A Experimental details

All our experiments were performed using the same hardware consisting of four V100 NVIDIA GPUs with 32GB of memory each and can be reproduced in less than 350 GPU hours. The details of each experiment are the following.

Fine-tuning. All the fine-tuning experiments follow the same training protocol specified in Ilharco et al. [39] with minor modifications to the training code to use linearized models when needed. In particular, we fine-tune all datasets starting from the same CLIP pre-trained checkpoint downloaded from the `open_clip` repository [37]. We fine-tune for $2,000$ iterations with a batch size of 128, learning rate of 10^{-5} and a cosine annealing learning rate schedule with 200 warm-up steps and the AdamW optimizer [49]. As introduced in Ilharco et al. [38], during fine-tuning, we freeze the weights of the classification layer obtained by encoding a standard set of *zero-shot* template prompts for each dataset. Freezing this layer does not harm accuracy and ensures that no additional learnable parameters are introduced during fine-tuning [38]. We use this exact same protocol to fine-tune the non-linear and linearized models and do not perform any form of hyperparameter search in our experiments.

Tuning of α in task arithmetic benchmarks. As in Ilharco et al. [39] we use a single coefficient α to tune the size of the task vectors used to modify the pre-trained models. This is equivalent to setting $\alpha = \alpha_1 = \dots \alpha_T$ in Eq. (1). Both in the task addition and task negation benchmarks, after fine-tuning, we evaluate different scaling coefficients $\alpha \in \{0.0, 0.05, 0.1, \dots, 1.0\}$ and choose the value that achieves the highest target metric on a small held-out proportion of the training set as specified in Ilharco et al. [39]. Namely, maximum normalized average accuracy, and minimum target accuracy on each dataset that still retains at least 95% of the accuracy of the pre-trained model on the control task; for task addition and negation, respectively. The tuning of α is done independently for non-linear FT, linearized FT, and post-hoc linearization.

Normalized accuracies in task addition. Table 1 shows the normalized accuracies after editing different models by adding the sum of the task vectors on 8 tasks $\boldsymbol{\tau} = \sum_t \boldsymbol{\tau}_t$. Here, the normalization is performed with respect to the single-task accuracies achieved by the model fine-tuned on each task. Mathematically,

$$\text{Normalized accuracy} = \frac{1}{T}\sum_{t=1}^{T} \frac{\underset{\boldsymbol{x}\sim\mu_t}{\text{acc}}\left[f(\boldsymbol{x}; \boldsymbol{\theta}_0 + \sum_{t'} \boldsymbol{\tau}_{t'})\right]}{\underset{\boldsymbol{x}\sim\mu_t}{\text{acc}}\left[f(\boldsymbol{x}; \boldsymbol{\theta}_0 + \boldsymbol{\tau}_t)\right]}. \tag{7}$$

Disentanglement error. To produce the weight disentanglement visualizations of Figure 3 we compute the value of $\xi(\alpha_1, \alpha_2)$ on a 20×20 grid of equispaced values in $[-3, 3] \times [-3, 3]$. To estimate the disentanglement error, we use a random subset of $2,048$ test points for each dataset.

NTK eigenfunction estimation. We use the finite-width NTK implementation from the `functorch` sublibrary of PyTorch [66] to compute the K_{NTK} matrices described in Section 6.1. In particular, we use a random subset of 200 training points for each dataset and compute the singular value decomposition (SVD) of K_{NTK} to estimate the entries of ϕ_ρ on each dataset. As described in Bordelon et al. [11], and to avoid a high memory footprint, we estimate a different set of singular vectors for each output class, equivalent to estimating one kernel matrix per output logit. Figure 6 shows the values of $\mathcal{E}_{\text{loc}}(\boldsymbol{x})$ for each class with a different line. However, there is little variability of the NTK among classes, and hence all curves appear superimposed in the figure.

B Implementation aspects of linearized models

We now provide more details of the different implementation aspects of linearized models, including basic code and a discussion on their computational complexity.

Practical implementation. Creating linearized models of a neural network is very simple using the `functorch` sublibrary of PyTorch. Specifically, using the fast Jacobian-vector implementation of this library, we can easily create a custom class that takes any `nn.Module` as input and generates a trainable linearized version of it around its initialization. We give a simple example of this in

Listing 1, where we see that the resulting `LinearizedModel` can be directly used in any training script as any other neural network.

In our experiments, we linearize the ViT image encoder of CLIP as the text encoder is frozen in our experiments. In this regard, during training and inference, as it is common in standard CLIP models [69], we normalize the output of the linearized image encoder prior to performing the inner product with the text embeddings. This normalization does not change the classification decision during inference, but it has a rescaling effect on the loss that can influence training dynamics. In our fine-tuning experiments, we found this standard normalization technique has a clearly positive effect in single-task accuracy both for the non-linear and linearized models.

```python
import copy
import torch.nn as nn
from functorch import jvp, make_functional_with_buffers

class LinearizedModel(nn.Module):
    """ Creates a linearized version of any nn.Module.

    The linearized version of a model is a proper PyTorch model and can be
    trained as any other nn.Module.

    Args:
        init_model (nn.Module): The model to linearize. Its parameters are
        used to initialized the linearized model.
    """
    def __init__(self, init_model):
        # Convert models to functional form.
        func, params0, buffers0 = make_functional_with_buffers(init_model)

        # Store parameters and forward function.
        self.func0 = lambda params, x: func(params, buffers0, x)
        self.params0 = params0                  # Initialization parameters.
        self.params = copy.deepcopy(params0)    # Trainable parameters.

        # Freeze initial parameters and unfreeze current parameters.
        for p0 in self.params0: p0.requires_grad = False
        for p in self.params: p.requires_grad = True

    def __call__(self, x):
        # Compute linearized model output.
        dparams = [p - p0 for p, p0 in zip(self.params, self.params0)]
        out, dp = jvp(self.func0, (self.params0,), (dparams,))
        return out + dp
```

Listing 1: Basic PyTorch code to linearize a model.

Computational complexity. Jacobian-vector products can be computed efficiently, at the same marginal cost as a forward pass, using forward-mode automatic differentiation rules [9]. This means that doing inference with a linearized model usually takes around two or three times more than with its non-linear counterpart, as for every intermediate operation in the forward pass, its derivative also needs to be computed and evaluated.

Training the linearized models, on the other hand, uses the backpropagation algorithm which, for every forward pass, requires another backward pass to compute the gradients. In this regard, the computational cost of obtaining the gradient with respect to the trainable parameters of the linearized models $\nabla_{\boldsymbol{\theta}} f_{\text{lin}}(\boldsymbol{x}; \boldsymbol{\theta})$ is also roughly twice the cost of obtaining the gradient of its non-

linear counterparts $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}; \boldsymbol{\theta})$. Similarly, as the forward-mode differentiation required to compute the forward pass also depends on the values of the derivatives at this step, the final memory footprint of training with the linearized models is also double than the one of training the non-linear ones.

C Spectral analysis of linearized models

In this section, we present the formal statement and proof of Proposition 1. Additionally, we delve deeper into the question of whether eigenfunction localization is a necessary condition for task arithmetic and provide analytical examples with exactly diagonalizable NTKs to support our discussion.

Proposition 2 (Formal version of Proposition 1)**.** *Suppose that the task functions $\{f_t^\star\}_{t\in[T]}$ belong to the RKHS of the kernel k and their coefficients in the kernel eigenbasis are $\{(c_{t,\rho}^\star)_{\rho\in\mathbb{N}}\}_{t\in[T]}$. If $\forall\, t, \rho$, either $c_{t,\rho}^\star = 0$ or $\operatorname{supp}(\phi_\rho) \subseteq \mathcal{D}_t$, then the kernel k has the task arithmetic property with respect to $\{f_t^\star\}_{t\in[T]}$ and $\{\mathcal{D}_t\}_{t\in[T]}$.*

Proof. The task arithmetic property requires that $\forall t' \in [T],\ \forall \boldsymbol{x} \in \mathcal{D}_{t'},\ \sum_{t\in[T]} f_t^\star(\boldsymbol{x}) = f_{t'}^\star(\boldsymbol{x})$. Representing the task functions in the kernel basis, we have

$$\forall t' \in [T],\ \forall \boldsymbol{x} \in \mathcal{D}_{t'},\ \sum_{t\in[T]} \sum_{\rho\in\mathbb{N}} c_{t,\rho}^\star \phi_\rho(\boldsymbol{x}) = \sum_{\rho\in\mathbb{N}} c_{t',\rho}^\star \phi_\rho(\boldsymbol{x}). \tag{8}$$

This condition can be rewritten as

$$\int_{\mathcal{D}_{t'}} \left(\sum_{t\in[T],\, t\neq t'} \sum_{\rho\in\mathbb{N}} c_{t,\rho}^\star \phi_\rho(\boldsymbol{x}) \right)^2 d\boldsymbol{x} = 0. \tag{9}$$

If, for each t, the eigenfunctions corresponding to non-zero coefficients are supported within a subset of $\mathcal{D}_t$ and all domains $\mathcal{D}_t$'s are disjoint, then all the summands inside the integral in Eq. (9) become zero inside $\mathcal{D}_{t'}$, and thus the proof is complete. □

As we discussed in Section 6.1, eigenfunction localization is generally not a necessary condition to achieve task arithmetic. However, we now show that if the eigenfunctions are locally linear independent across the different task domains, then the localization property becomes a necessary condition for task arithmetic. The proposition presented below formalizes this concept.

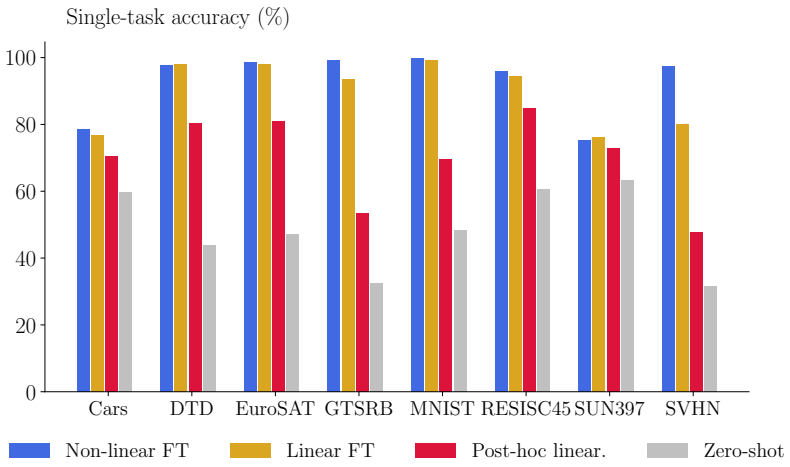
Proposition 3. *Suppose that the task functions $\{f_t^\star\}_{t\in[T]}$ belong to the RKHS of the kernel k and their coefficients in the kernel eigenbasis are $\{(c_{t,\rho}^\star)_{\rho\in\mathbb{N}}\}_{t\in[T]}$. Furthermore, let the kernel eigenfunctions be either zero or linearly independent over each domain $\mathcal{D}_t$. The kernel k has the task arithmetic property with respect to $\{f_t^\star\}_{t\in[T]}$ and $\{\mathcal{D}_t\}_{t\in[T]}$ if and only if $\forall\, t, \rho$, either $c_{t,\rho}^\star = 0$ or $\operatorname{supp}(\phi_\rho) \subseteq \mathcal{D}_t$.*

Proof. The initial steps of the proofs follow those of the previous proposition. In particular, let's consider the integral in Eq. (9). Due to the linear independence of the non-zero kernel eigenfunctions on $\mathcal{D}_{t'}$, for this integral to be zero, we have only two possibilities: either *i)* all coefficients $\{(c_{t,\rho}^\star)_{\rho\in\mathbb{N}}\}_{t\in[T],\, t\neq t'}$ must be zero or *ii)* the eigenfunctions corresponding to non-zero coefficient $c_{t,\rho}^\star$ $(t \neq t')$ must be zero in $\mathcal{D}_{t'}$. Since the proposition is valid for any set of functions, condition *i)* is not feasible. Therefore, condition *ii)* must hold. Furthermore, since Eq. (9) is valid $\forall t' \in [T]$, it follows that the eigenfunctions used to represent each task t' are zero in $\overline{\mathcal{D}}_{t'} = \bigcup_{t\in[T],\, t\neq t'} \mathcal{D}_t$. Consequently, these eigenfunctions are only supported in $\mathcal{D}_{t'}$ or a subset thereof. □

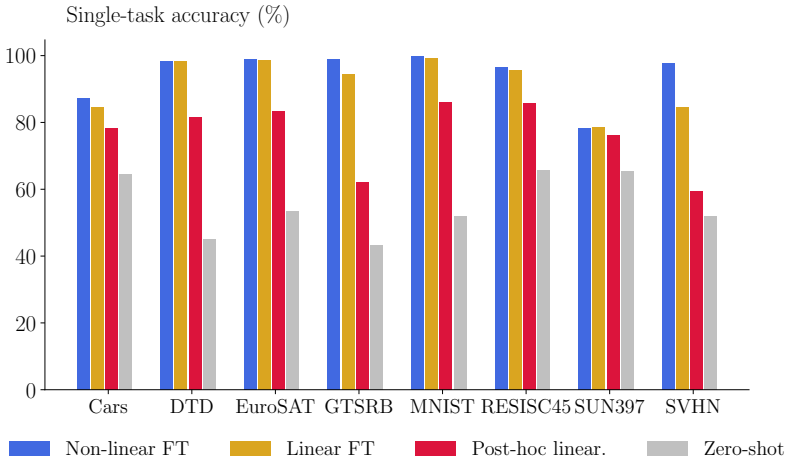
In order to understand the implications of this proposition, it is useful to examine simple data geometries and architectures for which the NTK can be analytically diagonalized. For instance, when data is uniformly distributed on a ring or a torus, the NTK of fully-connected and convolutional neural networks at initialization can be diagonalized with the Fourier series [12, 25, 29, 75]. Fourier atoms are linearly independent on any interval [18] and not localized. Consequently, according to Proposition 3, these architectures cannot perform task arithmetic within such settings. This straightforward calculation aligns with the observation that task arithmetic generally emerges as a property of pre-training and is not inherently present at initialization, as we numerically demonstrated for CLIP models in Section 6.2.

D Further experimental results

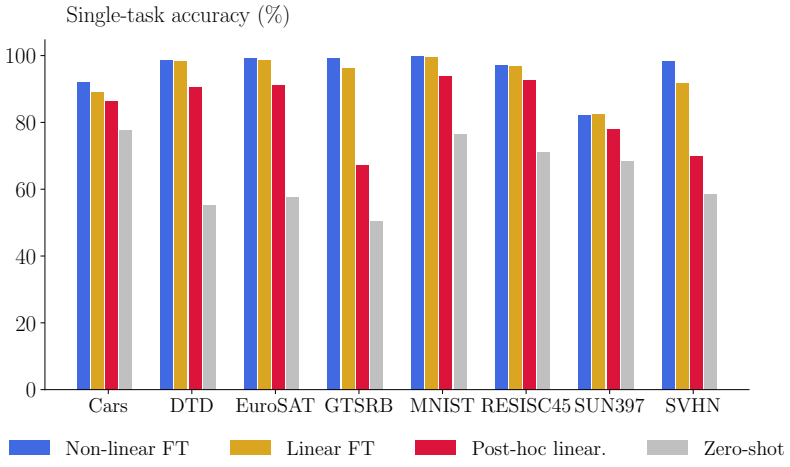
We now present additional experiments that expand the findings discussed in the main text.



(a) ViT-B/32



(b) ViT-B/16



(c) ViT-L/14

Figure 7: **Single-task accuracies (CLIP).** Accuracy of different models obtained using different strategies on each of the tasks.

D.1 Fine-tuning accuracies

In Figure 7, we report the single-task accuracies achieved by different CLIP models before fine-tuning (referred to as *zero-shot*), after fine-tuning with different dynamics (referred to as *non-linear FT* and *linear FT*), and after linearizing the non-linearly fine-tuned models (*post-hoc linearization*).

These results demonstrate that non-linear fine-tuning consistently achieves the highest accuracy, indicating a *non-linear advantage*. However, an interesting observation is that the gap between non-linear, linear, and post-hoc linearized models diminishes as the model size increases. This trend can be explained by the fact that larger models, which are more over-parameterized, inherently induce a stronger kernel behavior during fine-tuning. As a result. they tend to stay closer to the NTK approximation, closing the gap with linearized models.

D.2 Detailed results on task addition

In addition to the results presented in Table 1 in the main text, we report in Figure 8 the absolute accuracies of different CLIP models on the single tasks before (*zero-shot*) and after performing task addition with different strategies (*non-linear fine-tuning*, *post-hoc linearization*, and *linear fine-tuning*).

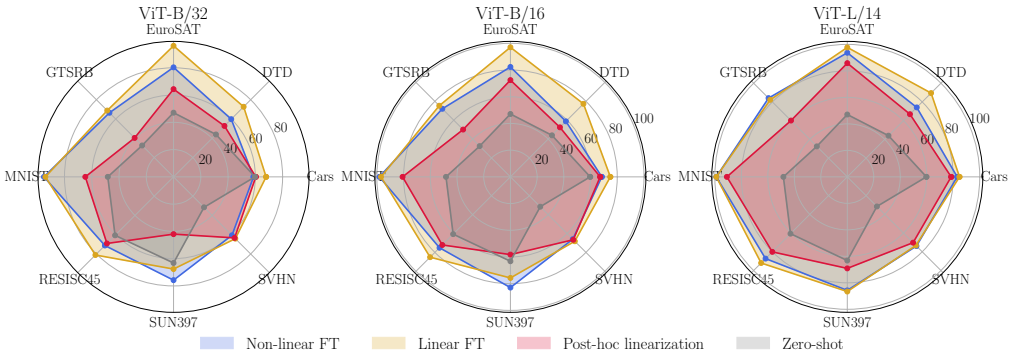


Figure 8: **Task addition performance.** Absolute accuracy (%) of each task after performing task addition with different linear/non-linear strategies and over different CLIP ViT models.

For all models and all datasets, except SUN397 [88], we observe that linearized task arithmetic achieves the highest accuracies. Interestingly, as commented in the main text, the gap in performance between linear and non-linear task vectors decreases while increasing the size of the model. This observation aligns with the previous observation that fine-tuning with larger models is better approximated by the NTK description.

D.3 Weight disentanglement and model scale

We note that weight disentanglement can also explain the increased performance of task arithmetic with model scale. As we see in Figure 9, the size of the areas with a low disentanglement error grows with the model scale. One plausible explanation for this is that larger models inherently induce a stronger kernel behavior during fine-tuning. Namely, since the models have more parameters, each parameter has to change less to fit the training examples. As a result, they tend to stay closer to the NTK approximation, closing the gap with linearized models and taking benefit of the better weight disentanglement of the models lying in the tangent space.

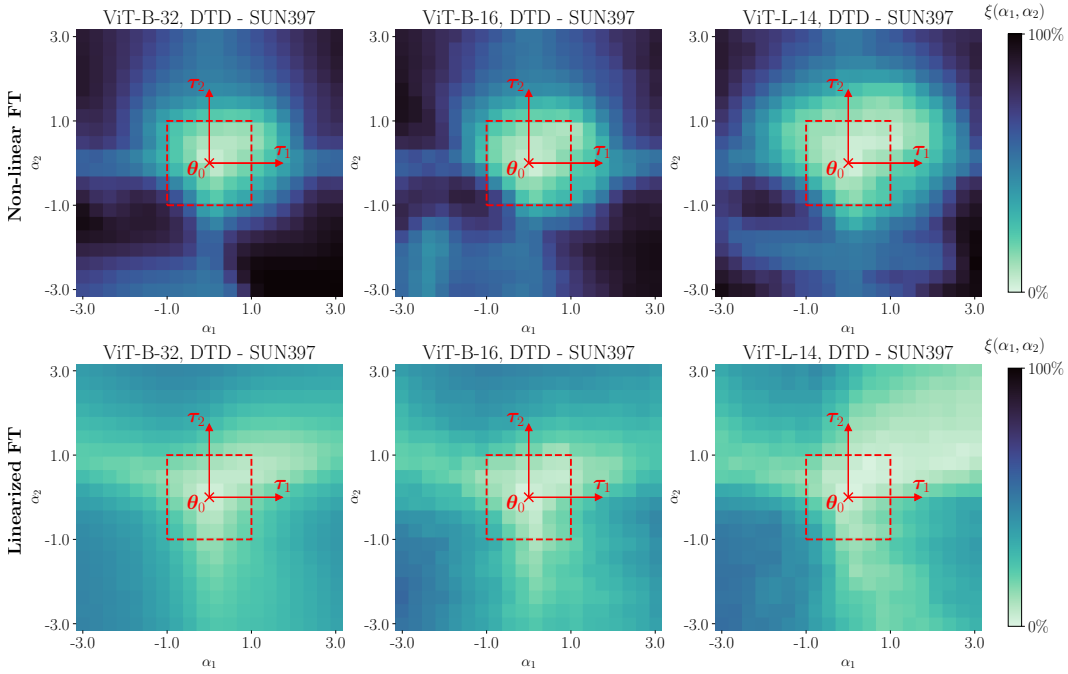


Figure 9: **Weight disentanglement and model scale.** The heatmaps show the disentanglement error $\xi(\alpha_1, \alpha_2)$ of different non-linear CLIP ViTs (top) and their post-hoc linearizations (bottom) on DTD and SUN397. The light regions denote areas of the weight space where weight disentanglement is stronger. The red box delimits the search space used to compute the best α in all our experiments.

D.4 Weight disentanglement of linearized and random models

In Figure 10, we present the disentanglement error of a linearized CLIP ViT-B/32 model across three different dataset pairs. By comparing these results with Figure 3, we can clearly observe that linearized models exhibit significantly more weight disentanglement compared to their non-linear counterparts, similar to the findings obtained for post-hoc linearization.

Conversely, in Figure 11, we showcase the disentanglement error of a CLIP ViT-B/32 model that was non-linearly fine-tuned starting from a random initialization. In all panels, we observe a high disentanglement error, which supports the claim that weight disentanglement and, consequently, task arithmetic are emergent properties of pre-training.

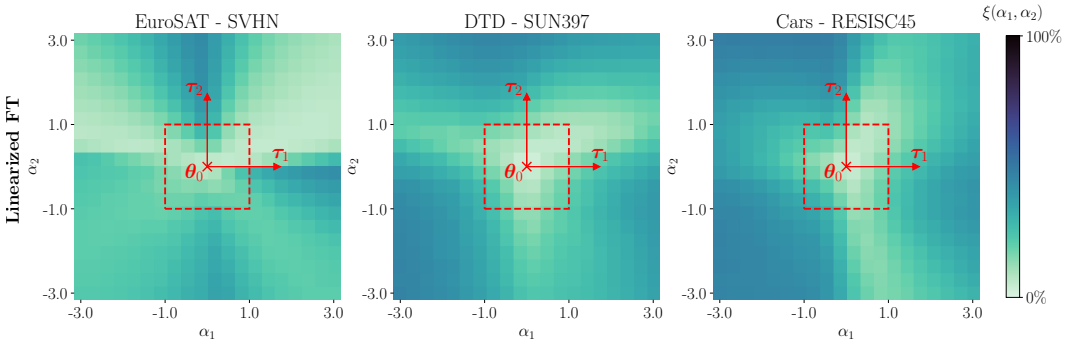


Figure 10: **Visualization of weight disentanglement from linearized models.** The heatmaps show the disentanglement error $\xi(\alpha_1, \alpha_2)$ of a ViT-B/32 linearly fine-tuned on different example task pairs. The light regions denote areas of the weight space where weight disentanglement is stronger. The red box delimits the search space used to compute α in our experiments.

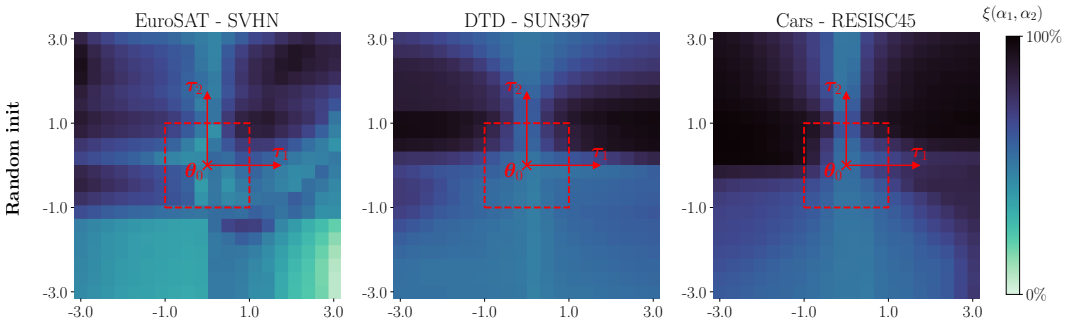


Figure 11: **Visualization of weight disentanglement from random initialization.** The heatmaps show the disentanglement error $\xi(\alpha_1, \alpha_2)$ of a ViT-B/32 fine-tuned from a random initialization on different example task pairs. The light regions denote areas of the weight space where weight disentanglement is stronger. The red box delimits the search space used to compute α in our experiments.

D.5 Task arithmetic with a convolutional architecture

We replicate our task addition experiments using a convolutional architecture rather than a ViT. Specifically, we finetune a ConvNeXt [47] pre-trained on LAION-400M using CLIP [78] on the 8 tasks from our task addition benchmark. We observe that also for this architecture linearized fine-tuning improves task arithmetic performance (see Table 4).

Table 4: **Task addition with a CNN.** Average absolute (%) and normalized accuracies (%) of a CLIP ConvNeXt edited by adding the sum of the task vectors of 8 tasks. We report results for the non-linear and linearized models normalizing performance by their single-task accuracies.

Method		ConvNeXt	
		Abs. (↑)	Norm. (↑)
Pre-trained	$f(\boldsymbol{\theta}_0^{rd})$	57.5	–
Non-lin. FT	$f(\boldsymbol{\theta}_0^{rd} + \boldsymbol{\tau}^{rd})$	79.1	83.6
Linear. FT	$f_{lin}(\boldsymbol{\theta}_0^{rd} + \boldsymbol{\tau}_{lin}^{rd})$	**81.1**	**85.7**

D.6 Task arithmetic with closed vocabulary models

We also replicate our task addition experiments using a ViT-B/16 pre-trained on ImageNet-1k using standard supervised learning. Note that this is a closed vocabulary model, and therefore, when fine-tuning on the different tasks we need to also fine-tune a randomly initialized head. When interpolating the different task vectors we freeze the resulting head to arrive at the final models.

Interestingly, we find that this closed-vocabulary model can do task arithmetic, albeit at a lower performance (both in terms of absolute and normalized accuracy) than the open vocabulary ones (see Table 5). Remarkably, linearized fine-tuning also enhances disentanglement in this model – it yields higher normalized accuracies – but does not perform better than non-linear task addition due to its lower single-task performance.

Table 5: **Task addition with a closed vocabulary model.** Average absolute (%) and normalized accuracies (%) of a ViT-B/16 pre-trained on ImageNet-1k edited by adding the sum of the task vectors of 8 tasks. We report results for the non-linear and linearized models normalizing performance by their single-task accuracies.

Method		ViT-B/16 (supervised)	
		Abs. (↑)	Norm. (↑)
Pre-trained	$f(\boldsymbol{\theta}_0^{rd})$	2.3	–
Non-lin. FT	$f(\boldsymbol{\theta}_0^{rd} + \boldsymbol{\tau}^{rd})$	**54.9**	52.2
Linear. FT	$f_{lin}(\boldsymbol{\theta}_0^{rd} + \boldsymbol{\tau}_{lin}^{rd})$	41.8	**66.1**

D.7 Weight disentanglement in other architectures and modalities

To substantiate the generality of weight disentanglement, we conduct a new experiment on a pre-trained T5-Base model [70] from Hugging Face Hub [84], fine-tuned on two benchmark Natural Language Processing tasks, i.e., sentiment analysis on movie reviews with the IMDB dataset [51] and question answering with the QASC dataset [42]. The results, illustrated in the right panel in Figure 12, show a notable region around the pre-trained checkpoint characterized by low disentanglement error. This finding echoes the ability of T5 to perform task arithmetic as demonstrated in Ilharco et al. [39], thereby reinforcing the robustness of our conclusions.

Similarly, in Figure 12 we also see that the tangent space of CLIP ConvNeXt models is also more disentangled than the non-linear function space around the pre-trained initialization. The same findings also apply to closed vocabulary models, as shown in Figure 13.

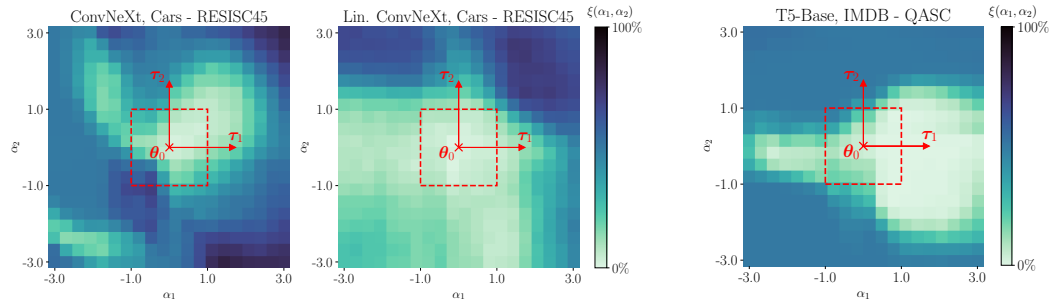


Figure 12: **Visualization of weight disentanglement for other architectures and modalities.** The heatmaps show the disentanglement $\xi(\alpha_1, \alpha_2)$ of a non-linearly and linearly fine-tuned ConvNeXt on a pair of vision tasks (two left panels) and a T5-Base model fine-tuned on a pair of NLP tasks (right).

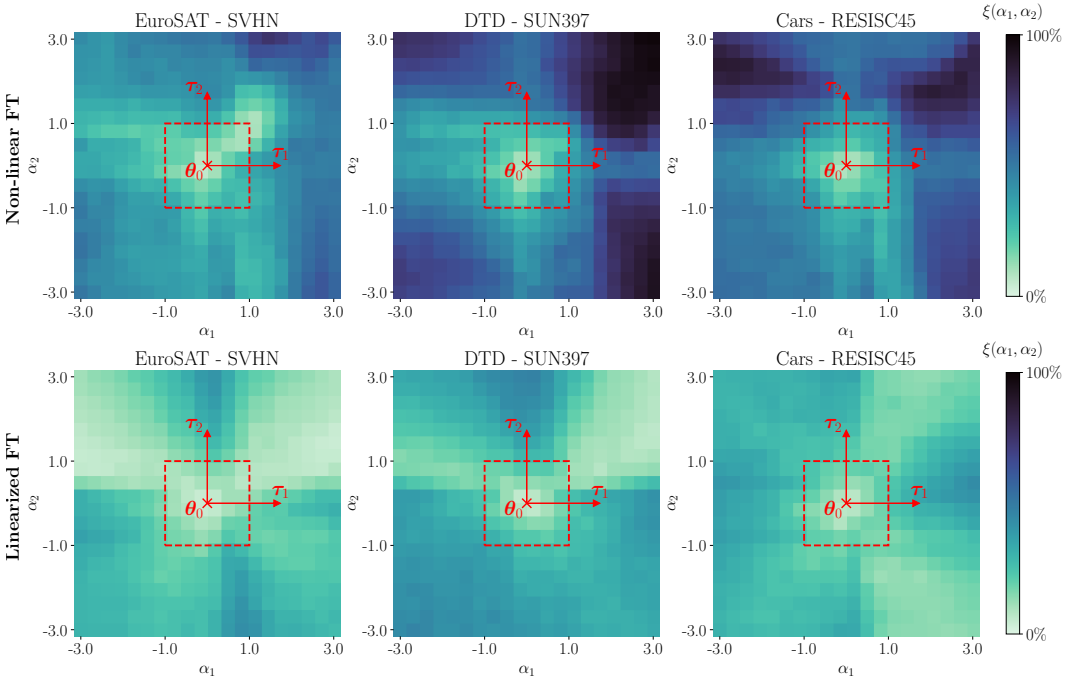


Figure 13: **Weight disentanglement of closed vocabulary models.** The heatmaps show the disentanglement error $\xi(\alpha_1, \alpha_2)$ of a ViT-B/16 model pre-trained on ImageNet-1k using supervised training in the non-linear function space (top) and its linearizations (bottom) on different task pairs. The light regions denote areas of the weight space where weight disentanglement is stronger. The red box delimits the search space used to compute the best α in all our experiments.

D.8 Localization of eigenfunctions of CLIP's NTK

In Figure 14, we plot the local energy of the NTK eigenfunctions for a pre-trained CLIP ViT-B/32 model evaluated on three different data supports and control data supports. These panels complement the information presented in Figure 6 in the main text, where we observed that the CLIP has eigenfunctions whose energy is concentrated on points belonging to the respective dataset.

In Figure 15, we extend this analysis to a randomly-initialized CLIP ViT-B/32 model. In all panels, we observe a non-trivial but considerably poor degree of eigenfunction localization. This observation aligns with the finding that randomly-initialized linearized models cannot perform task arithmetic. Indeed, as we showed in the previous subsection, in this case the model's weights are not effectively disentangled, hindering its ability to perform task arithmetic operations. In summary, eigenfunction localization offers a complementary perspective on the limitations of randomly-initialized models.

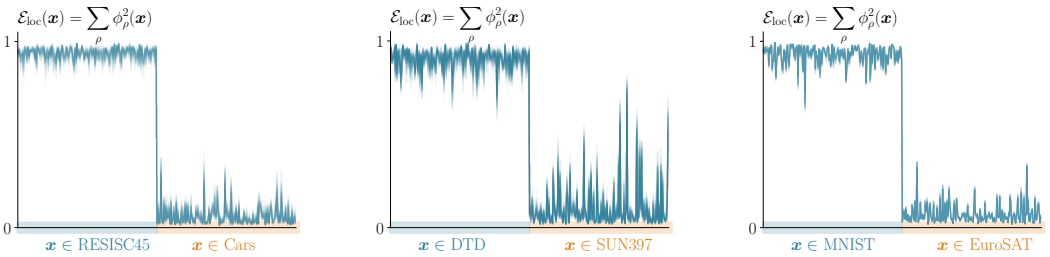


Figure 14: **Eigenfunction localization.** Estimated support of the eigenfunctions of the NTK of a ViT-B/32 CLIP model trained on different datasets. The plot shows the sum of the local energy of the eigenfunctions over a random subset of the training and control supports

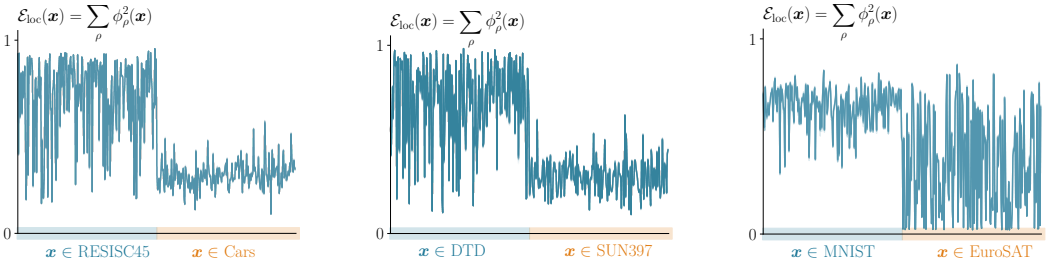


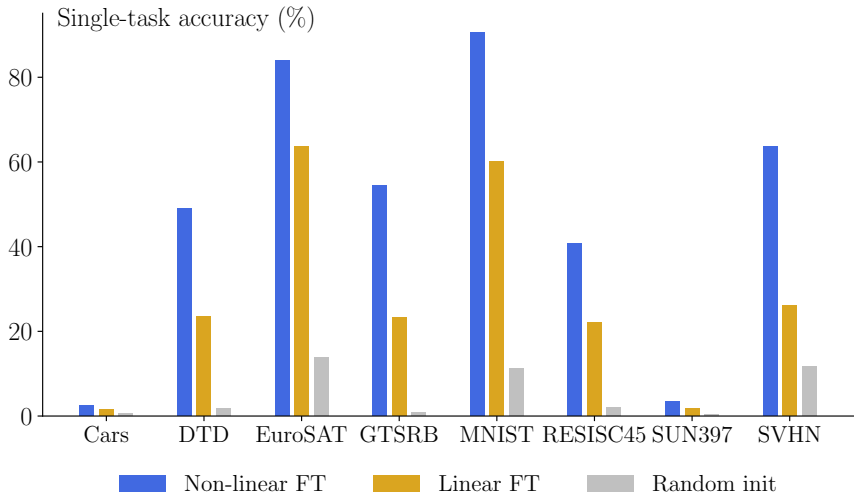
Figure 15: **Eigenfunction localization.** Estimated support of the eigenfunctions of the NTK of a randomly initialized ViT-B/32 model trained on different datasets. The plot shows the sum of the local energy of the eigenfunctions over a random subset of the training and control supports

D.9 Further experiments with randomly-initialized networks

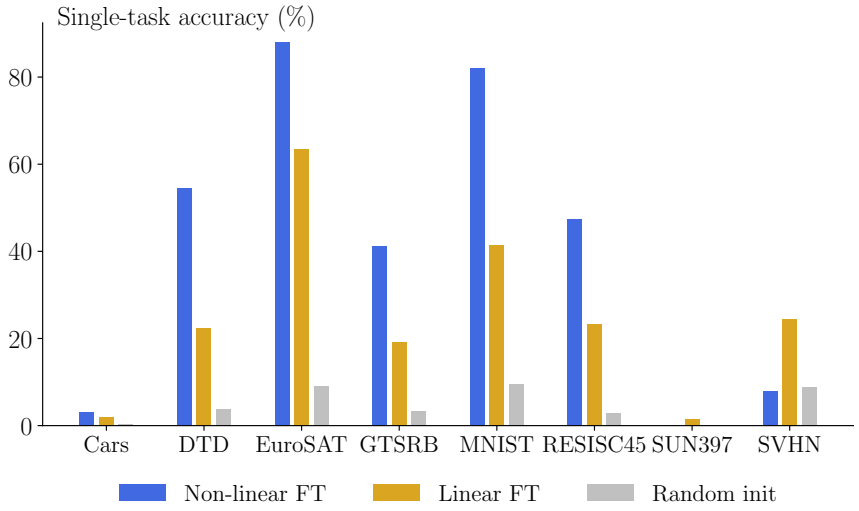
We conclude by showing, in Figure 16, the absolute single-task accuracy achieved by different CLIP ViT models that were fine-tuned from a random initialization. Both the base models achieve non-trivial or moderate accuracy on the majority of benchmark tasks, using both non-linear and linearized fine-tuning dynamics.

These findings reinforce the intuition that non-pretrained models are not failing in task arithmetic due to their inability to learn the task initially. Instead, as argued earlier, the primary reason for the failure of non-pre-trained models in task arithmetic is their lack of weight disentanglement.

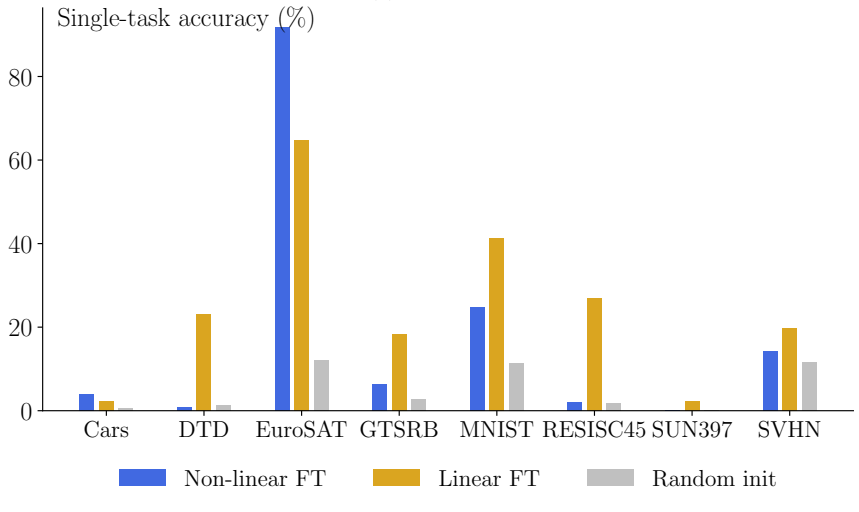
Interestingly, the performance of the randomly-initialized large model is generally poorer compared to the base models. This observation can be attributed to the models' tendency to overfit the training data, which is more likely to occur when a model has a larger capacity.



(a) ViT-B/32



(b) ViT-B/16



(c) ViT-L/14

Figure 16: **Single-task accuracies (random init).** Accuracy of different models obtained using different strategies on each of the tasks.