteration Time (s) | Peak Memory Usage (GiB) | Abs. (↑) | Norm. (↑) | Targ. (↓) | Cont. (↑) |
| **ViT-B/32** | | | | | | | | |
| Non-linear FT (Ilharco et al., 2023) | 0.3608 ± 0.0036 | 0.0114 ± 0.0010 | 0.3722 ± 0.0037 | 6.5 | 71.25 | 76.94 | 24.04 | 60.36 |
| Linearized FT (Ortiz-Jimenez et al., 2023) | 0.6858 ± 0.0042 | 0.0103 ± 0.0020 | 0.6961 ± 0.0047 | 10.2 | 76.70 | 85.86 | 11.20 | 60.74 |
| L-LoRA (Tang et al., 2024) | 0.3270 ± 0.0076 | **0.0036** ± 0.0032 | 0.3306 ± 0.0082 | 5.3 | 78.00 | 86.08 | 17.29 | 60.75 |
| LoTA (Panda et al., 2024) | 0.3289 ± 0.0041 | 0.1269 ± 0.0050 | 0.4558 ± 0.0065 | 6.8 | 64.94 | 74.37 | 21.09 | **61.01** |
| **TaLoS (Ours)** | **0.1256** ± 0.0045 | 0.0388 ± 0.0040 | **0.1644** ± 0.0060 | **4.7** | **79.67** | **90.73** | **11.03** | 60.69 |
| **ViT-L/14** | | | | | | | | |
| Non-linear FT (Ilharco et al., 2023) | 1.2174 ± 0.0097 | 0.0156 ± 0.0055 | 1.2330 ± 0.0112 | 18.6 | 86.09 | 90.14 | 20.61 | 72.72 |
| Linearized FT (Ortiz-Jimenez et al., 2023) | 1.6200 ± 0.0067 | 0.0262 ± 0.0082 | 1.6462 ± 0.0106 | 21.3 | 88.29 | 93.01 | 10.86 | 72.43 |
| L-LoRA (Tang et al., 2024) | 0.5153 ± 0.0077 | **0.0082** ± 0.0015 | 0.5235 ± 0.0078 | 9.7 | 87.77 | 91.87 | 19.39 | 73.14 |
| LoTA (Panda et al., 2024) | 0.8438 ± 0.0052 | 0.4449 ± 0.0074 | 1.2887 ± 0.0090 | 15.4 | 87.66 | 91.69 | 22.11 | 73.21 |
| **TaLoS (Ours)** | **0.1891** ± 0.0039 | 0.1372 ± 0.0036 | **0.3263** ± 0.0053 | **7.8** | **88.37** | **95.20** | **10.68** | **73.63** |
| **T5-Large** | | | | | | | | |
| Non-linear FT (Ilharco et al., 2023) | 0.9047 ± 0.0068 | 0.0894 ± 0.0034 | 0.9941 ± 0.0076 | 30.0 | 75.37 | 85.25 | 41.54 | 45.49 |
| Linearized FT (Ortiz-Jimenez et al., 2023) | 1.7683 ± 0.0084 | 0.1170 ± 0.0060 | 1.8853 ± 0.0103 | 35.1 | 69.38 | 78.95 | 41.37 | **45.70** |
| L-LoRA (Tang et al., 2024) | 0.7452 ± 0.0084 | **0.0136** ± 0.0029 | 0.7588 ± 0.0089 | 18.2 | 72.10 | 87.78 | 48.37 | 45.51 |
| LoTA (Panda et al., 2024) | 0.8526 ± 0.0043 | 0.3842 ± 0.0019 | 1.2368 ± 0.0047 | 32.1 | 75.84 | 88.14 | 44.33 | 45.47 |
| **TaLoS (Ours)** | **0.4358** ± 0.0075 | 0.0509 ± 0.0046 | **0.4867** ± 0.0088 | **12.1** | **79.07** | **90.61** | **37.20** | **45.70** |

Table 3: **Computational cost and memory footprint of fine-tuning.** Average iteration time (in seconds) and peak memory usage (in Gibibytes) of different fine-tuning approaches on CLIP ViT-B/32, ViT-L/14 and T5-Large models, alongside their performance on the task arithmetic benchmark. To improve granularity, we report also the average forward-backward time of a single iteration and the average step time of the optimizer. We separate full fine-tuning methods from parameter-efficient fine-tuning methods. Further details on the resource monitoring process can be found in Appendix A.1. **Bold** indicates the best results. Underline the second best.

This structured sparsity not only provides insights into our method's mask calibration mechanism but also hints at potential efficiency gains, which we explore further in the following.

**Computational cost and memory footprint.** The observed structured sparsity pattern of TaLoS suggests that it also provides a highly efficient task arithmetic fine-tuning strategy. To verify it we performed a comparative analysis of the computational cost and memory footprint of TaLoS against several fine-tuning methods.

In Table 3 we present the collected time and memory costs with detailed average time (in seconds) for a single training iteration's forward and backward pass. This is separated because approaches like Linearized FT and L-LoRA involve specialized forward passes that require Jacobian-vector products with respect to LoTA and TaLoS, which operate similarly to non-linear FT. We also report the time (in seconds) spent by the optimizer updating parameters, as LoTA and TaLoS require an additional mask-based element-wise multiplication to prevent updates to certain parameters by masking gradients. Additionally, we provide the total time (sum of these two values) and the peak memory usage (in Gibibytes) recorded during fine-tuning. Overall, the ability to freeze a large number of parameters, thanks to well-structured mask sparsity of our approach improves the total iteration time. Although our method has a slower optimizer step compared to other approaches, the faster forward-backward pass compensates, making TaLoS the leading method. In terms of memory usage, the benefits are especially notable for large models, where only a small subset of parameters requires fine-tuning, thus, yielding pronounced savings.

## 6 Conclusion

In this work we have proposed TaLoS, an efficient and effective strategy to edit pre-trained models in the framework of task arithmetic. We started from the observation that the parameters showing the least variation in the fine-tuning process of a single task are also those minimally relevant for other tasks. Thus, we have leveraged them through a sparse learning process that promotes task localization and avoids task interference. A thorough experimental analysis across vision and language domains confirmed that TaLoS yields state-of-the-art results in task addition and negation, showing a significant efficiency advantage over competitors. Moreover, with a tailored set of evaluations we assessed model linearization and function localization properties, providing insights on the inner functionig of our approach.

Overall, we have discussed how preserving the regularities provided by a large scale pre-trained model are sufficient to maintain weight disentaglement and observe beneficial effects in task arithmentic. Future work may investigate whether explicitly enforcing localization constraints during fine-tuning could enhance performance and further advance model editing capabilities.

## REPRODUCIBILITY STATEMENT

We have made significant efforts to ensure the reproducibility of our results. Full implementation details are provided in Appendix A.1. Pseudocode for our algorithm is included in Appendix A.2 to clarify key steps, as well as practical design choices to address potential challenges in implementing our experiments. Additionally, we provide the complete codebase as a supplementary material of our submission, which includes scripts to reproduce all the results. Upon acceptance, we will publicly release the code to further facilitate reproducibility.

## REFERENCES

Shun-ichi Amari. Neural learning in structured parameter spaces-natural riemannian gradient. In *Advances in Neural Information Processing Systems (NeurIPS)*, 1996. URL https://proceedings.neurips.cc/paper_files/paper/1996/file/39e4973ba3321b80f37d9b55f63ed8b8-Paper.pdf.

Alan Ansell, Edoardo Ponti, Anna Korhonen, and Ivan Vulić. Composable sparse fine-tuning for cross-lingual transfer. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1778–1796, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.125. URL https://aclanthology.org/2022.acl-long.125.

Alan Ansell, Ivan Vulić, Hannah Sterz, Anna Korhonen, and Edoardo M. Ponti. Scaling sparse fine-tuning to large language models, 2024.

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/dbc4d84bfcfe2284ba11beffb853a8c4-Paper.pdf.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1–9, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.1. URL https://aclanthology.org/2022.acl-short.1.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European conference on computer vision (ECCV)*, 2018. URL https://arxiv.org/abs/1801.10112.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016. URL https://arxiv.org/pdf/1604.06174.

Gong Cheng, Junwei Han, and Xiaoqiang Lu. Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the IEEE*, 105(10):1865–1883, 2017. URL https://ieeexplore.ieee.org/document/7891544.

Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. URL https://openaccess.thecvf.com/content_cvpr_2014/papers/Cimpoi_Describing_Textures_in_2014_CVPR_paper.pdf.

Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In *Machine learning challenges workshop*, 2005. URL https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=e808f28d411a958c5db81ceb111beb2638698f47.

Mohammad Reza Davari and Eugene Belilovsky. Model breadcrumbs: Scaling multi-task model merging with sparse masks. In *European Conference on Computer Vision (ECCV)*, 2024. URL https://arxiv.org/abs/2312.06795.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE conference on computer vision and pattern recognition (CVPR)*, 2009. URL https://projet.liris.cnrs.fr/imagine/pub/proceedings/CVPR-2009/data/papers/0103.pdf.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://arxiv.org/abs/2010.11929.

Ronald A Fisher. On the mathematical foundations of theoretical statistics. *Philosophical transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character*, 222(594-604):309–368, 1922. URL https://royalsocietypublishing.org/doi/10.1098/rsta.1922.0009.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

Samir Yitzhak Gadre, Gabriel Ilharco, Alex Fang, Jonathan Hayase, Georgios Smyrnis, Thao Nguyen, Ryan Marten, Mitchell Wortsman, Dhruba Ghosh, Jieyu Zhang, et al. Datacomp: In search of the next generation of multimodal datasets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2304.14108.

Demi Guo, Alexander Rush, and Yoon Kim. Parameter-efficient transfer learning with diff pruning. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4884–4896, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.378. URL https://aclanthology.org/2021.acl-long.378.

Marton Havasi, Rodolphe Jenatton, Stanislav Fort, Jeremiah Zhe Liu, Jasper Snoek, Balaji Lakshminarayanan, Andrew M Dai, and Dustin Tran. Training independent subnetworks for robust prediction. *arXiv preprint arXiv:2010.06610*, 2020.

Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226, 2019. URL https://ieeexplore.ieee.org/document/8736785.

Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations. *arXiv preprint arXiv:2212.13345*, 2022. URL https://arxiv.org/pdf/2212.13345.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *ICML*, 2019.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *ICLR*, 2022.

Gabriel Ilharco, Mitchell Wortsman, Samir Yitzhak Gadre, Shuran Song, Hannaneh Hajishirzi, Simon Kornblith, Ali Farhadi, and Ludwig Schmidt. Patching open-vocabulary models by interpolating weights. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2208.05592.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2110.08207.

Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/5a4be1fa34e62bb8a6ec6b91d2462f5a-Paper.pdf.

Xisen Jin, Xiang Ren, Daniel Preotiuc-Pietro, and Pengxiang Cheng. Dataless knowledge fusion by merging weights of language models. In *ICLR*, 2023.

Tushar Khot, Peter Clark, Michal Guerquin, Peter Jansen, and Ashish Sabharwal. Qasc: A dataset for question answering via sentence composition. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020. URL https://arxiv.org/abs/1910.11473.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4015–4026, 2023.

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2013. URL https://www.cv-foundation.org/openaccess/content_iccv_workshops_2013/W19/papers/Krause_3D_Object_Representations_2013_ICCV_paper.pdf.

Woosuk Kwon, Sehoon Kim, Michael W Mahoney, Joseph Hassoun, Kurt Keutzer, and Amir Gholami. A fast post-training pruning framework for transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/987bed997ab668f91c822a09bce3ea12-Paper-Conference.pdf.

Yann LeCun. The mnist database of handwritten digits, 1998. URL http://yann.lecun.com/exdb/mnist/.

Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/0d1a9651497a38d8b1c3871c84528bd4-Paper.pdf.

Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Thirteenth international conference on the principles of knowledge representation and reasoning*, 2012. URL https://cdn.aaai.org/ocs/4492/4492-21843-1-PB.pdf.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *preprint arXiv:2101.00190*, 2021.

Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021.

Baohao Liao, Yan Meng, and Christof Monz. Parameter-efficient fine-tuning without introducing new latency. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4242–4260, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.233. URL https://aclanthology.org/2023.acl-long.233.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965, 2022.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32100–32121. PMLR, 21–27 Jul 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. 2023a. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/a627810151be4d13f907ac898ff7e948-Paper-Conference.pdf.

Sadhika Malladi, Alexander Wettig, Dingli Yu, Danqi Chen, and Sanjeev Arora. A kernel-based view of language model fine-tuning. In *International Conference on Machine Learning (ICML)*, 2023b. URL https://proceedings.mlr.press/v202/malladi23a/malladi23a.pdf.

Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 7765–7773, 2018.

Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 67–82, 2018.

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

Michael Matena and Colin Raffel. Merging models with fisher-weighted averaging. In *neurIPS*, 2022.

Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Mean-field theory of two-layers neural networks: dimension-free bounds and kernel limit. In *Conference on Learning Theory*, 2019. URL https://proceedings.mlr.press/v99/mei19a/mei19a.pdf.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *Advances in Neural Information Processing Systems (NeurIPS) Workshops*, 2011. URL https://static.googleusercontent.com/media/research.google.com/it//pubs/archive/37648.pdf.

Guillermo Ortiz-Jiménez, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. What can linearized neural networks actually say about generalization? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. URL https://proceedings.neurips.cc/paper/2021/file/4b5deb9a14d66ab0acc3b8a2360cde7c-Paper.pdf.

Guillermo Ortiz-Jimenez, Alessandro Favero, and Pascal Frossard. Task arithmetic in the tangent space: Improved editing of pre-trained models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://openreview.net/pdf?id=0A9f2jZDGW.

Ashwinee Panda, Berivan Isik, Xiangyu Qi, Sanmi Koyejo, Tsachy Weissman, and Prateek Mittal. Lottery ticket adaptation: Mitigating destructive interference in llms. *arXiv preprint arXiv:2406.16797*, 2024. URL https://arxiv.org/abs/2406.16797.

R Pascanu and Y Bengio. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013. URL https://arxiv.org/abs/1301.3584.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. Adapterhub: A framework for adapting transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020): Systems Demonstrations*, pp. 46–54, Online, 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.7.

Clifton Poth, Hannah Sterz, Indraneil Paul, Sukannya Purkayastha, Leon Engländer, Timo Imhof, Ivan Vulić, Sebastian Ruder, Iryna Gurevych, and Jonas Pfeiffer. Adapters: A unified library for parameter-efficient and modular transfer learning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 149–160, Singapore, December 2023. Association for Computational Linguistics. URL https://aclanthology.org/2023.emnlp-demo.13.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*, 2021. URL https://arxiv.org/abs/2103.00020.

Colin Raffel. Building machine learning models like open source software. *Communications of the ACM*, 66(2):38–40, 2023.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021. URL https://dl.acm.org/doi/pdf/10.1145/3474381.

Rishi Sharma, James Allen, Omid Bakhshandeh, and Nasrin Mostafazadeh. Tackling the story ending biases in the story cloze test. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018. URL https://aclanthology.org/P18-2119.pdf.

Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In *International Joint Conference on Neural Networks (IJCNN)*, 2011. URL https://ieeexplore.ieee.org/document/6033395.

Yi-Lin Sung, Varun Nair, and Colin A Raffel. Training neural networks with fixed sparse masks. *Advances in Neural Information Processing Systems*, 34:24193–24205, 2021.

Yi-Lin Sung, Jaehong Yoon, and Mohit Bansal. Ecoflap: Efficient coarse-to-fine layer-wise pruning for vision-language models. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/pdf/2310.02998.

Oyvind Tafjord, Matt Gardner, Kevin Lin, and Peter Clark. Quartz: An open-domain dataset of qualitative relationship questions. In *Proceedings of the 2019 conference on empirical methods in natural language processing (EMNLP)*, 2019. URL https://arxiv.org/abs/1909.03553.

Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL https://arxiv.org/abs/2006.05467.

Anke Tang, Li Shen, Yong Luo, Yibing Zhan, Han Hu, Bo Du, Yixin Chen, and Dacheng Tao. Parameter efficient multi-task model fusion with partial linearization. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2310.04742.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Ke Wang, Nikolaos Dimitriadis, Guillermo Ortiz-Jimenez, François Fleuret, and Pascal Frossard. Localizing task information for improved model merging and compression. In *ICML*, 2024.

Yite Wang, Dawei Li, and Ruoyu Sun. Ntk-sap: Improving neural network pruning by aligning training dynamics. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2304.02840.

Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. *Advances in Neural Information Processing Systems*, 33:15173–15184, 2020.

Jianxiong Xiao, Krista A Ehinger, James Hays, Antonio Torralba, and Aude Oliva. Sun database: Exploring a large collection of scene categories. *International Journal of Computer Vision*, 119:3–22, 2016. URL https://link.springer.com/article/10.1007/s11263-014-0748-y.

Runxin Xu, Fuli Luo, Zhiyuan Zhang, Chuanqi Tan, Baobao Chang, Songfang Huang, and Fei Huang. Raise a child in large language model: Towards effective and generalizable fine-tuning. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 9514–9528, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.749. URL https://aclanthology.org/2021.emnlp-main.749.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models. In *NeurIPS*, 2023.

Enneng Yang, Zhenyi Wang, Li Shen, Shiwei Liu, Guibing Guo, Xingwei Wang, and Dacheng Tao. Adamerging: Adaptive model merging for multi-task learning. In *ICLR*, 2024.

Yi Yang, Wen-tau Yih, and Christopher Meek. Wikiqa: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 conference on empirical methods in natural language processing (EMNLP)*, 2015. URL https://aclanthology.org/D15-1237.pdf.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2311.03099.

Yuan Zhang, Jason Baldridge, and Luheng He. Paws: Paraphrase adversaries from word scrambling. In *2024 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019. URL https://arxiv.org/abs/1904.01130.

# A Appendix

## A.1 Implementation Details

**Computational resources.** We execute all the vision experiments using ViT-B/32, ViT-B/16, and ViT-L/14 on a machine equipped with two NVIDIA GeForce RTX 2080 Ti (11 GB VRAM), an Intel Core i7-9800X CPU @ 3.80GHz and 64 GB of RAM. For all the language experiments using T5-Small, T5-Base, and T5-Large we employ a machine equipped with a a single NVIDIA A100 SXM (64 GB VRAM), an Intel Xeon Platinum 8358 CPU @ 2.60GHz and 64 GB of RAM.

**Starter code.** We developed our codebase starting from the repositories provided by Ortiz-Jimenez et al. (2023)[1] (based on the code by Ilharco et al. (2022; 2023)[2]) and Yadav et al. (2023)[3], which allow to reproduce the full fine-tuning results (Non-linear FT and Linearized FT). TIES-Merging (Yadav et al., 2023)[3], TALL Mask / Consensus (Wang et al., 2024)[4], DARE (Yu et al., 2024)[5], Breadcrumbs (Davari & Belilovsky, 2024)[6] and LoTA (Panda et al., 2024)[7] provide official implementations of their methods from which we carefully adapted their code to work within the Task Arithmetic framework. L-LoRA (Tang et al., 2024) unfortunately doesn't provide any official implementation, but the guidelines in the paper are sufficient to reproduce their results. To this end, we used the `peft` library (Mangrulkar et al., 2022)[8] for implementing the LoRA modules.

**Hyperparameter selection.** As highlighted by Ortiz-Jimenez et al. (2023), task vectors that perform well in Task Negation tend to exhibit higher degrees of weight disentanglement in Task Addition. This relationship informed our hyperparameter selection strategy. For each method, we cross-validate its hyperparameters on each individual task by leveraging Task Negation performance on a small held-out portion of the training set, as implemented by Ilharco et al. (2023); Ortiz-Jimenez et al. (2023). It's important to note that hyperparameter selection shall not be performed separately for addition and negation, as each choice of hyperparameters yields a unique task vector. Hyperparameter search of each method is carried out according to the guidelines presented in each paper. Specifically, for **post-hoc** methods, the sparsity ratio is searched in the set $\{0.1, 0.2, ..., 0.9, 0.95, 0.99\}$. Furthermore, for TALL Mask / Consensus (Wang et al., 2024) we also tune the *consensus threshold* in the set $\{0, ..., T\}$, where $T$ is the number of tasks. For Breadcrumbs (Davari & Belilovsky, 2024) we also tune the percentage of top-$k$ parameters considered outliers, using values from the set $\{0.8, 0.9, 0.95, 0.99, 0.992, 0.994, ..., 0.999\}$. Regarding **parameter-efficient fine-tuning** methods, when using L-LoRA (Tang et al., 2024) we progressively reduce its rank $r \in \{512, 256, 128, 64, 32, 16, 8\}$. While, for LoTA (Panda et al., 2024) and our method we tune sparsity at the task level using values in the set $\{0.1, 0.2, ..., 0.9, 0.95, 0.99\}$. Regarding the amount of data used to perform mask calibration on each task, we align with Panda et al. (2024) by using the validation split as it accounts for the 10% of the total training data. For LoTA, we set the number of iterations for mask calibration so to match the number of mask calibration rounds used by our method (further details at Section A.2). This ensures that the drop in performance is negligible with respect to using the full training split while significantly reducing the computational overhead.

**Datasets & Tasks.** In line with what introduced in Ilharco et al. (2022; 2023); Ortiz-Jimenez et al. (2023), our vision experiments consider image classification across various domains. We adhere to the proposed experimental setup by utilizing eight datasets: Cars (Krause et al., 2013), DTD (Cimpoi et al., 2014), EuroSAT (Helber et al., 2019), GTSRB (Stallkamp et al., 2011), MNIST (LeCun, 1998), RESISC45 (Cheng et al., 2017), SUN397 (Xiao et al., 2016) and SVHN (Netzer et al., 2011).

For the natural language processing (NLP) experiments, we follow the methodology outlined in Yadav et al. (2023), incorporating seven prescribed datasets: three regarding question answering (QASC (Khot et al., 2020), WikiQA (Yang et al., 2015) and QuaRTz (Tafjord et al., 2019)), one for paraphrase identification (PAWS (Zhang et al., 2019)), one focusing on sentence completion (Story Cloze (Sharma et al., 2018)) and two for coreference resolution (Winogrande (Sakaguchi et al., 2021)

[1] https://github.com/gortizji/tangent_task_arithmetic
[2] https://github.com/mlfoundations/task_vectors
[3] https://github.com/prateeky2806/ties-merging
[4] https://github.com/nik-dim/tall_masks
[5] https://github.com/yule-BUAA/MergeLM
[6] https://github.com/rezazzr/breadcrumbs
[7] https://github.com/kiddyboots216/lottery-ticket-adaptation
[8] https://github.com/huggingface/peft

and WSC (Levesque et al., 2012)). Concerning Task Negation, we align with Ortiz-Jimenez et al. (2023) and consider ImageNet (Deng et al., 2009) as the control dataset for vision experiments, while for NLP, we utilize RTE (Dagan et al., 2005), as it provides a distinct task (*i.e*. natural language inference) with respect to the others considered for the NLP experiments.

**Architectures & Pre-trained models.** By following Ilharco et al. (2023); Ortiz-Jimenez et al. (2023); Yadav et al. (2023), on vision experiments, we use three variants of CLIP (Radford et al., 2021) with ViT-B/32, ViT-B/16, and ViT-L/14 models (Dosovitskiy et al., 2021). Regarding the NLP experiments, we employ T5-Small, T5-Base, and T5-Large models (Raffel et al., 2020).

**Fine-tuning details.** All fine-tuning experiments on vision adhere to the training protocol outlined by Ilharco et al. (2022; 2023); Ortiz-Jimenez et al. (2023), with minor modifications made to the training code to accommodate the additional baselines and our method. Specifically, we fine-tune all datasets starting from the same CLIP pre-trained checkpoint, which is obtained from the `open_clip` repository (Gadre et al., 2024). Each model is fine-tuned for 2,000 iterations with a batch size of 128, a learning rate of $10^{-5}$, and a cosine annealing learning rate schedule with 200 warm-up steps. We use the AdamW optimizer (Loshchilov & Hutter, 2019). Following Ilharco et al. (2022), the weights of the classification layer, which are derived from encoding a standard set of zero-shot template prompts for each dataset, are frozen during fine-tuning. Freezing this layer ensures no additional learnable parameters are introduced and does not negatively affect accuracy (Ilharco et al., 2022). Regarding the language experiments, we aligned with Yadav et al. (2023); Ilharco et al. (2023) and utilized three variants of the T5 model (Raffel et al., 2020), namely T5-Small, T5-Base, and T5-Large, with training conducted for a maximum of 75,000 steps. We employed an effective training batch size of 1024, with a learning rate of $10^{-4}$. To prevent overfitting, we implemented an early stopping mechanism with a patience threshold of 5. During training, we used `bfloat16` to reduce GPU memory consumption, and the maximum sequence length was set to 128. Evaluation is carried out by performing rank classification, where the model's log probabilities for all possible label strings are ranked. The prediction is considered correct if the highest-ranked label corresponds to the correct answer. This evaluation method is applicable for both classification and multiple-choice tasks.

**Disentanglement error heatmaps.** As prescribed by Ortiz-Jimenez et al. (2023), we produce the weight disentanglement visualizations of Figure 2 by computing the value of the disentanglement error $\xi(\alpha_1, \alpha_2)$ on a $20 \times 20$ grid of equispaced values in $[-3, 3] \times [-3, 3]$. Estimations are carried out on a random subset of 2,048 test points for each dataset.

**Tuning of $\alpha$ in Task Arithmetic experiments.** As outlined in Ilharco et al. (2023); Ortiz-Jimenez et al. (2023), we employ a single coefficient, denoted as $\alpha$, to adjust the size of the task vectors used to modify the pre-trained models (*i.e*. $\alpha_1 = \alpha_2 = ...\alpha_t$). For both the task addition and task negation benchmarks, following fine-tuning, we evaluate different scaling coefficients from the set $\alpha \in \{0.0, 0.05, 0.1, ..., 1.0\}$ and select the value that achieves the highest target metric on a small held-out portion of the training set, as specified in Ilharco et al. (2023); Ortiz-Jimenez et al. (2023). Specifically, we aim to maximize the *normalized average accuracy* for Task Addition and ensure the minimum *target accuracy* for Task Negation while maintaining at least 95% of the original accuracy of the pre-trained model on the control task. The tuning of $\alpha$ is performed independently for each method.

**Measuring computational costs and memory footprint.** The timings in Table 3 are obtained using the `perf_counter` clock from Python's `time` module. We monitored memory footprint using the NVIDIA `nvml` library [9]. All measurements are obtained during fine-tuning, with the very same setup explained in the fine-tuning details. Then, for each method, the mean and standard deviation of the timings are computed over all iterations of all tasks. Peak memory usage, instead, is taken as the maximum over all tasks. Memory usage is recorded at regular intervals of 1 second, starting from the first forward pass and ending when the training loop breaks.

**Normalized accuracy calculation in Task Addition.** *Normalized accuracy* is computed by taking the average of the normalized individual accuracies over the $T$ tasks. Given a task $t$, the normalized individual accuracy for $t$ is computed by taking the accuracy of the multi-task fused model on $t$ and dividing it by the single-task accuracy that the fine-tuned checkpoint obtained on $t$ before being fused. Formally,

$$\text{Normalized Accuracy} = \frac{1}{T}\sum_{t=1}^{T}\frac{\text{Accuracy}[f(\mathcal{D}_t, \boldsymbol{\theta}_0 + \sum_{t'}^{T}\alpha_{t'}\boldsymbol{\tau}_{t'})]}{\text{Accuracy}[f(\mathcal{D}_t, \boldsymbol{\theta}_0 + \alpha_t\boldsymbol{\tau}_t)]} \tag{11}$$

[9] `https://docs.nvidia.com/deploy/nvml-api/`

**Algorithm 1:** `TaLoS` to obtain task vectors

**Input :** Pre-trained model $\boldsymbol{\theta}_0 \in \mathbb{R}^m$, task dataset $\mathcal{D}_t$, final sparsity $k$, number of rounds $R$, number of epochs $E$, learning rate $\gamma$, loss function $\mathcal{L}$

**Output :** Task vector $\boldsymbol{\tau}_t \in \mathbb{R}^m$ for performing task arithmetic

*// Calibrate sparse fine-tuning mask*
$\boldsymbol{c} \leftarrow \mathbb{1}$ ▷ Initialize weight mask to all 1s
**for** $r = 1, 2, ..., R$ **do**
  $p \leftarrow k^{(r/R)}$ ▷ Compute the current sparsity at round $r$
  $\boldsymbol{s} \leftarrow \mathbf{0}$ ▷ Initialize parameter-wise scores to all 0s
  **for** $\boldsymbol{x} \in \mathcal{D}_t$ **do**
    $\boldsymbol{s} \leftarrow \boldsymbol{s} + |\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{c} \odot \boldsymbol{\theta}_0)|$ ▷ Compute and update scores on current example
  **end**
  *// Update $\boldsymbol{c}$ to retain only the bottom-$k$ parameters*
  $\hat{\boldsymbol{s}} \leftarrow$ `sort_descending(`$\boldsymbol{s}$`)` ▷ Sorted scores in descending order
  $p \leftarrow \lfloor p \cdot m \rfloor$ ▷ compute bottom-$p$ threshold index
  **for** $j = 1, 2, ..., m$ **do**
    **if** $\boldsymbol{s}_j - \hat{\boldsymbol{s}}_p > 0$ **then**
      $\boldsymbol{c}_j \leftarrow 0$ ▷ Set the mask of the $j$-th parameter to zero
    **end**
  **end**
**end**
*// Sparse fine-tuning, starting from $\boldsymbol{\theta}_0$ and obtaining $\boldsymbol{\theta}_t$*
**for** *epoch* $= 1, 2, ..., E$ **do**
  **for** $(\boldsymbol{x}, y) \in \mathcal{D}_t$ **do**
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \left[\boldsymbol{c} \odot \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\boldsymbol{x}, \boldsymbol{\theta}), y)\right]$ ▷ Update rule, mask gradients with $\boldsymbol{c}$
  **end**
**end**
$\boldsymbol{\tau}_t \leftarrow \boldsymbol{\theta}_t - \boldsymbol{\theta}_0$ ▷ Compute final task vector for task $t$
**return** $\boldsymbol{\tau}_t$

| Method | Avgerage Execution Time (s) | | | Peak Memory Usage (GiB) | | | Task Addition | | Task Negation | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Mask | Train | Total | Mask | Train | Overall | Abs. (↑) | Norm. (↑) | Targ. (↓) | Cont. (↑) |
| Non-linear FT (Ilharco et al., 2023) | - | 2479.99 | 2479.99 | - | 18.6 | 18.6 | 86.09 | 90.14 | 20.61 | 72.72 |
| Linearized FT (Ortiz-Jimenez et al., 2023) | - | 3311.77 | 3311.77 | - | 21.3 | 21.3 | <u>88.29</u> | <u>93.01</u> | <u>10.86</u> | 72.43 |
| L-LoRA (Tang et al., 2024) | - | <u>1053.07</u> | <u>1053.07</u> | - | <u>9.7</u> | <u>9.7</u> | 87.77 | 91.87 | 19.39 | 73.14 |
| LoTA (Panda et al., 2024) | **51.84** | 2592.40 | 2644.24 | 12.9 | 15.4 | 15.4 | 87.60 | 91.89 | 22.02 | <u>73.22</u> |
| **`TaLoS` (Ours)** | 63.04 | **656.23** | **719.27** | **7.8** | **7.8** | **7.8** | **88.40** | **95.19** | **10.63** | **73.55** |

Table 4: **Computational cost and memory footprint of mask calibration and fine-tuning.** Average time (in seconds) and peak memory usage (in Gibibytes) of mask calibration and fine-tuning approaches on CLIP ViT-L/14, alongside their performance on the task arithmetic benchmark. For both LoTA and `TaLoS`, we used batch size 128 for 40 iterations (in detail, 10 iterations per round for `TaLoS`, with 4 rounds total). We employ gradient checkpointing during mask calibration. Further details on the resource monitoring process can be found in Appendix A.1. **Bold** indicates the best results. <u>Underline</u> the second best.

### A.2 Details on Mask Calibration & Computational Overhead

Sparse fine-tuning prescribes to mask gradients when updating the model parameters. Thus, is is foundational that the mask is correctly calibrated before training. We mask only Linear, Attention, LayerNorm, and Convolutional layers (Kwon et al., 2022). Embedding layers and final projection layers are kept frozen. Furthermore, following standard procedures in Pruning-at-Initialization (PaI) (Tanaka et al., 2020; Wang et al., 2023), we iteratively refine the mask in multiple rounds to obtain better estimates from the mask calibration procedures. In detail, at each round, we select the bottom-$p$ parameters (according to our score, detailed in Section 4) and we exponentially increase the current sparsity $p$. We repeat this process until we reach the target sparsity $k$. For the sake of major clarity, we report in Algorithm 1 the pseudocode for our procedure, encompassing both mask calibration and sparse fine-tuning. We remark that the choice of the bottom-$k$ values may lead to *layer collapse* (Tanaka et al., 2020), namely, removing all parameters in a layer, disrupting the information flow in

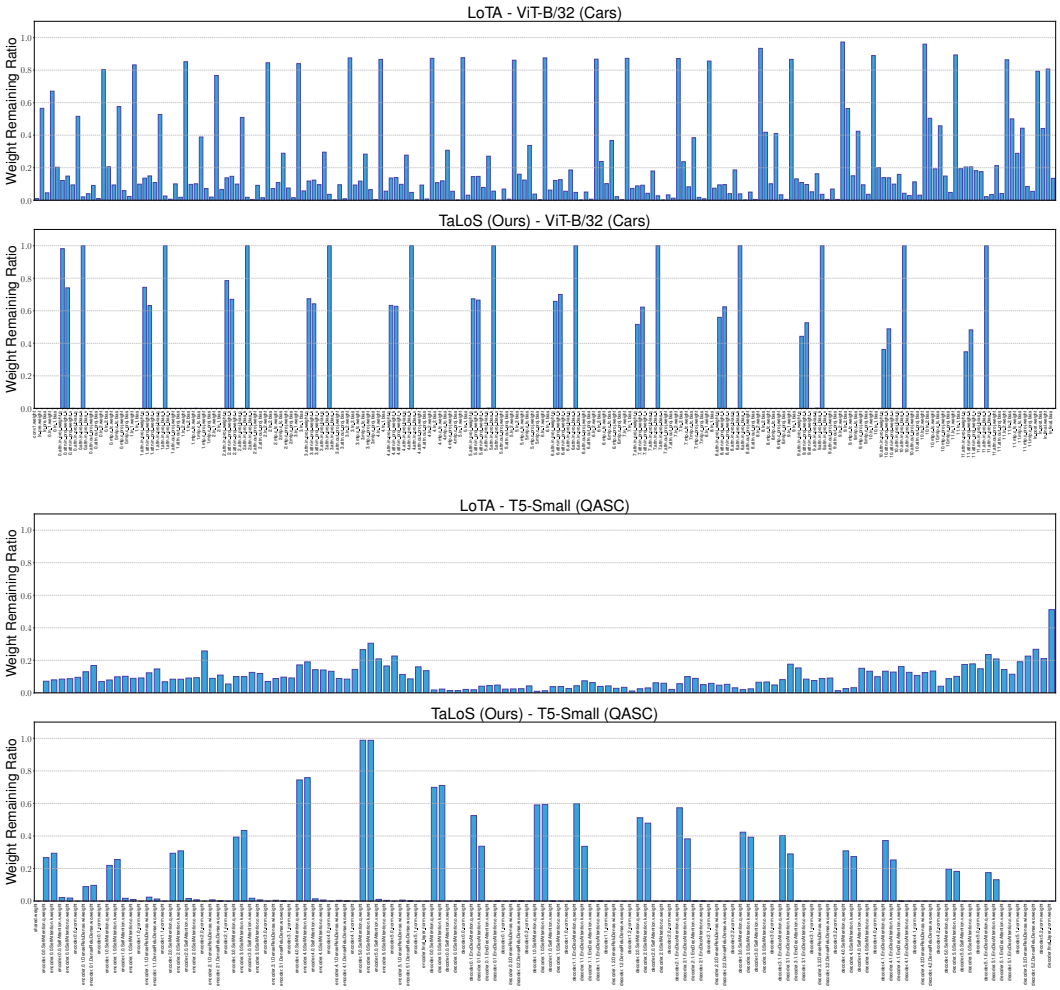


Figure 5: **Visualization of mask calibration.** Percentage of parameters selected for sparse fine-tuning in a ViT-B/32 (top) and a T5-Small (bottom) models, after our method's mask calibration vs. LoTA's mask calibration, at 90% sparsity. On ViT-B/32, we calibrate the masks on the Cars dataset (Krause et al., 2013), while on T5-Small we use QASC (Khot et al., 2020).

the network. To face this problem, we set $\boldsymbol{c}$ to some positive value close to zero (*e.g.* 0.01) and we don't include in the ranking those entries that are already soft-masked. This ensures that we are not changing the nature of our estimation, while countering the possibility of disrupting gradient flow in the network, during calibration.

Unfortunately, mask calibration introduces some amount of overhead before training. It is of paramount importance that such overhead doesn't hinder the computational gains obtained during fine-tuning.

**Time overhead.** The time spent for a single iteration of mask calibration is comparable to that of a single forward-backward iteration of non-linear fine-tuning (refer to Table 3). Our mask calibration process typically employs an average of 10 iterations per round, with satisfactory results observed at just 4 rounds (*i.e.*, approximately 40 iterations total, we use the same batch size for mask calibration as for fine-tuning). Given that fine-tuning generally requires around 2,000 iterations for vision experiments and substantially more for language tasks, we argue that the time overhead introduced by our mask calibration is negligible.

**Memory overhead.** The memory cost of each mask calibration iteration is equivalent to that of each training iteration in non-linear fine-tuning. While we have not implemented any specific mechanism to reduce the memory footprint for calculating gradients (used as scores) during mask calibration, there are several approaches available to achieve this. Most of these methods involve estimating gradients

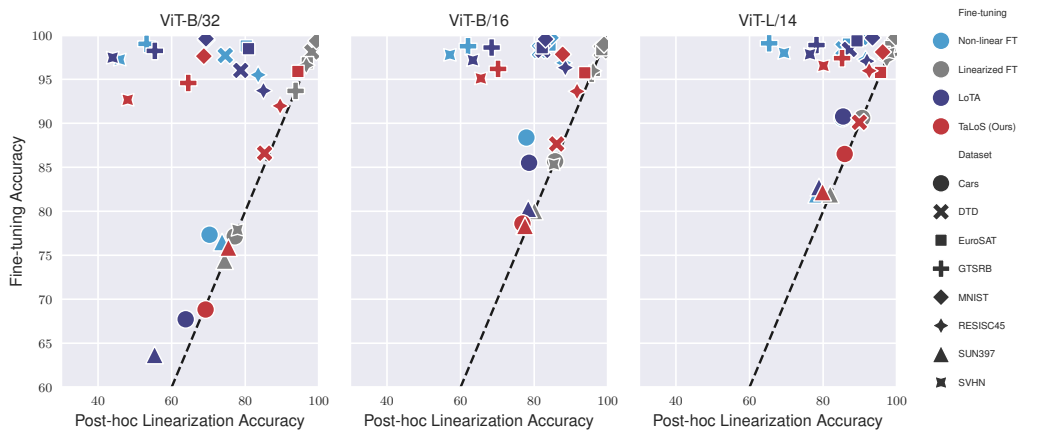


Figure 6: **Testing linearized behavior.** Single-task accuracies of different fine-tuning strategies, each used to obtain their corresponding task vectors $\boldsymbol{\tau}_t$, and the accuracy of their post-hoc linearization $f_{\text{lin}}(\cdot, \boldsymbol{\theta}_0 + \boldsymbol{\tau}_t)$. Different colors represent distinct fine-tuning strategies, while different markers indicate different tasks. Points that lie on the bisector (black dashed line) indicate that the fine-tuning process exhibited linearized behavior.

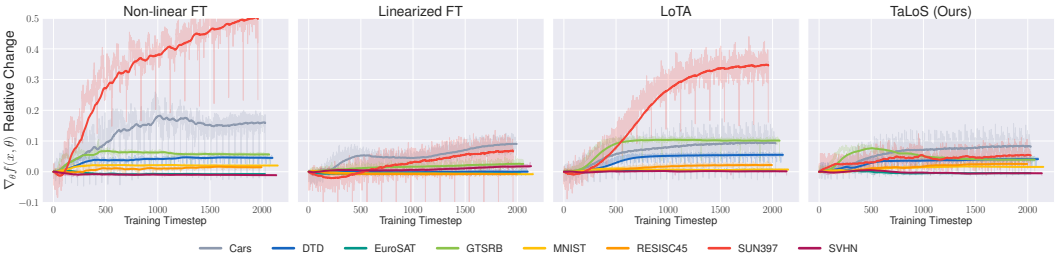


Figure 7: **Change in parameter sensitivity throughout fine-tuning.** We visualize the average relative change in the output derivative of the parameters of a CLIP ViT-B/32 model when fine-tuned using different approaches. The starting point is the same for all methods.

using zeroth-order information (Hinton, 2022; Malladi et al., 2023a; Sung et al., 2024), which allows to trade off speed for reduced memory usage by approximating gradients through multiple forward passes, eliminating the need to store computational graphs for automatic differentiation. Alternatively, gradient checkpointing (Chen et al., 2016) is another practical solution.

To further clarify the overall computational cost of TaLoS, encompassing both mask calibration and sparse fine-tuning, we provide a comparison in Table 4 of the timings in seconds (averaged over the 8 vision tasks) and the peak memory usage in Gibibytes of mask calibration and fine-tuning on a CLIP ViT-L/14. The results show that mask calibration time is approximately the same for TaLoS and LoTA, however, the costs in terms of memory are very different (LoTA requires storing optimizer states). Regarding total time, we recover what was presented in Table 3, highlighting the beneficial effect of the highly structured sparsity of TaLoS on fine-tuning. The task arithmetic results are in line with Tables 1, 2, with no detrimental effect given by the usage of gradient checkpointing.

### A.3 Full Mask Calibration Visualizations

For the sake of completeness, we provide a full visualization in Figure 5 of the masks obtained after calibration with TaLoS and LoTA. As shown, a repeating sparsity pattern emerges for our method across each transformer block. Notably, TaLoS consistently identifies only the **Q** and **K** parameters for fine-tuning, demonstrating a more structured behavior. In contrast, the mask generated by LoTA appears far more unstructured, with no clear pattern across the blocks.

### A.4 Analyzing the Fine-tuning Behavior

We provide an empirical validation on the linear fine-tuning regime of our TaLoS (*i.e.* the change in the network output can be well-approximated by its first-order Taylor expansion around $\boldsymbol{\theta}_0$). As

discussed by Ortiz-Jimenez et al. (2023), a cheap test consists of performing post-hoc linearization of the fine-tuned model around $\boldsymbol{\theta}_0$ and checking whether the performance produced by such a linearized model matches that of the original fine-tuned model. We use this approach and report the results in Figure 6. The scatter plots compare the fine-tuning accuracy against the post-hoc linearization accuracy for various tasks and fine-tuning strategies across different ViT architectures. Our method, TaLoS, consistently demonstrates linearized behavior during fine-tuning for most tasks, as evidenced by its proximity to the bisector line. This supports our claim that sparse fine-tuning, which both TaLoS and LoTA employ, inherently promotes the emergence of linearized behavior during fine-tuning. Interestingly, while TaLoS exhibits this property across a wide range of tasks, LoTA does not consistently demonstrate the same level of linearized behavior. This discrepancy can be attributed to differences in parameter selection, as discussed in the next paragraph, closely matching what happens during linearized fine-tuning. It's worth noting that linearized behavior may arise for various fine-tuning strategies, but its occurrence depends on the interaction between the task and pre-training (Malladi et al., 2023b). For instance, tasks such as GTSRB (Stallkamp et al., 2011), MNIST (LeCun, 1998), and SVHN (Netzer et al., 2011) do not exhibit fine-tuning in the linear regime, hinting at a potential mismatch with the pre-training, as evidence suggests (Radford et al., 2021).

To further test the fine-tuning regime, we examine the evolution of parameter sensitivity during fine-tuning across different methods, as depicted in Figure 7. Notably, for TaLoS, the gradient $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta})$ remains almost unchanged throughout training, closely mirroring the behavior of linearized fine-tuning. In contrast, LoTA diverges from this pattern, behaving more in line with non-linear fine-tuning. This phenomenon reinforces our claim that our method fine-tunes in the linearized regime, as maintaining a constant $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta})$ during fine-tuning is critical for operating in the linearized regime (Malladi et al., 2023b).

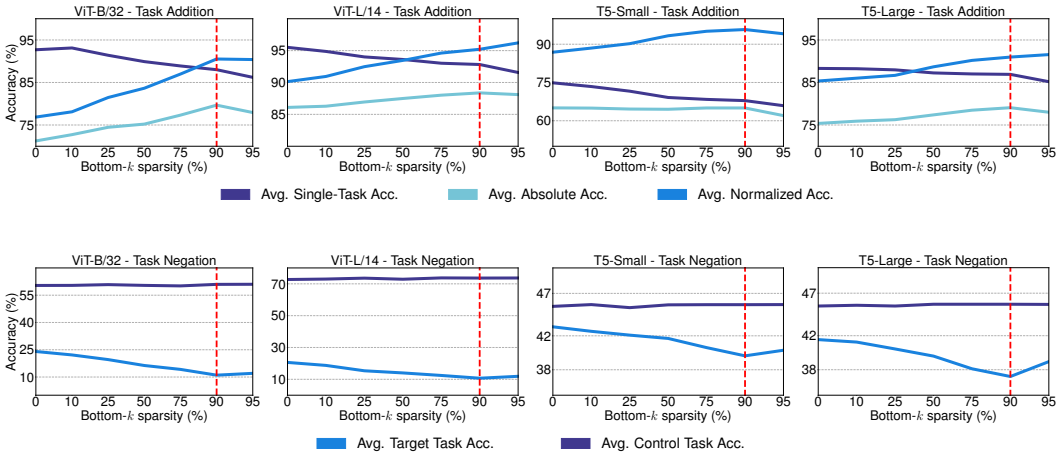


Figure 8: **Effect of the choice of $k$ in TaLoS.** Results of hyperparameter tuning of $k$ in TaLoS for task addition and negation on both vision and language. Note that we tune $k$ indirectly by controlling its value via the sparsity ratio. For **task addition** (top) we report the average single-task accuracy (before addition), absolute and normalized accuracies (after addition). For **task negation** (bottom) we report average target and control accuracies (after negation).

### A.5 Ablations on Mask Sparsity Ratio

For a clear understanding of the effect of sparsity on TaLoS, we report in Figure 8 the task arithmetic performance achieved by TaLoS, while varying the sparsity level. At 0% sparsity, we recover full (non-linear) fine-tuning results. Increasing the sparsity improves the task arithmetic performance, while slightly decreasing the average single-task accuracy, as fewer parameters are updated during fine-tuning. Optimal values for absolute accuracy (in task addition) and target accuracy (in task negation) are observed for a sparsity level of 90% across a variety of models. After 90% sparsity there is a slight drop in both task arithmetic and single-task performance, making such sparsity levels not ideal. Intuitively, if the fine-tuning involves too little weight the resulting entries in the task vector will be mostly zero, reducing the ability to perform task arithmetic effectively. We can conclude that, like other parameter-efficient fine-tuning methods, our approach trades some single-task performance

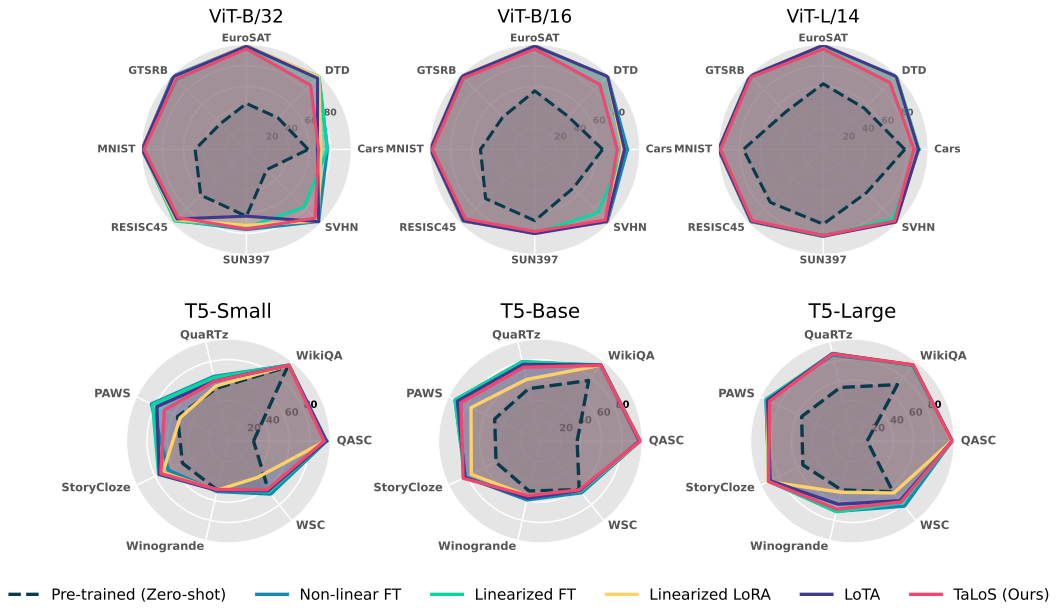


Figure 9: **Task performance after fine-tuning.** Single-task accuracies obtained by different fine-tuning approaches across vision and language experiments. Results are displayed for three model sizes of CLIP ViT (B/32, B/16, L/14) and T5 (Small, Base, Large), with outer edges representing higher accuracy. The dashed line represents the accuracies before fine-tuning.

for parameter efficiency. But this trade-off allows also for superior task arithmetic capabilities for TaLoS (Tables 1, 2) while maintaining competitive single-task accuracy, especially for larger models where the performance drop becomes negligible (Figure 9).

### A.6 Single-task Performance of Fine-tuning Methods

In this analysis we focus on discussing the single-task performance of TaLoS before task addition. To this goal, we compare in Figure 9 the accuracies obtained by TaLoS (at 90% sparsity) vs. the other fine-tuning strategies. In almost all cases TaLoS achieves approximately the same performance of full fine-tuning methods (Non-linear FT and Linearized FT), occasionally improving over Linearized FT (ViT-B/32 on SVHN), which is remarkable, as TaLoS updates only a very small subset of parameters, while full fine-tuning (both linearized and non-linear) updates the whole set of model parameters. Furthermore, compared with parameter-efficient fine-tuning methods, which allows for a truly fair comparison (the parameter count is the same across methods), almost always TaLoS improves with respect to Linearized LoRA and matches the performance of LoTA. However, we remark that the task arithmetic performance of TaLoS is much higher than the latter (see Tables 1, 2).

### A.7 Sensitivity Analysis of Parameters and Connection to Fisher Information

Applying a perturbation $\boldsymbol{\theta}_0' \leftarrow \boldsymbol{\theta}_0 + \delta\boldsymbol{\theta}_0$ to a subset of the pre-trained weights $\boldsymbol{\theta}_0$ and observing no change in the output $f(\boldsymbol{x}, \boldsymbol{\theta}_0') \approx f(\boldsymbol{x}, \boldsymbol{\theta}_0)$ intuitively means that those weights have low sensitivity to the task. So, pruning or randomizing them would not affect input-output behavior.

However, there may be a problem in assessing sensitivity via extreme randomizations/perturbations: if "extreme" randomization refers to very high magnitude perturbations (perhaps, additive), then such perturbations will not be suitable to assess the sensitivity of the parameters, as this could potentially move the current solution (parametrized by $\boldsymbol{\theta}_0 \in \mathbb{R}^m$) away from the current local optimum, to a distinct region of the loss landscape. Indeed, sensitivity analysis generally refers to "robustness to small perturbation". This concept, alongside how to perform proper sensitivity analysis on the parameters of a neural network has been formalized by a rich literature dedicated to applications of information geometry (Amari, 1996; Chaudhry et al., 2018; Pascanu & Bengio, 2013). Specifically, as shown by Chaudhry et al. (2018); Pascanu & Bengio (2013), to assess the influence of each weight on the output of a network, we can use the Kullback-Leibler (KL) divergence between the output

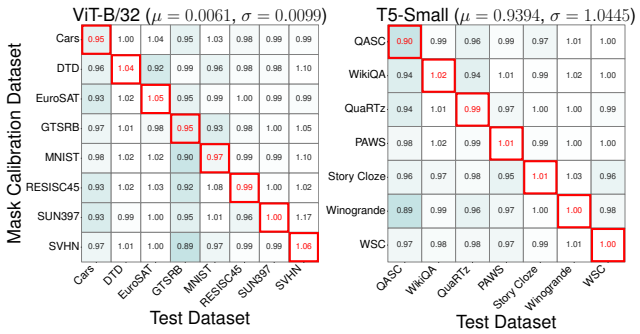


Figure 10: **Perturbing parameters with low sensitivity.** The heatmaps illustrate the effect of perturbing the parameters with the lowest sensitivity (measured by $\mathbb{E}_{\boldsymbol{x}\in\mathcal{D}_t}[|\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}_0)|]$) on different tasks across various pre-trained models. Each grid compares the accuracy ratios for models after pruning, with the rows representing the task $\mathcal{D}_t$ used to identify the parameters with the lowest sensitivity and the columns showing the model's performance on each task after pruning those parameters. The accuracy ratios are normalized by the model's performance before perturbation. The average magnitude $\mu$ and standard deviation $\sigma$ across perturbed parameters, prior to applying noise are also reported. The ratio of perturbed parameters (10%) is chosen based on the experiment of Figure 1.

distribution induced by the original network ($p_{\boldsymbol{\theta}_0}$) vs. the one induced by the perturbed network ($p_{\boldsymbol{\theta}_0+\delta\boldsymbol{\theta}}$). Mathematically, assuming $\delta\boldsymbol{\theta} \to 0$ (a small perturbation),

$$D_{\mathrm{KL}}(p_{\boldsymbol{\theta}_0} \| p_{\boldsymbol{\theta}_0+\delta\boldsymbol{\theta}}) = \frac{1}{2}\delta\boldsymbol{\theta}^\top F(\boldsymbol{\theta}_0)\delta\boldsymbol{\theta} + \mathcal{O}(\|\delta\boldsymbol{\theta}\|^3)$$

The KL divergence is zero if the perturbation doesn't affect the output, revealing that the modified weights are not influential for the output. It is $> 0$ otherwise. Here $F(\boldsymbol{\theta}_0) \in \mathbb{R}^{m\times m}$ is the Fisher Information matrix (FIM) (Fisher, 1922; Amari, 1996). It is a positive semi-definite symmetric matrix defined as,

$$F(\boldsymbol{\theta}_0) = \mathbb{E}_{\boldsymbol{x}}[\mathbb{E}_{y\sim p_{\boldsymbol{\theta}_0}(y|\boldsymbol{x})}[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}_0}(y|\boldsymbol{x}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}_0}(y|\boldsymbol{x})^\top]]$$

It can be used to relate the changes in the parameters to the changes in the outputs, effectively implementing a proper sensitivity analysis of the parameters of a neural network by studying the magnitude of its diagonal elements, as they represent the sensitivity of each parameter (Chaudhry et al., 2018; Pascanu & Bengio, 2013; Matena & Raffel, 2022). Formally, for each parameter $j$, with $j \in 1, ..., m$ its corresponding entry on the diagonal of the FIM has value

$$F_{jj}(\boldsymbol{\theta}_0) = \mathbb{E}_{\boldsymbol{x}}[\mathbb{E}_{y\sim p_{\boldsymbol{\theta}_0}(y|\boldsymbol{x})}[\nabla_{\boldsymbol{\theta}_j} \log p_{\boldsymbol{\theta}_0}(y|\boldsymbol{x})]^2]$$

The higher this value is, the more the $j$-th parameter will be sensitive (*i.e.* will influence the output of the model). Note that in our work we rank parameter sensitivity using $|\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}_0)|$, which is equivalent to using information from the diagonal elements of the FIM. Modeling $p_{\boldsymbol{\theta}_0}$ as a categorical distribution induced by the outputs of $f$ (which is a proper assumption for multi-class classification tasks), the logarithm function ($y = \log(x)$), the squaring function ($y = x^2$) and the absolute value function ($y = |x|$) are monotonically increasing in the open interval $]0, +\infty[$. Hence, the sensitivity ranking is the same as $[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}_0}(y|\boldsymbol{x})]^2$.

We repeat in Figure 10 the experiment of Figure 1, but by adding noise distributed as $\mathcal{N}(0, 2\sigma I)$ to the bottom-10% of parameters, instead of pruning them. $\sigma$ is the standard deviation of the parameters, previous to perturbation. The results align with the analysis reported in Figure 1, highlighting the stability of these parameters across tasks.

Additionally, in Figure 11 we provide further evidence about the overlap of low-sensitivity parameters across tasks. For each parameter, we compute the mean Intersection over Union (mIoU) of masks, between each task pair: starting from pre-trained parameters $\theta_0$, we predict the mask on task $t$ and

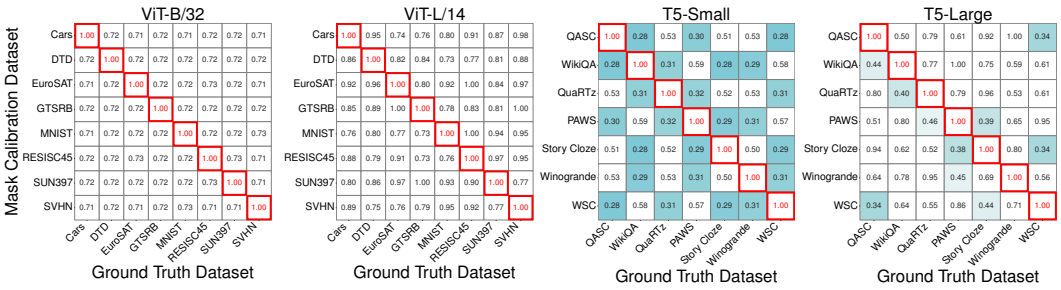


Figure 11: **Masks intersections of low sensitivity parameters.** The heatmaps illustrate the mean Intersection over Union (mIoU) between masks pairs of the lowest sensitivity parameters (measured by $\mathbb{E}_{\boldsymbol{x}\in\mathcal{D}_t}[|\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x},\boldsymbol{\theta}_0)|]$) on all tasks across different pre-trained models. For each mask, the amount of selected parameters (10%) is chosen based on the experiment of Figure 1.

| Method | ViT-B/32 | | T5-Small | |
|---|---|---|---|---|
| | Abs. (↑) | Norm. (↑) | Abs. (↑) | Norm. (↑) |
| Pre-trained (Zero-shot) | 47.72 | - | 55.70 | - |
| Non-linear FT (Ilharco et al., 2023) | 71.25 | 76.94 | 65.04 | 87.98 |
| TIES-Merging (Yadav et al., 2023) | 74.79 | 82.84 | 62.53 | 94.83 |
| Task-wise AdaMerging (Yang et al., 2024) | 73.39 | 79.02 | 66.19 | 89.86 |
| Layer-wise AdaMerging (Yang et al., 2024) | 77.06 | 82.98 | <u>66.61</u> | 89.86 |
| **TaLoS (Ours)** | 79.67 | 90.73 | 65.04 | 97.22 |
| **TaLoS + TIES-Merging** | 78.15 | 89.10 | 54.54 | 85.42 |
| **TaLoS + Task-wise AdaMerging** | <u>79.73</u> | <u>90.84</u> | 66.47 | <u>99.21</u> |
| **TaLoS + Layer-wise AdaMerging** | **80.25** | **91.40** | **66.76** | **99.63** |

Table 5: **TaLoS on different model merging schemes.** Average absolute accuracies (%) and normalized accuracies (%) of CLIP ViT-B/32 and T5-Small pre-trained models edited by adding task vectors on each of the downstream tasks. We normalize performance of each method by their single-task accuracy. **Bold** indicates the best results. <u>Underline</u> the second best.

then check its intersection over union against the mask predicted on task $t'$ (which acts as a ground truth). A mIoU of 1 signals perfect mask overlap between tasks. The number of parameters selected by each mask is 10%, in line with the experiment of Figure 1. Smaller vision models (ViT-B/32) exhibit high parameter sharing ($> 0.7$ mIoU) of low-sensitivity parameters, while smaller language models (T5-Small) share fewer (0.3–0.5 mIoU). However, with a fixed 10% mask sparsity, larger models in both vision and language domains share more low-sensitivity parameters across tasks.

### A.8 Combining TaLoS With Other Model Merging Schemes

We extend Table 1 in Table 5 by testing our TaLoS in combination with other merging schemes (TIES-Merging Yadav et al. (2023) and AdaMerging Yang et al. (2024)). Specifically, for TIES-Merging we skip the sparsification part, as the task vectors obtained by TaLoS are already sparse. Regarding AdaMerging, we test both Task-wise AdaMerging and Layer-wise AdaMerging. As we can see, in both vision and language experiments, applying TIES-Merging to our TaLoS is harmful. Seemingly, the signs of task vectors obtained via TaLoS play an important role and disrupting them according to some heuristics causes a drop in performance. Regarding AdaMerging, we can see that TaLoS has full compatibility with existing methods for automating the selection of optimal merging coefficients, highlighting its versatility. However, by itself TaLoS is already robust enough that it doesn't benefit this much from neither task-wise tunings nor layer-wise tunings.