CAYLEY-PARAMETERIZED TASK-VECTOR TRANSFER: INVARIANT RE-BASING OF FINE-TUNING UPDATES

Anonymous authors
Paper under double-blind review

ABSTRACT

Task vectors capture the changes in a model's weights from fine-tuning and offer a way to reuse learned knowledge across models. Effective task-vector transfer, however, depends on how these updates are aligned and scaled to the recipient model. We introduce Cayley-Parameterized Task-Vector Transfer (CTT), which jointly learns orthogonal residual-stream rotations and scaling factors that re-base the task vectors from the original model to the recipient model. We parameterize the rotations with the Cayley transform, which keeps them orthogonal and differentiable throughout training. We evaluate CTT through three sets of experiments. First, we compare it against other state-of-the-art task-vector transfer methods on both image and text domains using CLIP-ViT B/16. On average, CTT recovers more of the domain than the other methods on the text domains, and performs comparably well to the other methods on the image domains. Then, we apply CTT to natural language autoencoders (NLAs), which describe model activations in natural language and reconstruct them from those descriptions. Since NLAs degrade when their underlying base models are fine-tuned and struggle to reconstruct out-of-distribution activations, task-vector transfer provides an effective way to restore them without full-parameter retraining. We find that CTT consistently recovers more of the NLA domains than the other methods. Finally, we compare CTT directly with GRPO and SFT fine-tuning of the NLA. Under the same training-data distribution, CTT achieves comparable code-domain recovery and higher support retention while using significantly fewer estimated training FLOPs than GRPO, and achieves greater code-domain recovery than SFT. Together, these results show that CTT is the only method among those we compare that respects reparameterization symmetries and transfers task vectors effectively across models. The differences that we observe between the image and text domains also suggest that the effectiveness of task-vector transfer methods may be modality-specific.

1 INTRODUCTION

Machine learning researchers rely on task-specific fine-tuning to adapt large pre-trained models (Radford et al., 2021; Hu et al., 2022), as fine-tuning achieves strong downstream task performance at a fraction of the cost of pre-training the model from scratch (Devlin et al., 2019). However, as data improves and pre-training pipelines mature (Liu et al., 2019; Cherti et al., 2023; Gadre et al., 2023; Qwen Team, 2024b), newer pre-trained variants of the model are adopted, requiring practitioners to repeat fine-tuning (Rinaldi et al., 2025; Lin et al., 2025). The difficulty is that a fine-tuning update is expressed in the coordinates of the model it was trained on, so it cannot simply be added to a different pre-trained model (Rinaldi et al., 2025).

Two pre-trained models can learn similar features in different coordinates, because neural networks have *reparameterization symmetries*, where permuting or rotating parts of the model can change the parameters without changing the network's function (Entezari et al., 2022; Ainsworth et al., 2023). This leads to a fine-tuning update from one model's coordinates being misaligned with another's. Thus, re-basining exploits symmetries to align the weights of two models before interpolating between them (Ainsworth et al., 2023; Jordan et al., 2023; Theus et al., 2025).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

On the other hand, task-vector transfer reuses the fine-tuning update itself through a task vector, where the difference between a model's fine-tuned and pre-trained weights is added to a different pre-trained model (Ilharco et al., 2023). Adding it directly is often ineffective and can even degrade the target model's existing capabilities (Rinaldi et al., 2025; 2026a), so existing work draws on re-basining to align the source task vector with the target model. For instance, TransFusion (Rinaldi et al., 2025) and Git Re-Basin (Ainsworth et al., 2023) exploit permutation symmetries to align the models' weights; however, permutations are discrete and can be too restrictive (Zhang et al., 2025). Others fit orthogonal maps from activations or gradients (THESEUS, Rinaldi et al., 2026b; BiCo, Son et al., 2026), learn per-layer orthogonal transformations (Kawamoto et al., 2025), or mask entries of the task vector (GradFix, Rinaldi et al., 2026a).

Therefore, existing methods trade off task recovery against respecting reparameterization symmetries. Permutation-based methods like TransFusion respect the symmetries but recover very little of the fine-tuning domain, whereas methods like GradFix recover substantially more of the fine-tuning domain but do not respect reparameterization symmetries (Rinaldi et al., 2025; 2026a). We prioritize adherence to reparameterization symmetries because a transfer that respects them changes only the coordinates in which the task vector is written, never its content, so the update it adds is always the source's own fine-tuning update, expressed in another basis. If a method does not respect reparameterization symmetries, it is unclear whether the task vector transfers knowledge from the source model or knowledge fit to the target data.

To this end, we introduce Cayley-Parameterized Task-Vector Transfer (CTT), a task-vector transfer method that respects reparameterization symmetries. CTT jointly learns a single orthogonal rotation in the special orthogonal group $\mathrm{SO}(d)$ that is applied to the residual stream of the model, together with a task-vector scaling coefficient, similar to Chen et al. (2025). Like learned matching (Theus et al., 2025), CTT fits both to a downstream objective rather than matching weights or activations. Because the residual stream is invariant to orthogonal transformations once normalization gains are folded into nearby weights (Ashkboos et al., 2024), the rotation only changes the coordinates in which the task vector is written. Additionally, following prior work on reparameterization of deep networks (Helfrich et al., 2018; Li & Shen, 2026; Yang et al., 2026), we use the Cayley transform to parameterize the rotation, ensuring that it remains in $\mathrm{SO}(d)$ throughout training.

Empirically, we show that CTT is the most effective task-vector transfer method among those we evaluate that respect reparameterization symmetries. Furthermore, CTT matches or outperforms most methods that do not respect reparameterization symmetries, although results vary depending on the specific setting and domain. We additionally show that CTT performs better than other methods we evaluate on natural language autoencoders (NLAs; Fraser-Taliente et al., 2026), an interpretability tool that is expensive to train and tends to degrade whenever its base model is fine-tuned. We make the following contributions:

- First, we introduce Cayley-Parameterized Task-Vector Transfer (CTT), a method for task-vector transfer that respects reparameterization symmetries and jointly learns a rotation parameterized by the Cayley transform and a task-vector transfer scaling factor.
- Second, we experimentally compare CTT with state-of-the-art task-vector transfer methods, both those that respect reparameterization symmetries and those that do not, across three settings: CLIP-ViT B/16 image encoders, CLIP-ViT B/16 text encoders, and natural language autoencoders (NLAs). We find that CTT recovers substantially more of the fine-tuning task domain than other methods that respect reparameterization symmetries, while matching or outperforming most methods that do not respect these symmetries.
- Finally, we compare CTT with directly training an NLA on the code domain using full-parameter Group Relative Policy Optimization (GRPO) (Shao et al., 2024) and supervised fine-tuning (SFT) (Ouyang et al., 2022). Under each method's standard configuration, CTT recovers more code-domain fraction of variance explained (FVE) and retains more support-domain FVE than both. Under matched training data, CTT matches GRPO's code-domain FVE with roughly $9\times$ fewer floating-point operations (FLOPs), outperforms SFT at a similar FLOP budget, and trains with roughly $1000\times$ fewer parameters.

2 Related Work

Model merging. Model merging combines models that share pre-training checkpoints but have different fine-tuned checkpoints. Model Soups (Wortsman et al., 2022) naively averages the weights of models, often improving accuracy and generalization. However, combining models from different fine-tuned checkpoints can introduce task interference: TIES-Merging (Yadav et al., 2023) trims small-magnitude parameters and resolves sign conflicts before merging, DARE (Yu et al., 2024) randomly drops parameters and rescales the remaining ones, and Gargiulo et al. (2025) use singular value decomposition (SVD) to decorrelate singular directions. More recent work takes advantage of the geometries of model weights: OrthoMerge (Yang et al., 2026) applies orthogonal transformations (Cayley) and combines models in skew-symmetric coordinates, CORAM (Sui et al., 2026) decomposes weights into rotational, spectral, and subspace components, and Chen et al. (2025) jointly learn task-specific feature-space rotations and task-vector scaling coefficients.

Re-basin. Model re-basining extends model merging by exploiting symmetries to align independently trained models to a shared weight space. It builds on mode connectivity: the loss basins of independently trained models can be connected by curved paths of low loss (Garipov et al., 2018), and these paths are also linear when the models share sufficient parts of initial training (Frankle et al., 2020). Git Re-Basin (Ainsworth et al., 2023) aligns models with permutations, building on Entezari et al. (2022). Peña et al. (2023) make permutations differentiable by applying Sinkhorn relaxations, and other work aligns models with methods such as optimal transport (Singh & Jaggi, 2020; Imfeld et al., 2024). More recent work such as Zhang et al. (2025) introduces continuous rotation symmetries, which serve as a richer form of coordinate alignment, and Theus et al. (2025) learn both permutations and orthogonal rotations to align models. Finally, Li & Shen (2026) extend re-basining to billion-parameter models by jointly optimizing both models and apply the Cayley transform to maintain orthogonality during training.

Task-vector transfer. Task-vector transfer reuses task-specific fine-tuning updates across models (Lin et al., 2025). Task Arithmetic (Ilharco et al., 2023) applies linear operations to task vectors to add or suppress the task domain within the same pre-trained model. Building on Task Arithmetic, TransFusion respects reparameterization symmetries by exploiting permutations, with separate inter- and intra-head permutations (Rinaldi et al., 2025), to transfer task vectors from one pre-trained model to another. Methods that do not respect reparameterization symmetries, such as Kawamoto et al. (2025), learn layerwise orthogonal transformations with an orthogonality penalty term in the objective. GradFix masks task-vector components whose signs disagree with the target model's negative gradient (Rinaldi et al., 2026a). THESEUS accommodates models of different widths by using orthogonal Procrustes (Schönemann, 1966) to align input and output activations (Rinaldi et al., 2026b). Finally, BiCo exploits the bilinear relationship between input-side activations and output-side gradients and finds orthogonal alignments that transfer task vectors across models of different sizes and pre-training configurations (Son et al., 2026). Like TransFusion, CTT respects reparameterization symmetries, but it learns a continuous rotation rather than a permutation. Unlike Kawamoto, THESEUS, and BiCo, it applies a single rotation consistently to every layer that reads from or writes to the residual stream.

3 Preliminaries

Task-Vector Transfer. A task vector $\tau = \theta_A^{\text{ft}} - \theta_A$ measures the change in a model's weights from fine-tuning (Ilharco et al., 2023). We transfer a task vector from a source model A to a recipient model B as follows: $\theta'_B = \theta_B + \alpha\,\pi(\tau)$, where α is a scalar that scales the transferred update and π maps the task vector from A's parameter coordinates into B's. We say π respects reparameterization symmetries if it is a symmetry of B's architecture, i.e., $f(x;\pi(\theta)) = f(x;\theta)$ for all inputs x.

Natural Language Autoencoders (NLAs). NLAs (Fraser-Taliente et al., 2026) translate model activations into natural-language descriptions. NLAs are made up of two components: the activation verbalizer (AV) describes an activation at a selected token and layer, and the activation reconstructor (AR) reconstructs that activation from the description. The AV is trained via GRPO, and the AR is trained via regression (see section B.3). Both components are initialized from the base model and trained to minimize reconstruction error.

4 CAYLEY-PARAMETERIZED TASK-VECTOR TRANSFER (CTT)

4.1 PARAMETRIC FORMULATION OF CTT

CTT is an architecture-dependent and application-dependent transfer method. Let $\tau = \theta_{\text{src}}^{\text{ft}} - \theta_{\text{src}}^{\text{base}}$ be the source model's task vector, and $\mathcal{C}$ denote selected recipient components. The parameters of each selected component are denoted as θ_c, and its corresponding task vector is denoted as τ_c. Each component c has its own learned rotation $R_c \in \text{SO}(d_c)$ and scale $\alpha_c > 0$, shared across its layers, where d_c is the dimension of the model's residual stream.

Thus, we write the component-specific task-vector transfer as follows:

$$\theta_{\text{rec},c}(\phi) = \theta_{\text{rec},c} + \alpha_c \, \mathcal{T}_c(R_c)[\tau_c], \qquad c \in \mathcal{C}. \tag{1}$$

$\mathcal{T}_c$ specifies what is being rotated; in CTT, we are rotating the weights that write to and read from the residual stream, and parameters outside of $\mathcal{T}_c$ remain unchanged. In parametric form, CTT jointly learns rotations and task-vector scalar coefficients to minimize

$$\min_{\phi} \; \mathcal{L}_{\text{app}}\big(\theta(\phi); \mathcal{D}_{\text{fit}}\big), \tag{2}$$

where $\phi = \{(R_c, \alpha_c)\}_{c \in \mathcal{C}}$ is the collection of all learned rotations and scaling coefficients for each component, $\theta(\phi)$ denotes, for each of the recipient model's components, its updated parameters after applying the component-specific transfers, and $\mathcal{D}_{\text{fit}}$ denotes the fitting data. The specific choice of $\mathcal{L}_{\text{app}}$ depends on the application.

4.2 PARAMETERIZATIONS

In CTT, during each step in training, we parameterize both the orthogonal rotation R_c and the task-vector scalar coefficient α_c. Specifically, we parameterize R_c using the Cayley transform (Cayley, 1846; Helfrich et al., 2018; Lezcano-Casado & Martínez-Rubio, 2019). This parameterization ensures orthogonality throughout training and allows us to perform gradient-based updates to an unconstrained parameter A_c (see section A.5).

We define the Cayley transform as follows:

$$R_c = \text{Cay}(A_c) = (I - S_c)^{-1}(I + S_c), \quad S_c = A_c - A_c^{\top}, \tag{3}$$

where S_c is skew-symmetric, thereby ensuring $R_c \in \text{SO}(d_c)$ for every $A_c \in \mathbb{R}^{d_c \times d_c}$. We also parameterize the task-vector scalar coefficient using $\alpha_c = e^{s_c}$, ensuring that α_c remains positive, and optimize (A_c, s_c) jointly with Adam (Kingma & Ba, 2017).

4.3 GENERAL ALGORITHM & REPARAMETERIZATION SYMMETRIES

See algorithm 1 for the complete, general formulation of Cayley-Parameterized Task-Vector Transfer (CTT).

Before forming the task vector on the source model, we fold normalization terms into adjacent weights and biases. For RMSNorm, we absorb the learned gains into the weights that read from the residual stream. For LayerNorm, we additionally absorb the normalization biases into reader biases and fold mean-centering into weights that write to the residual stream (see section A.6).

Therefore, when applying a reparameterization symmetry π to the task vector, we have:

$$\tau_{\text{read}} \mapsto [\pi(\tau)]_{\text{read}}, \qquad \tau_{\text{write}} \mapsto [\pi(\tau)]_{\text{write}}, \qquad \tau_{b_{\text{write}}} \mapsto [\pi(\tau)]_{b_{\text{write}}},$$

where τ_{read}, τ_{write}, and $\tau_{b_{\text{write}}}$ are the task vectors corresponding to the folded weights that read from the residual stream, the weights that write to the residual stream, and the writer biases, respectively.

We assess whether a task-vector transfer method respects reparameterization symmetry ($f(x; \pi(\theta)) = f(x; \theta)$) according to the following two conditions: **(a)** After applying the symmetries to the weights of the model, the model's computations must still lie in the same basis. If weights write to a new basis, subsequent readers of these weights must compensate for this change. **(b)** The symmetry applied must preserve normalization and nonlinearities. We explain this formalization in greater detail in the Appendix (see section A.3).

Algorithm 1 Cayley-Parameterized Task-Vector Transfer (CTT)

Input: Recipient weights θ_{rec}; architecture-dependent components $\mathcal{C}$; component-specific task vectors $\{\tau_c\}_{c\in\mathcal{C}}$; loss objective $\mathcal{L}_{\text{app}}$; fitting data $\mathcal{D}_{\text{fit}}$; holdout $\mathcal{H}$; validation score $\mathcal{M}_{\text{app}}$; steps T.

Initialize $(A_c, s_c) \leftarrow (0, 0)$ for each $c \in \mathcal{C}$.
for $t = 1, \ldots, T$ **do**
  Parameterize: $R_c^{(t)} \leftarrow \text{Cay}(A_c)$, $\alpha_c^{(t)} \leftarrow e^{s_c}$ for each $c \in \mathcal{C}$.
  Set $\phi^{(t)} \leftarrow \{(R_c^{(t)}, \alpha_c^{(t)})\}_{c\in\mathcal{C}}$.
  Transfer: $\theta_{\text{rec},c}(\phi^{(t)}) \leftarrow \theta_{\text{rec},c} + \alpha_c^{(t)} \mathcal{T}_c(R_c^{(t)})[\tau_c]$ for each $c \in \mathcal{C}$.
  Sample batch B from $\mathcal{D}_{\text{fit}}$.
  Objective: $L \leftarrow \mathcal{L}_{\text{app}}\big(\theta(\phi^{(t)}); B\big)$.
  Update: $(A, s) \leftarrow \text{Adam}\big((A, s), \nabla_{A,s} L\big)$.
end for
Select: $t^\star \leftarrow \arg\max_{t\in\{1,\ldots,T\}} \mathcal{M}_{\text{app}}\big(\theta(\phi^{(t)}); \mathcal{H}\big)$.
return $\{(R_c^{(t^\star)}, \alpha_c^{(t^\star)})\}_{c\in\mathcal{C}}$.

According to this formalization, CTT respects reparameterization symmetry. This is important because, in an effort to better understand the loss geometries of models, we want to measure how much of the task-specific knowledge encoded in the source model's task vector we can transfer to the recipient model. We are only able to test this if the method we employ for task-vector transfer respects reparameterization symmetries.

4.4 CTT ON NATURAL LANGUAGE AUTOENCODERS (NLAS)

When applying CTT to NLAs, we transfer a task vector from a base model to the AV and AR components of an NLA trained on it. Under a shared reconstruction objective, CTT jointly optimizes separate orthogonal rotations and task-vector scaling coefficients for the AV and AR.

We evaluate the resulting NLA on a batch of activations $B = \{x_i\}_{i=1}^{|B|}$ sampled from $\mathcal{D}_{\text{fit}}$. This dataset contains activations extracted at selected token positions from a mix of domain and support documents (see section B.5). For each activation x_i, the AV first generates a verbalization, and then the AR produces a reconstruction $\hat{x}_i$ from that verbalization.

We compute the following reconstruction objective:

$$\mathcal{L}_{\text{app}}\big(\theta(\phi); B\big) = \frac{1}{|B|} \sum_{i=1}^{|B|} \|\text{nrm}(\hat{x}_i) - \text{nrm}(x_i)\|_2^2, \qquad \text{nrm}(x) = \sqrt{d}\, \frac{x}{\|x\|_2}. \tag{4}$$

After computing the loss objective, we backpropagate to update (A_c, s_c), the parameterizations of (R_c, α_c), the orthogonal rotations and task-vector scaling coefficients. However, because the AV's verbalizations are sequences of discrete tokens, backpropagation cannot pass through the AV's token embeddings. Therefore, we address this by instead backpropagating against the weighted sum of the candidate token embeddings at each position (the soft embedding), following the straight-through estimator (Bengio et al., 2013). In totality, we have:

$$e_j = e_j^{\text{hard}} + e_j^{\text{soft}} - \text{sg}(e_j^{\text{soft}}), \tag{5}$$

where e_j^{soft} is the aforementioned soft embedding, and $e_j^{\text{hard}} = W_E[y_j]$ is the hard embedding of the token generated by the AV at position j. Gradients can now flow through e_j^{soft} to the AV's token probabilities.

4.5 CTT ON CLIP-VIT

When applying CTT to CLIP, we transfer a task vector to either the image encoder or the text encoder. We jointly optimize the orthogonal rotation and task-vector scaling coefficient using a

cross-entropy loss objective, $\mathcal{L}_{\text{app}}$, between the model's predicted class probabilities and the ground-truth labels. The remaining steps follow the general algorithm in algorithm 1.

5 CTT vs Other Methods

In this section, we compare CTT with other state-of-the-art task-vector transfer methods. We compare across two metrics: recovery, the fraction of the task domain recovered by the task-vector transfer method, and retention, the fraction of the support domain retained. We first present results for CLIP-ViT B/16 image and text encoders, followed by results for NLAs trained on Qwen3-4B (Qwen Team, 2025) and Qwen2.5-7B-Instruct (Qwen Team, 2024b). We distinguish between methods that, like CTT, respect reparameterization symmetries and those that do not. We report results on each method averaged across three seeds. We explain how we fine-tune CLIP-ViT B/16 and the NLAs on their task domains in the Appendix (see sections B.4 and B.9).

5.1 CLIP-ViT

CTT performs better than or comparably to other task-vector transfer methods on CLIP-ViT B/16's image and text encoders, the standard benchmark used in the literature. We evaluate on eight image classification datasets following Ilharco et al. (2023): Cars (Krause et al., 2013), DTD (Cimpoi et al., 2014), EuroSAT (Helber et al., 2019), GTSRB (Stallkamp et al., 2012), MNIST (LeCun et al., 1998), RESISC45 (Cheng et al., 2017), SUN397 (Xiao et al., 2010), and SVHN (Netzer et al., 2011); and on four GLUE tasks (Wang et al., 2019): QQP, SST-2, RTE, and CoLA.

We first consider the image encoder (see table 1). Across *all* eight image domains, CTT recovers more of the task domain compared to the best method that also respects reparameterization symmetry. When we instead compare CTT to methods that do *not* respect reparameterization symmetry, we find that CTT performs better than average and only slightly trails behind the best method.

Next, we consider the text encoder (see table 2). Across *all* four text domains, CTT again recovers substantially more of the task domain than the best method that also respects reparameterization symmetry. When we instead compare CTT to methods that do *not* respect reparameterization symmetry, CTT recovers more than the best of them on three of the four domains. Averaged over domains, CTT recovers the most of any method.

On the image domains, support retention varies by task, with larger losses on DTD and EuroSAT. Additionally, variability in performance across different domains may suggest that the effectiveness of task-vector transfer is modality-specific (see section C.4 and table C.4).

5.2 NLAs

NLAs degrade when their base model is fine-tuned and reconstruct poorly on out-of-distribution data ((section C.1). Transferring the fine-tuned base model's task vector to the NLA can therefore restore its reconstructions on the new domain without retraining the NLA.

CTT outperforms every other task-vector transfer method on NLAs, regardless of whether the method respects reparameterization symmetry (see table 3).

For the NLA trained on Qwen3-4B, across *all* four NLA domains, CTT recovers substantially more of the task domain than the best method that also respects reparameterization symmetry. When we instead compare CTT to methods that do *not* respect reparameterization symmetry, CTT recovers more than their average on every domain and more than the best of them on three of the four. CTT does so while retaining all of the support FVE, and *for the NLA trained on Qwen2.5-7B-Instruct, it outperforms every method on every domain.*

To complement these results, we compare NLA verbalizations of activations extracted from models fine-tuned on code and prose (see fig. 1). In both examples, CTT identifies the domain and describes what the model is doing more accurately than the other transfer methods.

Table 1: On CLIP image domains, CTT outperforms other task-vector transfer methods that also respect reparameterization symmetries and performs comparably well to those that do not. Zero-shot is the baseline performance without task-vector transfer.

		cars		dtd		eurosat		gtsrb	
Method	Reparameterization	TASK / REC	SUPP / RET	TASK / REC	SUPP / RET	TASK / REC	SUPP / RET	TASK / REC	SUPP / RET
Zero-shot	–	88.58 / 0.00	79.77 / 1.00	56.01 / 0.00	79.77 / 1.00	49.35 / 0.00	79.77 / 1.00	48.38 / 0.00	79.77 / 1.00
Naive	–	-5.41 / -1.38	71.90 / 0.90	-0.64 / -0.02	77.10 / 0.97	-6.41 / -0.13	53.48 / 0.67	-11.11 / -0.22	71.13 / 0.89
Git Re-Basin	symmetric	-1.26 / -0.32	78.81 / 0.99	+0.12 / 0.00	77.99 / 0.98	-7.08 / -0.14	74.03 / 0.93	+1.81 / 0.04	78.68 / 0.99
TransFusion	symmetric	-1.23 / -0.31	79.03 / 0.99	+0.21 / 0.01	78.04 / 0.98	-5.42 / -0.11	75.84 / 0.95	+1.45 / 0.03	78.82 / 0.99
GradFix	non-symmetric	-3.47 / -0.88	79.76 / 1.00	+3.72 / 0.14	79.27 / 0.99	+16.36 / 0.33	79.80 / 1.00	+16.68 / 0.33	79.73 / 1.00
THESEUS	non-symmetric	-0.22 / -0.06	79.95 / 1.00	+2.89 / 0.11	77.10 / 0.97	+10.97 / 0.22	75.69 / 0.95	+16.13 / 0.32	76.67 / 0.96
BiCo	non-symmetric	-0.20 / -0.05	78.63 / 0.99	**+15.39 / 0.58**	77.59 / 0.97	+29.47 / 0.59	77.96 / 0.98	**+32.72 / 0.65**	77.57 / 0.97
Kawamoto	non-symmetric	**+0.09 / 0.02**	80.02 / 1.00	+3.02 / 0.11	79.71 / 1.00	+28.15 / 0.57	79.72 / 1.00	+23.42 / 0.46	80.45 / 1.01
CTT	symmetric	-0.36 / -0.09	79.77 / 1.00	+8.19 / 0.31	70.30 / 0.88	**+32.39 / 0.65**	66.33 / 0.83	+21.63 / 0.43	74.73 / 0.94

		mnist		resisc45		sun397		svhn	
Method	Reparameterization	TASK / REC	SUPP / RET	TASK / REC	SUPP / RET	TASK / REC	SUPP / RET	TASK / REC	SUPP / RET
Zero-shot	–	65.78 / 0.00	79.77 / 1.00	67.90 / 0.00	79.77 / 1.00	70.44 / 0.00	79.77 / 1.00	50.52 / 0.00	79.77 / 1.00
Naive	–	+1.30 / 0.04	75.15 / 0.94	-2.76 / -0.09	75.93 / 0.95	-2.80 / -0.33	72.67 / 0.91	-24.82 / -0.52	67.67 / 0.85
Git Re-Basin	symmetric	+3.87 / 0.11	78.83 / 0.99	-1.46 / -0.05	78.26 / 0.98	-0.85 / -0.10	78.32 / 0.98	-4.02 / -0.08	77.92 / 0.98
TransFusion	symmetric	+5.47 / 0.16	78.94 / 0.99	-1.59 / -0.05	78.40 / 0.98	-0.75 / -0.09	78.75 / 0.99	-2.82 / -0.06	78.13 / 0.98
GradFix	non-symmetric	+23.07 / 0.68	79.87 / 1.00	+3.92 / 0.13	79.70 / 1.00	+0.95 / 0.11	79.84 / 1.00	+21.69 / 0.46	79.90 / 1.00
THESEUS	non-symmetric	+20.10 / 0.59	75.40 / 0.95	+4.22 / 0.14	77.98 / 0.98	+1.13 / 0.13	79.06 / 0.99	+14.52 / 0.31	78.08 / 0.98
BiCo	non-symmetric	+24.31 / 0.72	77.53 / 0.97	**+17.70 / 0.60**	77.60 / 0.97	**+3.31 / 0.39**	77.70 / 0.97	+24.82 / 0.52	77.83 / 0.98
Kawamoto	non-symmetric	+26.46 / 0.78	79.56 / 1.00	+6.69 / 0.23	79.83 / 1.00	+0.60 / 0.07	79.63 / 1.00	+25.81 / 0.54	79.79 / 1.00
CTT	symmetric	**+28.12 / 0.83**	74.89 / 0.94	+11.63 / 0.39	75.22 / 0.94	+0.41 / 0.05	76.61 / 0.96	**+28.31 / 0.60**	76.70 / 0.96

Table 2: On CLIP text domains, CTT outperforms most other task-vector transfer methods, regardless of whether they respect reparameterization symmetries.

Method	Reparameterization	qqp ACC / REC	sst2 ACC / REC	rte ACC / REC	cola ACC / REC
Zero-shot	–	52.30 / 0.00	51.26 / 0.00	45.85 / 0.00	43.91 / 0.00
Naive	–	-15.24 / -0.50	-0.34 / -0.01	+1.44 / 0.57	+12.46 / 0.54
Git Re-Basin	symmetric	-13.99 / -0.46	+0.61 / 0.02	+0.24 / 0.10	-10.39 / -0.45
TransFusion	symmetric	-13.36 / -0.44	+0.34 / 0.01	+0.60 / 0.24	-10.35 / -0.45
GradFix	non-symmetric	-0.71 / -0.02	+2.19 / 0.06	+1.40 / 0.55	+4.53 / 0.20
THESEUS	non-symmetric	+1.58 / 0.05	+1.54 / 0.04	+0.46 / 0.18	+18.52 / 0.80
BiCo	non-symmetric	-5.90 / -0.19	+0.63 / 0.02	+1.47 / 0.58	**+25.16 / 1.09**
Kawamoto	non-symmetric	+0.67 / 0.02	+3.05 / 0.09	+1.18 / 0.47	+1.90 / 0.08
CTT	symmetric	**+6.11 / 0.20**	**+7.54 / 0.22**	**+5.08 / 2.01**	+10.90 / 0.47

6 CTT vs Direct NLA Training (GRPO, SFT)

In this section, we first compare CTT with direct training using GRPO (Shao et al., 2024) and SFT to recover code-domain performance in the Qwen3-4B NLA. We then show that CTT requires far less compute than GRPO and far fewer trainable parameters than either method. We describe in detail the GRPO and SFT training procedures in the Appendix (see section B.10).

6.1 Task Domain Recovery and Support Retention

Following the standard NLA training recipe (Fraser-Taliente et al., 2026), GRPO and SFT update all parameters of the AV and AR, and each method, including CTT, uses its own configuration and data distribution (fig. 2). For GRPO, we varied the learning rate, gradient clipping, and advantage normalization. We also tested different KL reference models, per-token clamping, removing the KL penalty, and replaying FineWeb (Penedo et al., 2024) support-domain data. We report the configuration with the best trade-off between code- and support-domain FVE. For SFT, we followed the training procedure used for the original NLA. For CTT, we compared learning-rate schedules (see section B.7, table B.1, and fig. B.2). We find that CTT recovers more code-domain FVE and retains more support-domain FVE than both GRPO and SFT.

Table 3: Across both Qwen3-4B and Qwen2.5-7B-Instruct NLA domains, CTT outperforms other task-vector transfer methods regardless of whether they respect reparameterization symmetries.

		code		prose		hardcode		animalwelf	
Method	Reparameterization	TASK / REC	SUPP / RET	TASK / REC	SUPP / RET	TASK / REC	SUPP / RET	TASK / REC	SUPP / RET
Qwen3-4B									
Zero-shot	–	0.23 / 0.00	0.69 / 1.00	0.54 / 0.00	0.69 / 1.00	0.24 / 0.00	0.69 / 1.00	0.47 / 0.00	0.69 / 1.00
Naive	–	-0.00 / 0.00	0.68 / 0.99	-0.00 / 0.00	0.67 / 0.97	+0.03 / 0.04	0.68 / 0.99	+0.05 / 0.08	0.65 / 0.94
Git Re-Basin	symmetric	-0.00 / 0.00	0.68 / 0.99	-0.00 / 0.00	0.67 / 0.97	+0.03 / 0.04	0.68 / 0.99	+0.05 / 0.08	0.65 / 0.94
TransFusion	symmetric	-0.00 / 0.00	0.68 / 0.99	-0.00 / 0.00	0.67 / 0.97	+0.03 / 0.04	0.68 / 0.99	+0.05 / 0.08	0.65 / 0.94
GradFix	non-symmetric	+0.08 / 0.11	0.67 / 0.97	+0.04 / 0.10	0.67 / 0.98	+0.06 / 0.08	0.69 / 1.00	+0.06 / 0.12	0.61 / 0.89
THESEUS	non-symmetric	+0.01 / 0.01	0.69 / 0.99	+0.00 / 0.00	0.68 / 0.99	+0.03 / 0.03	0.67 / 0.98	+0.03 / 0.05	0.68 / 0.98
BiCo	non-symmetric	+0.01 / 0.01	0.69 / 1.00	+0.01 / 0.02	0.69 / 1.00	+0.01 / 0.01	0.69 / 1.00	+0.01 / 0.02	0.68 / 0.98
Kawamoto	non-symmetric	+0.41 / 0.53	0.71 / 1.00	+0.16 / 0.35	0.70 / 1.00	+0.31 / 0.40	0.70 / 1.00	+0.29 / 0.54	0.71 / 1.00
CTT	symmetric	**+0.43 / 0.56**	0.71 / 1.00	**+0.19 / 0.42**	0.70 / 1.00	**+0.32 / 0.41**	0.69 / 1.00	**+0.29 / 0.54**	0.70 / 1.00
Qwen2.5-7B-Instruct									
Zero-shot	–	0.61 / 0.00	0.75 / 1.00	0.73 / 0.00	0.75 / 1.00	0.54 / 0.00	0.75 / 1.00	0.64 / 0.00	0.75 / 1.00
Naive	–	+0.01 / 0.02	0.73 / 0.97	+0.00 / 0.01	0.73 / 0.98	+0.03 / 0.06	0.74 / 1.00	+0.02 / 0.05	0.68 / 0.91
Git Re-Basin	symmetric	+0.01 / 0.02	0.73 / 0.97	+0.00 / 0.01	0.73 / 0.98	+0.03 / 0.06	0.74 / 1.00	+0.02 / 0.05	0.68 / 0.91
TransFusion	symmetric	+0.01 / 0.02	0.73 / 0.97	+0.00 / 0.01	0.73 / 0.98	+0.03 / 0.06	0.74 / 1.00	+0.02 / 0.05	0.68 / 0.91
GradFix	non-symmetric	+0.00 / 0.01	0.74 / 1.00	-0.01 / 0.00	0.73 / 0.98	+0.01 / 0.03	0.75 / 1.00	-0.04 / 0.00	0.65 / 0.88
THESEUS	non-symmetric	+0.00 / 0.01	0.74 / 0.99	+0.00 / 0.01	0.75 / 1.00	+0.02 / 0.04	0.74 / 0.99	+0.01 / 0.04	0.74 / 0.99
BiCo	non-symmetric	+0.00 / 0.00	0.74 / 1.00	+0.00 / 0.01	0.74 / 1.00	-0.00 / 0.00	0.75 / 1.00	-0.00 / 0.00	0.74 / 1.00
Kawamoto	non-symmetric	+0.14 / 0.35	0.75 / 1.00	+0.07 / 0.26	0.75 / 1.00	+0.15 / 0.33	0.75 / 1.00	+0.15 / 0.41	0.75 / 1.00
CTT	symmetric	**+0.17 / 0.45**	0.76 / 1.00	**+0.10 / 0.35**	0.76 / 1.00	**+0.16 / 0.36**	0.75 / 1.00	**+0.16 / 0.44**	0.74 / 0.99

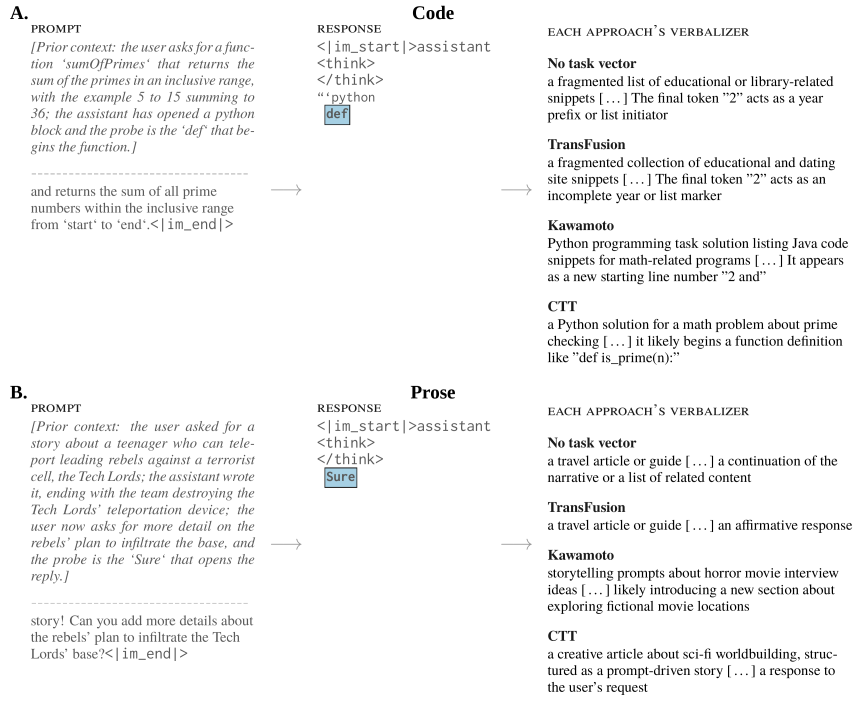

Figure 1: **For activations drawn from the code and prose domains, Qwen3-4B NLA verbalizations produced after CTT describe the model's thoughts more accurately than those produced using other transfer methods.** For the code domain, CTT correctly identifies the model's internal thoughts about writing the Python function. For the prose domain, CTT correctly identifies the model's internal thoughts about the sci-fi story.

6.2 Computational Cost

Next, we compare the computational cost of CTT, GRPO and SFT using the exact same training-data distribution. At their best evaluated checkpoints on code FVE, CTT achieves a code-domain FVE comparable to that of GRPO and higher support FVE with approximately $9.26\times$ fewer FLOPs,

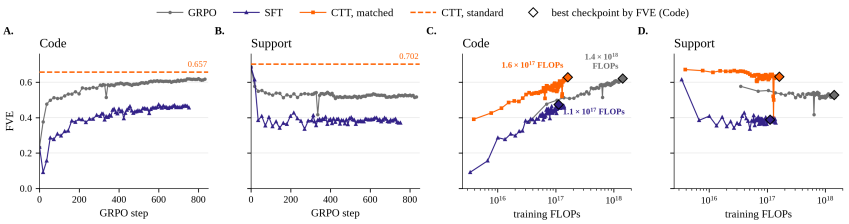

Figure 2: **On the Qwen3-4B NLA, CTT recovers more code-domain FVE and retains higher support-domain FVE than GRPO and SFT under its standard configuration, and uses substantially less compute than GRPO under matched training data.** **(a)** Under standard training, CTT achieves higher code-domain recovery than GRPO and SFT, and **(b)** CTT retains substantially more of the support domain. **(c)** Under the same training-data distribution, CTT achieves code-domain FVE comparable to that of GRPO with substantially fewer FLOPs, and higher code-domain FVE than SFT at the cost of slightly more FLOPs. **(d)** At the best code-domain FVE checkpoints, CTT also retains higher support-domain FVE than both GRPO and SFT.

Table 4: On the Qwen3-4B NLA, CTT is more compute-efficient than GRPO, more effective than SFT, and more parameter-efficient than both GRPO and SFT.

	GRPO	CTT	SFT
FLOPs	1.088×10^{18}	1.175×10^{17}	1.120×10^{17}
Trained parameters	7.3B	6.6M	7.3B
Code FVE	0.609	0.606	0.473

while SFT uses slightly fewer training FLOPs than CTT but recovers substantially less code-domain FVE and retains less support FVE (see table 4 and fig. 2). Additionally, CTT is substantially more parameter-efficient than GRPO and SFT.

7 Discussion & Limitations

In this work, we introduce Cayley-Parameterized Task-Vector Transfer (CTT), a method that respects reparameterization symmetries and jointly learns orthogonal rotations and task-vector scaling coefficients to align task vectors across models. On CLIP-ViT B/16, CTT recovers more of the task domain than other methods that respect reparameterization symmetries across all evaluated image and text domains. Across all methods, CTT achieves the highest average task-domain recovery on the CLIP text domains. Additionally, across both Qwen3-4B and Qwen2.5-7B-Instruct NLAs, CTT recovers more of the task domain than other methods while retaining nearly all of the original support-domain FVE. Furthermore, we also compare CTT with directly training the Qwen3-4B NLA using GRPO and SFT. Under its standard training configuration, CTT achieves higher code-domain FVE and retains more support-domain FVE than both GRPO and SFT. Under the same training-data distribution, CTT achieves code-domain FVE comparable to that of GRPO with substantially fewer FLOPs, and higher code-domain FVE than SFT with slightly more FLOPs. CTT is also substantially more parameter-efficient.

Limitations. First, how best to train NLAs with reinforcement learning remains an open question, and improvements to the training procedure may strengthen the GRPO baseline. Future work can also examine whether CTT improves subsequent GRPO training by providing a better initialization. Second, our NLA experiments cover Qwen3-4B and Qwen2.5-7B-Instruct, and we also evaluate CTT on CLIP's image and text encoders. Extending these evaluations to larger models, more aggressive fine-tuning, a broader range of domains, and transfer between models with different architectures would help establish CTT's broader applicability. Finally, we leave a geometric analysis of CTT to future work: how does fine-tuning change the relationship between the models' weights, in which direction do these weights move, and how closely do the learned rotations align the transferred task vectors to these directions?

AI USE STATEMENT

In this work, we used generative AI tools to help us ideate, to design and run some experiments, and to help us draft the Appendix. We have not used generative AI tools to write the main paper text. We have thoroughly reviewed all AI-assisted work, such as critically checking AI-generated ideas for legitimacy and plagiarism and verifying LLM-generated code for correctness. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Samuel K. Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git Re-Basin: Merging models modulo permutation symmetries, 2023. URL https://arxiv.org/abs/2209.04836.

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. SliceGPT: Compress large language models by deleting rows and columns, 2024. URL https://arxiv.org/abs/2401.15024.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. URL https://arxiv.org/abs/1308.3432.

Arthur Cayley. Sur quelques propriétés des déterminants gauches. *Journal für die reine und angewandte Mathematik*, 32:119–123, 1846. doi: 10.1515/crll.1846.32.119. URL https://eudml.org/doc/147336.

Zhikang Chen, Sen Cui, Deheng Ye, Min Zhang, Gang Niu, Yu Zhang, Masashi Sugiyama, and Tingting Zhu. From coefficients to directions: Rethinking model merging with directional alignment, 2025. URL https://arxiv.org/abs/2512.00391.

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. doi: 10.1109/JPROC.2017.2675998.

Mehdi Cherti, Romain Beaumont, Ross Wightman, Mitchell Wortsman, Gabriel Ilharco, Cade Gordon, Christoph Schuhmann, Ludwig Schmidt, and Jenia Jitsev. Reproducible scaling laws for contrastive language-image learning, 2023. URL https://arxiv.org/abs/2212.07143.

Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3606–3613, 2014. doi: 10.1109/CVPR.2014.461. URL https://www.robots.ox.ac.uk/~vgg/data/dtd/.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423/.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations, 2023. URL https://arxiv.org/abs/2305.14233.

Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks, 2022. URL https://arxiv.org/abs/2110.06296.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis, 2020. URL https://arxiv.org/abs/1912.05671.

Kit Fraser-Taliente, Subhash Kantamneni, Euan Ong, Dan Mossing, Christina Lu, Paul C. Bogdan, Emmanuel Ameisen, James Chen, Dzmitry Kishylau, Adam Pearce, Julius Tarng, Alex Wu, Jeff Wu, Yang Zhang, Daniel M. Ziegler, Evan Hubinger, Joshua Batson, Jack Lindsey, Samuel Zimmerman, and Samuel Marks. Natural language autoencoders produce unsupervised explanations of LLM activations. *Transformer Circuits Thread*, 2026. URL https://transformer-circuits.pub/2026/nla/index.html.

Samir Yitzhak Gadre, Gabriel Ilharco, Alex Fang, Jonathan Hayase, Georgios Smyrnis, Thao Nguyen, Ryan Marten, Mitchell Wortsman, Dhruba Ghosh, Jieyu Zhang, Eyal Orgad, Rahim Entezari, Giannis Daras, Sarah Pratt, Vivek Ramanujan, Yonatan Bitton, Kalyani Marathe, Stephen Mussmann, Richard Vencu, Mehdi Cherti, Ranjay Krishna, Pang Wei Koh, Olga Saukh, Alexander Ratner, Shuran Song, Hannaneh Hajishirzi, Ali Farhadi, Romain Beaumont, Sewoong Oh, Alex Dimakis, Jenia Jitsev, Yair Carmon, Vaishaal Shankar, and Ludwig Schmidt. DataComp: In search of the next generation of multimodal datasets, 2023. URL https://arxiv.org/abs/2304.14108.

Antonio Andrea Gargiulo, Donato Crisostomi, Maria Sofia Bucarelli, Simone Scardapane, Fabrizio Silvestri, and Emanuele Rodolà. Task singular vectors: Reducing task interference in model merging. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18695–18705, 2025. URL https://openaccess.thecvf.com/content/CVPR2025/html/Gargiulo_Task_Singular_Vectors_Reducing_Task_Interference_in_Model_Merging_CVPR_2025_paper.html.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of DNNs, 2018. URL https://arxiv.org/abs/1802.10026.

Gemma Team. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645 (8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. EuroSAT: A novel dataset and deep learning benchmark for land use and land cover classification, 2019. URL https://arxiv.org/abs/1709.00029.

Kyle Helfrich, Devin Willmott, and Qiang Ye. Orthogonal recurrent neural networks with scaled Cayley transform, 2018. URL https://arxiv.org/abs/1707.09520.

Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, Dawn Song, Jacob Steinhardt, and Justin Gilmer. The many faces of robustness: A critical analysis of out-of-distribution generalization, 2021. URL https://arxiv.org/abs/2006.16241.

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, 2022. URL https://arxiv.org/abs/2106.09685.

Evan Hubinger, Nicholas Schiefer, Carson Denison, and Ethan Perez. Model organisms of misalignment: The case for a new pillar of alignment research. AI Alignment Forum, August 2023. URL https://www.alignmentforum.org/posts/ChDH335ckdvpxXaXX/model-organisms-of-misalignment-the-case-for-a-new-pillar-of-1.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic, 2023. URL https://arxiv.org/abs/2212.04089.

Moritz Imfeld, Jacopo Graldi, Marco Giordano, Thomas Hofmann, Sotiris Anagnostidis, and Sidak Pal Singh. Transformer fusion with optimal transport, 2024. URL https://arxiv.org/abs/2310.05719.

Keller Jordan, Hanie Sedghi, Olga Saukh, Rahim Entezari, and Behnam Neyshabur. REPAIR: Renormalizing permuted activations for interpolation repair, 2023. URL https://arxiv.org/abs/2211.08403.

Kazuhiko Kawamoto, Atsuhiro Endo, and Hiroshi Kera. Cross-model transfer of task vectors via few-shot orthogonal alignment, 2025. URL https://arxiv.org/abs/2505.12021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3D object representations for fine-grained categorization. In *IEEE International Conference on Computer Vision Workshops (ICCVW)*, pp. 554–561, 2013. doi: 10.1109/ICCVW.2013.77.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Mario Lezcano-Casado and David Martínez-Rubio. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group, 2019. URL https://arxiv.org/abs/1901.08428.

Tianyi Li and Zhiqiang Shen. Scaling linear mode connectivity and merging to billion parameter pretrained transformers, 2026. URL https://arxiv.org/abs/2606.23607.

Pin-Jie Lin, Rishab Balasubramanian, Fengyuan Liu, Nikhil Kandpal, and Tu Vu. Efficient model development through fine-tuning transfer, 2025. URL https://arxiv.org/abs/2503.20110.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach, 2019. URL https://arxiv.org/abs/1907.11692.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011. URL https://research.google/pubs/reading-digits-in-natural-images-with-unsupervised-feature-learning/.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Fidel A. Guerrero Peña, Heitor Rapela Medeiros, Thomas Dubail, Masih Aminbeidokhti, Eric Granger, and Marco Pedersoli. Re-basin via implicit Sinkhorn differentiation, 2023. URL https://arxiv.org/abs/2212.12042.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eighth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

Qwen Team. Qwen2.5: A party of foundation models, September 2024a. URL https://qwenlm.github.io/blog/qwen2.5/.

Qwen Team. Qwen2.5 technical report, 2024b. URL https://arxiv.org/abs/2412.15115.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

Filippo Rinaldi, Giacomo Capitani, Lorenzo Bonicelli, Donato Crisostomi, Federico Bolelli, Elisa Ficarra, Emanuele Rodolà, Simone Calderara, and Angelo Porrello. Update your transformer to the latest release: Re-basin of task vectors, 2025. URL https://arxiv.org/abs/2505.22697.

Filippo Rinaldi, Aniello Panariello, Giacomo Salici, Fengyuan Liu, Marco Ciccone, Angelo Porrello, and Simone Calderara. Gradient-sign masking for task vector transport across pre-trained models, 2026a. URL https://arxiv.org/abs/2510.09658.

Filippo Rinaldi, Aniello Panariello, Giacomo Salici, Angelo Porrello, and Simone Calderara. Transporting task vectors across different architectures without training, 2026b. URL https://arxiv.org/abs/2602.12952.

Peter H. Schönemann. A generalized solution of the orthogonal Procrustes problem. *Psychometrika*, 31(1):1–10, 1966. doi: 10.1007/BF02289451. URL https://doi.org/10.1007/BF02289451.

Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. LAION-5B: An open large-scale dataset for training next generation image-text models. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 35, pp. 25278–25294, 2022. doi: 10.52202/068431-1833.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Abhay Sheshadri, Aidan Ewart, Kai Fronsdal, Isha Gupta, Samuel R. Bowman, Sara Price, Samuel Marks, and Rowan Wang. AuditBench: Evaluating alignment auditing techniques on models with hidden behaviors, 2026. URL https://arxiv.org/abs/2602.22755.

Sidak Pal Singh and Martin Jaggi. Model fusion via optimal transport, 2020. URL https://arxiv.org/abs/1910.05653.

Jungyong Son, Jinwook Jung, Minhee Park, and Sungyong Baik. Bilinear coordinate alignment for training-free task-vector transfer, 2026. URL https://arxiv.org/abs/2605.28444.

Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, 2012. doi: 10.1016/j.neunet.2012.02.016. URL https://doi.org/10.1016/j.neunet.2012.02.016.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding, 2021. URL https://arxiv.org/abs/2104.09864.

Xinyi Sui, Ziran Liu, Nam Ling, Wei Wang, and Wei Jiang. CORAM: Coherent orthogonal rotation for model merging, 2026. URL https://arxiv.org/abs/2608.17366.

Alexander Theus, Alessandro Cabodi, Sotiris Anagnostidis, Antonio Orvieto, Sidak Pal Singh, and Valentina Boeva. Generalized linear mode connectivity for transformers, 2025. URL https://arxiv.org/abs/2506.22712.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding, 2019. URL https://arxiv.org/abs/1804.07461.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with OSS-Instruct, 2024. URL https://arxiv.org/abs/2312.02120.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time, 2022. URL https://arxiv.org/abs/2203.05482.

Jianxiong Xiao, James Hays, Krista A. Ehinger, Aude Oliva, and Antonio Torralba. SUN database: Large-scale scene recognition from abbey to zoo. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3485–3492, 2010. doi: 10.1109/CVPR.2010.5539970.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A. Raffel, and Mohit Bansal. TIES-merging: Resolving interference when merging models. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1644c9af28ab7916874f6fd6228a9bcf-Abstract-Conference.html.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

Sihan Yang, Kexuan Shi, and Weiyang Liu. Orthogonal model merging, 2026. URL https://arxiv.org/abs/2602.05943.

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 *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 57755–57775. PMLR, 2024. URL https://proceedings.mlr.press/v235/yu24p.html.

Z.ai. GLM-5.2. Hugging Face model card, 2026. URL https://huggingface.co/zai-org/GLM-5.2.

Biao Zhang and Rico Sennrich. Root mean square layer normalization, 2019. URL https://arxiv.org/abs/1910.07467.

Binchi Zhang, Zaiyi Zheng, Zhengzhang Chen, and Jundong Li. Beyond the permutation symmetry of transformers: The role of rotation for model fusion. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 77090–77106. PMLR, 2025. URL https://proceedings.mlr.press/v267/zhang25dk.html.

A BACKGROUND

This appendix collects the theory behind section 3 and our comparison of transfer methods. We first review re-basin and linear mode connectivity (section A.1) and the weight symmetries of a pre-norm transformer (section A.2). We then define reparameterization symmetries and show which transfer methods respect them (section A.3). Next, we describe each transfer method we compare (section A.4). Finally, we explain how CTT parameterizes its rotation (section A.5) and when a residual-stream rotation can be folded into the neighboring weights (section A.6).

A.1 RE-BASIN AND LINEAR MODE CONNECTIVITY

Consider two networks A and B with the same architecture and weights Θ_A and Θ_B, trained independently, and suppose we want to combine them by interpolating in weight space, $\Theta(\lambda) = (1-\lambda)\Theta_A + \lambda\Theta_B$ for $\lambda \in [0,1]$. Following Entezari et al. (2022), the *loss barrier* on a dataset $\mathcal{D}$ is the largest excess of the interpolated loss over the linear interpolation of the endpoint losses,

$$\mathcal{B}[\Theta_A, \Theta_B](\mathcal{D}) = \sup_{\lambda\in[0,1]} \Big\{ \mathcal{L}\big[(1-\lambda)\Theta_A + \lambda\Theta_B\big](\mathcal{D}) - \big((1-\lambda)\mathcal{L}_A + \lambda\mathcal{L}_B\big) \Big\}.$$

In Frankle et al. (2020), the baseline is the mean $\frac{1}{2}(\mathcal{L}_A + \mathcal{L}_B)$, and two networks are *linearly mode connected* when the barrier is close to zero. For independently trained networks, the barrier is usually large, and the loss peaks near the middle of the segment. Garipov et al. (2018) showed that the two minima are joined by a *curved* path of low loss. However, such a path has to be trained and does not give a single merged model.

Later, Entezari et al. (2022) conjectured that the barrier on the straight segment comes from permutation symmetry. Reordering the hidden units of a layer, together with the matching rows and columns of its neighbors, leaves the function unchanged. Each trained network is therefore one member of a large equivalence class, and two networks that look far apart may have members that are close. Building on this, Ainsworth et al. (2023) gave algorithms that find such a permutation π and showed that the barrier between Θ_A and $\pi(\Theta_B)$ falls to zero on MNIST and on sufficiently wide CIFAR-10 ResNets, while ImageNet-scale models keep a smaller but positive barrier. Merging Θ_A with $\pi(\Theta_B)$ is called *re-basining* B into A's basin.

Because a symmetry satisfies $f[\pi(\Theta)] = f[\Theta]$ on every input, re-basin changes where the merge happens and leaves each endpoint's accuracy untouched. Task-vector transfer uses the same idea in a different setting, where instead of interpolating two models we apply π to the task vector τ and add it to the recipient θ_B (section 3). Of the methods we compare, Git Re-Basin, TransFusion and CTT respect reparameterization symmetries, whereas GradFix, THESEUS, BiCo and Kawamoto do not (section A.3).

A.2 SYMMETRIES OF A PRE-NORM TRANSFORMER

We start with the groups involved. The *orthogonal group* $\mathrm{O}(d) = \{R \in \mathbb{R}^{d\times d} : R^\top R = I\}$ preserves inner products, and hence lengths and angles. Its determinant is ± 1. It has two connected components, of which the one that contains the identity is the *special orthogonal group* $\mathrm{SO}(d) = \{R : R^\top R = I,\ \det R = +1\}$ of rotations, and the other holds the orientation-reversing maps such as reflections. $\mathrm{SO}(d)$ is a smooth manifold of dimension $d(d-1)/2$, and its tangent space at the identity is the space of skew-symmetric matrices $\mathfrak{so}(d) = \{S : S^\top = -S\}$. In addition, the *permutation matrices* Π_d form a finite subgroup of $\mathrm{O}(d)$ of order $d!$, and since $\det P = \mathrm{sgn}(\pi)$, only the even permutations lie in $\mathrm{SO}(d)$.

These facts shape the methods we compare in three ways. First, Π_d is a set of isolated points with no tangent space, so a loss has no gradient with respect to P and a permutation has to be found by a combinatorial solver or through a relaxation. Second, the convex hull of Π_d is the *Birkhoff polytope* of doubly stochastic matrices, $\mathcal{B}_d = \{P \geq 0 : P\mathbf{1} = \mathbf{1},\ P^\top\mathbf{1} = \mathbf{1}\}$, whose vertices are the permutation matrices. A linear objective over $\mathcal{B}_d$ attains its optimum at a vertex, so the linear assignment problem is solvable in polynomial time. Third, the rotations form a continuum with $d(d-1)/2$ degrees of freedom ($2560 \cdot 2559/2 = 3{,}275{,}520$ at the Qwen3-4B width and $3584 \cdot 3583/2 = 6{,}420{,}736$ at the Qwen2.5-7B-Instruct width). As a result, the permutations form

a finite set of measure zero in $\mathrm{O}(d)$, whereas CTT optimizes over the continuous group by gradient descent.

A pre-norm transformer keeps a residual stream $X_\ell \in \mathbb{R}^d$ at each position and updates it as $X_{\ell+1} = X_\ell + F_\ell(\mathrm{Norm}(X_\ell))$. Here F_ℓ is an attention or feed-forward block. Norm is RMSNorm (Zhang & Sennrich, 2019) in Qwen3 and Qwen2.5, and LayerNorm (Ba et al., 2016) in CLIP. We write the gain-free RMSNorm as $n(x) = x/\sqrt{\frac{1}{d}\|x\|_2^2 + \varepsilon}$ and the full one as $\mathrm{RMSNorm}_g(x) = g \odot n(x)$. Because $\|Rx\| = \|x\|$ for every $R \in \mathrm{O}(d)$, the gain-free norm is *equivariant*, $n(Rx) = R\,n(x)$, but the gain is not, since $g \odot Rn(x) \neq R(g \odot n(x))$ unless $\mathrm{diag}(g)$ commutes with R. We remove this obstruction by absorbing every gain into the linear map that reads it, $W_{\mathrm{read}} \leftarrow W_{\mathrm{read}}\,\mathrm{diag}(g)$ and $g \leftarrow \mathbf{1}$. The fold does not change the function, because $W(g \odot n(x)) = (W\mathrm{diag}(g))\,n(x)$. After the fold, the map $x \mapsto Rx$ on the stream is a symmetry under

$$W_{\mathrm{read}} \leftarrow W_{\mathrm{read}}R^\top, \quad W_{\mathrm{write}} \leftarrow R\,W_{\mathrm{write}}, \quad b_{\mathrm{write}} \leftarrow R\,b_{\mathrm{write}}, \quad E \leftarrow ER^\top, \quad U \leftarrow UR^\top. \tag{A.1}$$

The readers are the query, key, value and feed-forward input projections, the writers are the attention output and feed-forward output projections, E holds the token embeddings as rows, and U is the unembedding. This rule is the "computational invariance" of Ashkboos et al. (2024), which Theus et al. (2025) use to align transformers, and CTT applies it to the task vector (eq. (1) and the General Algorithm & Reparameterization Symmetries subsection of Section 4). For RMSNorm, the symmetry group is all of $\mathrm{O}(d)$, and we restrict CTT to $\mathrm{SO}(d)$ through its parameterization (section A.5), which leaves out the reflections. Section A.6 gives the full folding conditions, including the LayerNorm case.

Given activations $Z^{(A)}, Z^{(B)} \in \mathbb{R}^{n\times d}$ of the two models at matched positions, the orthogonal map that best aligns them in the least-squares sense solves the orthogonal Procrustes problem (Schönemann, 1966),

$$G^\star = \operatorname*{arg\,min}_{G^\top G = I} \left\| Z^{(A)} - Z^{(B)}G \right\|_F^2 = \operatorname*{arg\,max}_{G^\top G = I} \mathrm{tr}\big(G^\top Z^{(B)\top} Z^{(A)}\big). \tag{A.2}$$

The second form holds because G preserves the Frobenius norm. With the singular value decomposition $Z^{(B)\top}Z^{(A)} = U_Z \Sigma V_Z^\top$, the maximum is $\sum_i \sigma_i$, attained at $G^\star = U_Z V_Z^\top$, and THESEUS and BiCo compute solutions of this form from activations and gradients (section A.4), whereas CTT fits its rotation directly to the transfer objective (eq. (2)).

Next, we turn to attention, with weights acting on column vectors, $y = Wx$. An attention block with H heads forms, for each head, queries $q_i = W_i^Q x$, keys $k_i = W_i^K x$ and values $v_i = W_i^V x$ from the normalized stream, attends with $\mathrm{softmax}(q_i^\top k_i/\sqrt{d_k})$ over positions, and writes W_i^O times the attention-weighted values back. The products $(W_i^Q)^\top W_i^K$ and $W_i^O W_i^V$ are the head's QK and OV circuits. The sum over heads does not depend on their order, so any $P \in S_H$ applied to the head blocks of W^Q, W^K, W^V and to the matching blocks of W^O is a symmetry. Inside a head the symmetry is larger than a permutation, since the maps $W_i^Q \to MW_i^Q$, $W_i^K \to M^{-\top}W_i^K$, $W_i^V \to NW_i^V$ and $W_i^O \to W_i^O N^{-1}$ preserve the function for any invertible M and N. TransFusion shares one permutation across the query, key and value rows of a head, and this choice ($M = N = P$) is a small subgroup of these maps.

TransFusion (Rinaldi et al., 2025) aligns attention on two levels. Writing $h_{i,m}$ for head i's block of projection $m \in \{q, k, v, o\}$, its *inter-head* matching compares heads by the Euclidean distance between their sorted singular values, $D_{ij} = \sum_m \|\sigma(h_{i,m}^B) - \sigma(h_{j,m}^A)\|_2$, and solves

$$P_{\mathrm{inter}} = \operatorname*{arg\,min}_{P\in S_H} \sum_{i=1}^{H} D_{i,P[i]}. \tag{A.3}$$

Because singular values do not change under orthogonal maps on either side of a block, this cost does not depend on the order of rows inside a head or on a residual-stream rotation. *Intra-head* matching then permutes the d_k rows of each matched pair,

$$P_{\mathrm{intra}}^{(i)} = \operatorname*{arg\,max}_{P\in S_{d_k}} \left\langle h_i^B,\ P\,h^A_{P_{\mathrm{inter}}[i]} \right\rangle_F, \tag{A.4}$$

which is a linear assignment problem (LAP). The block's permutation is the composition $P_{\ell\text{-attn}} = \mathrm{blockdiag}(P_{\mathrm{intra}}^{(1)}, \ldots, P_{\mathrm{intra}}^{(H)}) \cdot (P_{\mathrm{inter}} \otimes I_{d_k})$, which acts on the rows of the query, key and value blocks and on the matching columns of the output block. Its block structure keeps it function-preserving, because a generic permutation of all Hd_k rows would mix rows across heads. Note that eq. (A.4) and eq. (A.2) are different problems, since the first optimizes over the finite set Π_{d_k} and needs a combinatorial solver, while the second optimizes over $\mathrm{O}(d)$ and has a closed form.

In Qwen3 and Qwen2.5, two architectural features narrow the attention symmetries further. First, rotary position embeddings (Su et al., 2021) apply a block-diagonal rotation Ω_m with distinct frequencies to the query and key vectors inside each head, so the score is $q^\top \Omega_{n-m} k$. A reparameterization $q \to Aq$, $k \to Bk$ preserves the score for every offset only if $B = A^{-\top}$ (take the offset zero) and $A^{-\top}$ commutes with every Ω_t. Since the identity is the only permutation that commutes with every Ω_t, the intra-head permutation of eq. (A.4) is not a symmetry of these models, and our TransFusion port drops it. Second, grouped-query attention shares each key/value head among several query heads (Qwen3-4B has 32 query heads and 8 key/value heads) (Qwen Team, 2025; 2024b). The head symmetry is then a permutation of query heads within a group, composed with a permutation of the groups that carries each group's key/value head along (section B.8 describes our ports).

Finally, the feed-forward block $\mathrm{FF}(x) = W_2\, \phi(W_1 x + b_1) + b_2$ applies its nonlinearity coordinate-wise. For a generic nonlinearity, the invertible linear maps that commute with it are the permutations. Positively homogeneous ones such as ReLU also allow positive diagonal scalings, and odd ones allow sign flips. The hidden units thus admit a permutation symmetry, $W_1 \to PW_1$, $b_1 \to Pb_1$, $W_2 \to W_2 P^\top$, and not a rotation. Qwen's gated SwiGLU, $W_2\,(\mathrm{silu}(W_g x) \odot W_u x)$, allows a permutation of its hidden index shared by the gate and up projections, together with an invertible diagonal rescaling of the up projection that the down projection undoes. In every case the hidden symmetries are monomial (a permutation times a diagonal), never a dense rotation. With the residual map held fixed, matching the hidden units of one layer between two models is a LAP, but in a plain MLP the permutations of adjacent layers enter one objective. Git Re-Basin (Ainsworth et al., 2023) calls this the *sum of bilinear assignments problem*, shows it to be NP-hard, and solves it by coordinate descent over the permutation groups (section A.4). In a pre-norm transformer with a single residual map, the feed-forward problems of different layers decouple once that map is fixed.

Weight matching finds the alignment that brings Θ_B closest to Θ_A, but this need not be the alignment under which the merge works best. To close this gap, Theus et al. (2025) introduce *learned matching*, which optimizes the alignment against the loss of the interpolated model,

$$\mathcal{J} = \frac{1}{|\mathcal{S}|} \sum_{(X,Y)\in\mathcal{S}} \mathcal{L}_{\mathrm{CE}}\big(\Theta_{\mathrm{interp}}; X, Y\big), \quad \Theta_{\mathrm{interp}} = (1-\lambda)\,\Theta_A + \lambda\,\pi(\Theta_B), \tag{A.5}$$

over batches $\mathcal{S}$, with λ drawn at random near the midpoint of the segment. They keep one latent matrix per symmetry, initialize it from weight matching, and take the residual orthogonal map to be the polar factor $U_L V_L^\top$ of its latent $L = U_L \Sigma_L V_L^\top$, which is differentiable in the latent. The permutations are projected by the Hungarian algorithm in the forward pass and receive the straight-through gradient (Bengio et al., 2013) in the backward pass. On a ViT trained on CIFAR-10 and a small GPT-2-style model trained on BookCorpus, learned matching lowers the loss barrier well below their weight matching (which also fits an orthogonal residual map). The authors also find that the permutations from weight matching stay essentially unchanged under learned matching, and that the improvement concentrates in the orthogonal map on the residual stream. CTT keeps only this orthogonal map, adds a scale, and learns both against the transfer objective (eq. (2)).

A.3 Reparameterization symmetries and which methods respect them

Here we make precise what it means for a task-vector transfer method to respect reparameterization symmetries. We first state the definition, and we then prove that CTT satisfies it for rotations of the residual stream and that Git Re-Basin and TransFusion satisfy it for permutations. Finally, we show where GradFix, THESEUS, BiCo and Kawamoto fail it.

A.3.1 The definition

We write $f(x;\theta)$ for the function a network computes, with its parameters $\theta = (\theta_k)_{k\in\mathcal{K}}$ collected tensor by tensor. A reparameterization symmetry is an invertible linear map ρ_g on parameters,

indexed by an element g of a group H, such that

$$f(x;\rho_g\theta) = f(x;\theta) \text{ for every input } x, \qquad \rho_g\rho_{g'} = \rho_{gg'}. \tag{A.6}$$

All the symmetries we consider act on each tensor separately, and because ρ_g is linear, reparameterizing a source pair reparameterizes its task vector in the same way, $\rho_g\theta_A^{\mathrm{ft}} - \rho_g\theta_A = \rho_g\tau$. A transfer method of the form given in section 3 considers a set Φ of candidates $\varphi = (\pi, \alpha)$. Each candidate gives a transferred model $\theta'_\varphi = \theta_B + \alpha\,\pi(\tau)$, and the method selects one candidate by a score computed from the three models and the method's data.

Definition A.3.1 *Let H act on the source and on the recipient by ρ. We say that a transfer method respects H if it satisfies the following two conditions.*

(i) The transport the method returns is a symmetry acting on the task vector, that is, the selected candidate (π, α) has $\pi(\tau) = \rho_g\tau$ for some $g \in H$.

(ii) The method is equivariant. For any $g, h \in H$, we replace the source pair by $(\rho_g\theta_A, \rho_g\theta_A^{\mathrm{ft}})$ and the recipient by $\rho_h\theta_B$, with the data unchanged. We then require a bijection $\varphi \mapsto \varphi'$ from the candidates of the original problem to those of the new one such that $\theta'_{\varphi'} = \rho_h\theta'_\varphi$ and such that the two candidates receive equal scores.

When the recipient consists of several networks that exchange only inputs and outputs, such as the AV and AR of an NLA, H acts on each network separately and both conditions are required for each of them.

Condition (i) says that the vector added to θ_B is the task vector of the pair $(\rho_g\theta_A, \rho_g\theta_A^{\mathrm{ft}})$, which, by eq. (A.6), computes the same two functions as the source pair. The recipient therefore receives what fine-tuning changed in the source, written in another basis. Condition (ii) says that the outcome does not depend on which of the equivalent parameterizations of the source and the recipient we are given. In particular, for an exact optimizer and a unique best-scoring candidate, the reparameterized problem selects $\rho_h\theta'$, which computes the same function as θ'. Condition (ii) alone does not imply (i). For instance, a method that discards τ satisfies (ii) and fails (i).

We note that the definition concerns the problem a method solves (its candidates and its score), not the path an iterative solver takes to its solution.

A.3.2 The residual-stream rotation

We work in the completed frame of section A.6, in which every normalization gain (and, for CLIP, every LayerNorm bias and the centering) is folded into neighboring weights. CTT forms τ and applies the transfer in this frame. For $R \in \mathrm{O}(d)$, we let ρ_R act on each tensor by eq. (A.1), according to its role with respect to the residual stream, and as the identity on every other tensor. The readers also include the maps that read the final stream (the unembedding, the output projection of a CLIP tower and the read-out of the AR). The writers also include every explicit write of an input into the stream. One such write is the AV's injection of the activation, whose matrix (the identity) is fixed and belongs to the recipient only, so it never carries τ. We call the remaining tensors axisless, because none of their axes is a residual axis. They include the biases of the query, key, value and feed-forward input projections, the per-head query and key norms of Qwen3, and the patch, class and positional embeddings of the CLIP vision frame. These embeddings enter the stream only through the explicit write that replaces the pre-norm (section A.6). Since each rule multiplies one axis of one tensor by R or $R^\top$, the map ρ_R is linear, preserves the Frobenius norm of every tensor, and satisfies $\rho_R\rho_{R'} = \rho_{RR'}$.

Lemma A.3.2 *In the completed frame, $f(x;\rho_R\theta) = f(x;\theta)$ for every $R \in \mathrm{O}(d)$ and every input x.*

Proof. Let x_ℓ be the residual stream after ℓ sublayers under θ, and x'_ℓ the stream under $\rho_R\theta$. The embeddings and the explicit writes give $x'_0 = Rx_0$. Now suppose that $x'_\ell = Rx_\ell$. In the frame, every read of the stream applies an affine map, whose bias lies on a non-residual axis, to $\nu(x_\ell)$, where ν is either the identity or the gain-free normalization $n(x) = x/\sqrt{\|x\|_2^2/d + \varepsilon}$. Because $\|Rx\|_2 = \|x\|_2$, we have $\nu(Rx) = R\,\nu(x)$, and therefore $WR^\top\nu(Rx_\ell) + c = W\nu(x_\ell) + c$. Everything between a read and a write (attention with its rotary embedding and head norms, or the

feed-forward nonlinearity) acts on non-residual axes and receives the same inputs. The write adds $R(W_{\text{out}}z + b) = Ru_\ell$, so $x'_{\ell+1} = Rx_\ell + Ru_\ell = Rx_{\ell+1}$. The final readers remove R in the same way, so the outputs agree. □

A.3.3 CTT

Proposition A.3.3 *If $H = \mathrm{SO}(d)$ acts by eq.* (A.1)*, then CTT respects H on a CLIP tower. On an NLA, CTT respects H acting on the source and $H \times H$ acting on the AV and the AR.*

Proof. We treat one component $c \in \mathcal{C}$ at a time. Each component has a single rotation shared by all of its blocks (eq. (1)), and $\mathcal{C}$ holds the single tower on CLIP and $\mathcal{C} = \{\mathrm{av}, \mathrm{ar}\}$ on an NLA. We write a candidate as $\varphi_c = (R_c, \alpha_c) \in \mathrm{SO}(d) \times (0, \infty)$. By eq. (1), it gives $\theta'_c(\varphi_c) = \theta_{\mathrm{rec},c} + \alpha_c \mathcal{T}_c(R_c)[\tau_c]$.

First, we show that $\mathcal{T}_c(R)$ is ρ_R applied to τ_c. It acts as $\tau_{\mathrm{read}} R^\top$ on readers and $R\tau_{\mathrm{write}}$ on writers, and the embedding rows rotate as writes into the stream. CTT leaves the axis-less tensors unrotated and scales them by α_c with the rest of τ_c. This agrees with eq. (A.1), because ρ_R acts on the axis-less tensors as the identity and the scalar α_c commutes with ρ_R. On an NLA, the AV's task vector is all of τ and the AR's is its restriction to the blocks and the embedding that the AR keeps, and this restriction commutes with ρ_R because ρ_R acts tensor by tensor. Therefore component c receives $\alpha_c \, \rho_{R_c} \tau_c$, the scaled task vector of the source pair reparameterized by R_c, and condition (i) holds with $g = R_c$.

Next, let the source be reparameterized by $g \in \mathrm{SO}(d)$ and component c of the recipient by $h_c \in \mathrm{SO}(d)$. The new task vector is $\rho_g \tau$. We map each candidate (R_c, α_c) to $(h_c R_c g^\top, \alpha_c)$, which is a bijection of $\mathrm{SO}(d) \times (0, \infty)$, with inverse $(R', \alpha) \mapsto (h_c^\top R' g, \alpha)$. The image candidate gives

$$\rho_{h_c} \theta_{\mathrm{rec},c} + \alpha_c \, \rho_{h_c R_c g^\top} \, \rho_g \tau_c = \rho_{h_c} \theta_{\mathrm{rec},c} + \alpha_c \, \rho_{h_c} \rho_{R_c} \tau_c = \rho_{h_c} \, \theta'_c(\varphi_c), \tag{A.7}$$

where the first step uses $\rho_{g^\top} \rho_g = \rho_I$ and the second the linearity of ρ_{h_c}.

Finally, corresponding candidates receive equal scores. The fitting loss $\mathcal{L}_{\mathrm{app}}$ of eq. (2) and the holdout score $\mathcal{M}_{\mathrm{app}}$ of algorithm 1 depend on the transferred model only through its outputs. On CLIP, these are the class probabilities computed from the tower's output embedding. On an NLA, they are the AV's tokens and the AR's reconstruction $\hat{x}$ in eq. (4), and the straight-through bridge changes only the gradient and never the forward value. By Lemma A.3.2, each component computes the same function after ρ_{h_c}, and the AV and AR exchange only text, so every output is unchanged. Since the fitting data and the holdout are the same in both problems, the loss and the holdout score of $(h_c R_c g^\top, \alpha_c)$ in the new problem equal those of (R_c, α_c) in the original, and condition (ii) holds. □

The Cayley map of section A.5 reaches the rotations that do not have -1 as an eigenvalue. This set is open and dense in $\mathrm{SO}(d)$, and its complement has measure zero, so the correspondence of eq. (A.7) holds for every Cayley parameter outside a null set. Additionally, Lemma A.3.2 holds for all of $\mathrm{O}(d)$, and because $hRg^\top \in \mathrm{SO}(d)$ whenever $\det g = \det h$, the proposition extends to every such pair (g, h), including such pairs of residual permutations.

A.3.4 Git Re-Basin and TransFusion

We consider permutation matchers that search a group P of permutation symmetries and transfer at $\alpha = 1$. The group has one permutation of the residual stream shared by all blocks, one permutation of the hidden units of every feed-forward block, and permutations of attention heads (on Qwen, of the query heads within each key/value group and of the groups). Each factor is a symmetry. A residual permutation is an orthogonal R in eq. (A.1), a permutation of hidden units commutes with the coordinatewise nonlinearity, and a permutation of heads reorders the sum over heads (section A.2). TransFusion adds a permutation inside each head, shared by the head's query, key and value rows and by the matching columns of the output projection. This is a symmetry of CLIP, while rotary position embeddings rule it out on Qwen (section A.2). Every ρ_π with $\pi \in P$ permutes the entries of each tensor, so $\langle \rho_\pi X, \rho_\pi Y \rangle_F = \langle X, Y \rangle_F$ for any two tensors of the same shape.

Proposition A.3.4 *Let a matcher search the group P with a score built from Frobenius inner products $\langle \theta_{B,k}, (\rho_\pi \theta_A)_k \rangle_F$ and from the singular values of attention head blocks, and let it return $\theta_B + \rho_\pi \tau$. Then the matcher respects P.*

Proof. Condition (i) holds with $g = \pi$. For (ii), we let the source be reparameterized by $\sigma \in P$ and the recipient by $\eta \in P$, so that the inner-product terms of the score become

$$\sum_k \left\langle (\rho_\eta \theta_B)_k, (\rho_\pi \rho_\sigma \theta_A)_k \right\rangle_F = \sum_k \left\langle \theta_{B,k}, (\rho_{\eta^{-1}\pi\sigma} \theta_A)_k \right\rangle_F. \tag{A.8}$$

Permuting the rows and columns of a head block leaves its singular values unchanged, and a permutation of heads only reindexes the head blocks, so the head cost transforms in the same way. Hence the map $\pi \mapsto \eta\pi\sigma^{-1}$, a bijection of P, gives each candidate the score of its preimage, and the image candidate transfers $\rho_\eta \theta_B + \rho_{\eta\pi\sigma^{-1}} \rho_\sigma \tau = \rho_\eta(\theta_B + \rho_\pi \tau)$, which is condition (ii). □

Git Re-Basin and TransFusion match weights with scores of this kind.

A.3.5 GradFix, THESEUS, BiCo and Kawamoto

We use three facts to show that condition (i) fails. First, every symmetry above acts on each tensor by an invertible map, so it sends a nonzero tensor to a nonzero tensor, and the orthogonal ones preserve its Frobenius norm. Second, a symmetry maps every write into the stream and every later read of it by the same matrix, because the skip connections add all writes into one stream (Lemma A.3.2). Third, the symmetries of the hidden axis of a feed-forward block are monomial matrices (section A.2), so no dense orthogonal map on that axis is a symmetry.

Proposition A.3.5 *GradFix, THESEUS, BiCo and Kawamoto do not respect reparameterization symmetries. For GradFix, THESEUS and BiCo, condition (i) fails for a task vector in general position, and for Kawamoto it fails for the maps it fits, whichever group of the symmetries above we take for H.*

Proof. We take the four methods in turn, using the mechanisms described in section A.4.

- GradFix returns $\theta_B + \alpha\, m \odot \tau$, where $m_i = 1$ if $\operatorname{sign} \tau_i = \hat{s}_i$ and $m_i = 0$ otherwise, and $\hat{s}$ is the sign vote of the recipient's negative per-example gradients. The mask is computed entry by entry in the recipient's basis, so it lowers a tensor's norm whenever it drops a nonzero entry of that tensor. No orthogonal symmetry can do this, and (i) fails.
- THESEUS maps each tensor as $\tau \mapsto T_{\text{out}}^\top \tau\, T_{\text{in}}$, with a separate Procrustes map at every activation site, each computed from that site's own cross-covariance between the two models. A sublayer's write into the stream is mapped by the output map of that sublayer, and the next read of the stream by the input map of the next sublayer. These are polar factors of two different cross-covariances (one from the sublayer outputs, the other from the normalized stream), so in general they differ. The transported vector then writes in one basis and reads in another, which contradicts the second fact. THESEUS also applies a dense orthogonal map to the hidden axis of each feed-forward block, which contradicts the third. In addition, it sets τ to zero on the embeddings, although the EuroSAT task vector is nonzero on the patch, class and positional embeddings, and by the first fact no symmetry sends them to zero.
- BiCo maps each linear layer as $\tau \mapsto R_{\text{out}}^\top \tau R_{\text{in}}$, with maps fitted per linear layer, where R_{in} is the polar factor of the cross-product of the two models' input activations, and R_{out} that of their output gradients. A write into the stream is thus mapped by a rotation fitted to gradients, and the next read by a rotation fitted to activations. These are polar factors of different matrices, so in general they differ, which contradicts the second fact. The output map of each feed-forward input projection is a dense orthogonal map on the hidden axis, which contradicts the third fact. BiCo also sets τ to zero on the embeddings and, on CLIP, on the LayerNorm parameters, which contradicts the first whenever the fine-tune moves them.
- Kawamoto transports its own attention-only task vector D, as $W \mapsto W + U^\top D U$ on each attention projection of CLIP and as $W \mapsto W + U_{\text{out}}^\top D U_{\text{in}}$ on each attention projection of an NLA. Every projection has its own U, initialized to the identity and trained on its own gradient, so the fitted maps differ from each other and from the identity. The four projections of a block therefore read and write the residual stream through different maps, and so do different blocks, which contradicts the second fact. Finally, U acts densely on the

head axis of each projection and mixes heads, whereas the head symmetries of section A.2 keep the rows of each head together. □

In tables 1 to 3, "symmetric" marks the methods that respect reparameterization symmetries (Git Re-Basin, TransFusion and CTT), "non-symmetric" those that do not, and "–" the rows that apply no transport. We test condition (ii) for a rotation of the recipient on the CLIP image tower in section C.5.

A.4 THE TRANSFER METHODS WE COMPARE

We compare CTT against seven methods in tables 1 to 3, and below we describe them as their original papers and released code define them. We write each method in the form given in section 3, $\theta'_B = \theta_B + \alpha\,\pi(\tau)$, and for each one we say what π is computed from, which data it uses and how its authors set α. We give the details of how we ran each method on CLIP and on the autoencoder in section B.8, and we discuss which methods respect reparameterization symmetries in section A.3.

The naive transfer applies task arithmetic (Ilharco et al., 2023) across models. It adds the source task vector to the recipient unchanged, so π is the identity and no data is used. TransFusion and BiCo report the naive transfer at $\alpha = 1$, and we follow them in every naive row. GradFix and THESEUS report it at an α selected on validation data. Note that the naive transfer treats the coordinates of the two models as the same, even though each pre-trained model has its own basis.

Git Re-Basin (Ainsworth et al., 2023) aligns two independently trained networks by permuting the hidden units of one of them. We use its weight-matching variant, which uses only the two sets of weights. Weight matching assigns one permutation to each group of hidden units (for instance, the hidden units of one feed-forward layer or the residual stream) and chooses the permutations to maximize the summed inner product between the recipient's weights and the permuted source weights. Most weight matrices are permuted on both of their axes, so this joint problem is hard. Git Re-Basin therefore solves it by coordinate descent. Visiting the groups in a random order, the descent holds all other permutations fixed and solves a linear assignment problem for the current group with the Hungarian algorithm. It stops once a full sweep changes no permutation. Its output is a permutation π of every weight axis of the source model, so it also acts on τ. In the original paper, π serves to merge two models by interpolating between their weights, so there is no task-vector scale. For transfer, we add $\pi(\tau)$ at $\alpha = 1$, as TransFusion does.

TransFusion (Rinaldi et al., 2025) extends weight matching to transformers and uses it to transfer task vectors between two pre-trained models. Like Git Re-Basin, it uses only the weights and matches the feed-forward and residual groups by weight matching. For attention, TransFusion uses two levels of permutation. It first pairs whole heads by a spectral cost and then permutes the dimensions within each matched pair (section A.2). TransFusion adds $\pi(\tau)$ to the recipient at $\alpha = 1$.

GradFix (Rinaldi et al., 2026a) keeps the entries of τ that agree in sign with a descent direction of the recipient and sets the rest to zero. It uses a few labeled examples of the task (a fixed number per class). For each example, it computes the gradient of the recipient's cross-entropy loss through its zero-shot head with respect to every parameter of the image encoder. Each example then votes with the sign of its negative gradient, and the majority sign gives an estimate $\hat{s}_i$ of the descent direction for parameter i. The mask m keeps τ_i when $\operatorname{sign}(\tau_i) = \hat{s}_i$, and kept entries are not rescaled. In the notation of section 3, $\pi(\tau) = m \odot \tau$, an elementwise mask chosen in the recipient's coordinates. GradFix selects α on a validation split (its released code searches $\{0.1, 0.2, \ldots, 1.0\}$).

THESEUS (Rinaldi et al., 2026b) maps τ into the recipient with one orthogonal map per activation site, fitted from the activations of the two models. Both pre-trained models run on the same unlabeled calibration inputs, and THESEUS records the activations at the input and the output of every linear layer. At each site, it accumulates the cross-covariance $H_A^\top H_B$ of the two models' activations and solves the orthogonal Procrustes problem, whose solution is $T = UV^\top$ for the singular value decomposition $H_A^\top H_B = U\Sigma V^\top$. Each layer's update is then transported as $T_{\text{out}}^\top \tau\, T_{\text{in}}$, using the maps at that layer's output and input. Because T may be rectangular (with orthonormal columns), the same construction also transfers between models of different widths. The paper leaves the positional and patch embeddings out of the transfer. It selects α per task by a linear search on validation data (the released configuration steps by 0.1 from 0.1 up to at most 5.0 and stops after eleven consecutive worse values).

BiCo (Son et al., 2026) also fits one pair of orthogonal maps per layer. Because the gradient of a linear layer's weight is the bilinear product $G^\top X$ of the output gradients G and the input activations X, BiCo aligns the input side of each layer by activations and the output side by gradients. Both pre-trained models run forward and backward on the same small labeled calibration set, and BiCo records X and $G = \partial\mathcal{L}/\partial y$ at every transferred linear layer. Two Procrustes problems per layer then give R_{in} from $X_A^\top X_B$ and R_{out} from $G_A^\top G_B$. BiCo transports the update of every attention and feed-forward weight matrix as $R_{\text{out}}^\top \tau\, R_{\text{in}}$ and adds it at $\alpha = 1$ with no search.

Kawamoto et al. (2025) learn one map per attention projection by gradient descent on the task loss. Their task vector changes only the attention weights, because they fine-tune the source with only the query, key, value and output projections trainable and take the resulting weight difference D of each projection as the task vector. For the transfer, they place a learnable matrix U, initialized to the identity, on each attention projection of every block and add $\lambda\, U^\top D\, U$ to the recipient's weight. They train the matrices U on a few labeled examples of the task, with a loss that adds a penalty on the distance of each U from the orthogonal group to the recipient's cross-entropy through its zero-shot head. The paper fixes $\lambda = 1$. Because Kawamoto transfers its own attention-only task vector, its rows use a different τ from that used in every other row, and we describe how we trained that task vector in section B.8.

All four papers behind GradFix, THESEUS, BiCo and Kawamoto evaluate on CLIP image classification. GradFix, BiCo and Kawamoto use a task loss there, which we take to be the cross-entropy through the zero-shot head. On the CLIP text encoder, we use the cross-entropy through the frozen classification head in its place, and on the autoencoder, the round-trip reconstruction error. Section B.8 gives these choices, the data each method uses and how we set α.

A.5 Parameterizing the rotation

We show here that the Cayley transform always lands in $\mathrm{SO}(d)$, explain why we chose it over other parameterizations of orthogonal matrices, and give the gradients CTT uses. For any $A \in \mathbb{R}^{d\times d}$, let $S = A - A^\top$, which is skew-symmetric, and define

$$R = \mathrm{Cay}(A) = (I - S)^{-1}(I + S) = (I + S)(I - S)^{-1}.$$

The map is defined for every real A, because the eigenvalues of a skew-symmetric matrix are purely imaginary, $i\theta$, so $I - S$ has eigenvalues $1 - i\theta \neq 0$ and is invertible. The matrix R is orthogonal because $(I - S)^{-1}$ and $I + S$ commute and $(I + S)^\top = I - S$, which gives $R^\top R = (I - S)(I + S)^{-1}(I-S)^{-1}(I+S) = I$. Its determinant is $\det(I+S)/\det\left((I+S)^\top\right) = 1$, so $\mathrm{Cay}(A) \in \mathrm{SO}(d)$ for every A. Therefore, CTT needs no projection, retraction or penalty term to keep the iterates of algorithm 1 in the group.

Each eigenvalue of R has the form $(1 + i\theta)/(1 - i\theta)$ and is never -1. Conversely, every rotation without -1 as an eigenvalue is reached, with inverse $S = (R - I)(R + I)^{-1}$, so the map $S \mapsto (I-S)^{-1}(I+S)$ is a bijection from $\mathfrak{so}(d)$ onto that set (Cayley, 1846). The rotations it misses have a half-turn in some invariant plane. They form a proper algebraic subvariety of $\mathrm{SO}(d)$ with measure zero.

The map has two further properties that matter for the fit. First, $\mathrm{Cay}(0) = I$, so algorithm 1 starts at the identity rotation. With the initial log-scale $s_c = 0$ of algorithm 1, which we use on the NLA and in the untuned CLIP row, the step-0 model is the naive transfer $\theta_{\text{rec},c} + \tau_c$. For the tuned CLIP start, we configure the same algorithm with s_c initialized at $\log \alpha_0$, so the step-0 model is the naive transfer at scale α_0 (section B.9). Second, only the skew part of A enters, so the d^2 stored entries carry $d(d-1)/2$ effective degrees of freedom and the symmetric directions receive zero gradient.

The Cayley map is also cheap. One evaluation costs a $d \times d$ linear solve with d right-hand sides, which is the same order as a matrix product. It is the $[1/1]$ Padé approximant of $\exp(2S)$ and, after rescaling S by $1/2$, a standard retraction on $\mathrm{SO}(d)$ (Lezcano-Casado & Martínez-Rubio, 2019). Its known failure mode is a diverging S as the optimum approaches a rotation with an eigenvalue at -1 (Helfrich et al., 2018; Lezcano-Casado & Martínez-Rubio, 2019).

We also considered two other ways to obtain a differentiable orthogonal matrix. The matrix exponential $\exp\colon \mathfrak{so}(d) \to \mathrm{SO}(d)$ is surjective, and Lezcano-Casado & Martínez-Rubio (2019) adopt it. However, it needs a high-degree Padé approximant with scaling and squaring, which costs several

matrix products per evaluation and again per gradient. Next, the polar projection maps a full-rank $X = U\Sigma V^\top$ to its nearest orthogonal matrix $UV^\top$ (Schönemann, 1966). It needs an SVD. It also lands in $\mathrm{O}(d)$, so it can return a reflection. Learned matching uses it for its orthogonal latent (section A.2). In CTT, we use the polar projection only as a numerical guard before saving the selected rotation (section B.6), and because that rotation is already orthogonal, this step does not change its determinant. Every CTT fit we report uses the Cayley map.

Because the transferred weight is bilinear in the rotation and the scale, the gradient with respect to (R, α) is a contraction of the ordinary weight gradient $G = \partial\mathcal{L}/\partial W'$ against the task vector. For a writer, $W' = W_B + \alpha R\tau$ gives $\partial\mathcal{L}/\partial\alpha = \langle G, R\tau\rangle$ and $\partial\mathcal{L}/\partial R = \alpha\, G\tau^\top$. For a reader, $W' = W_B + \alpha\tau R^\top$ gives $\langle G, \tau R^\top\rangle$ and $\alpha\, G^\top\tau$. A tensor with no residual axis contributes $\langle G, \tau\rangle$ to the scale gradient and nothing to the rotation gradient. We sum these contributions over tensors and carry them to the stored parameters with $\partial\mathcal{L}/\partial s = \alpha\,\partial\mathcal{L}/\partial\alpha$ and

$$\frac{\partial\mathcal{L}}{\partial S} = (I+S)^{-1}\,\frac{\partial\mathcal{L}}{\partial R}\,(I+R)^\top, \qquad \frac{\partial\mathcal{L}}{\partial A} = \frac{\partial\mathcal{L}}{\partial S} - \Big(\frac{\partial\mathcal{L}}{\partial S}\Big)^\top.$$

In our implementation, we materialize the transferred weights as plain leaves, run ordinary backpropagation, and then close the chain with one contraction per tensor and one Cayley backward per rotation. We checked this contraction against full automatic differentiation in fp64 on small synthetic problems. Note that the tensors without a residual axis cannot be left out of the scale gradient, because on CLIP their share of $\partial\mathcal{L}/\partial\alpha$ is comparable to that of the rotated tensors.

A.6 Folding rotations into neighboring weights

We first state when a residual-stream rotation preserves the function, fold the normalizations of Qwen and CLIP so that the condition holds, and close with the architectures where it fails. Let the stream be $x_0 = E(\text{input})$ and $x_{\ell+1} = x_\ell + f_\ell(x_\ell)$, with output $h(x_L)$, where each f_ℓ, as well as h, has the form $f(x) = \psi\big(W_{\text{out}}\,\varphi(W_{\text{in}}\,\nu(x))\big) + b$ for an arbitrary inner map φ. Then $x \mapsto Rx$ is a symmetry under eq. (A.1) when two conditions hold. First, every *read* of the stream must be an affine map, with its bias on a non-residual axis, composed with an R-equivariant ν (the identity, or a parameter-free RMS normalization). Second, every *write* must be a linear map, optionally followed by an R-equivariant ψ, that is added to the stream with its bias rotated. The proof is an induction on ℓ with $x'_\ell = Rx_\ell$ (Lemma A.3.2). More generally, the residual symmetry group is the commutant of every operation applied to the stream itself. For instance, a diagonal gain with distinct entries cuts it to $\{\mathrm{diag}(\pm 1)\}$, and the centering projector cuts it to $\mathrm{O}(d-1)\times\{\pm 1\}$. Permutations survive centering, but they survive a gain only when the gain is constant on every cycle.

Qwen3 and Qwen2.5 satisfy the condition once we fold every RMSNorm gain into its readers, namely each block's input norm into the query, key and value projections, each post-attention norm into the gate and up projections, and the final norm into the unembedding. We set the folded gains to one. The task vector is then zero on them, and they drop out of the transfer. Qwen3's per-head query and key norms act on the head dimension, not on the stream, so we leave them in place. We call the folded parameterization *RMS-form*. For Qwen, it is also the *completed frame* (the parameterization in which every rotation of the stream is a symmetry). We put the base model, the fine-tune, the AV and the AR in RMS-form before forming τ. Because the naive transfer and the six other baselines of section B.8 also run in RMS-form, every row of table 3 acts on the same parameterization.

CLIP's LayerNorm subtracts the mean before normalizing, $\mathrm{LN}(x) = g \odot Mx/\sqrt{\frac{1}{d}\|Mx\|^2 + \varepsilon} + b$ with $M = I - \frac{1}{d}\mathbf{1}\mathbf{1}^\top$, and M commutes with R only when R maps the all-ones vector to $\pm$ itself. However, every read of the stream in a pre-LN tower passes through M, so the mean of the stream has no effect on the output. Applied to a LayerNorm, RMS-form folds the gain into the readers and divides the bias by the gain. We instead complete the frame of CLIP with two folds. First, we center every residual writer, $W_{\text{out}} \leftarrow MW_{\text{out}}$ and $b_{\text{out}} \leftarrow Mb_{\text{out}}$, together with the embedding that initializes the stream. In the image tower, the pre-norm on the patch embeddings produces the stream itself and has no neighbor to fold into, so we replace it with a parameter-free normalization followed by an explicit write $M\,\mathrm{diag}(\gamma)$ with bias $M\beta$. In the text tower, M folds into the token and positional embeddings. After this fold, the stream stays centered, so we replace every LayerNorm by a gain-free RMS normalization followed by an affine map, which leaves the function unchanged. The completed frame is this network, and every transferred model is evaluated

in it. Second, we fold the gain *and* the bias of each pre-branch norm into the linears that read it, $W(g \odot n(x) + b) + c = (W\mathrm{diag}(g))\, n(x) + (c + Wb)$. After both folds, the 48 per-block norm tensors of the image tower carry no parameters, and its state dictionary shrinks from 224 to 176 tensors.

For the pre-branch norms, we fold the bias into the reader because the algebraically equivalent alternative, dividing the bias by the gain, is badly conditioned on CLIP. The source image tower of our benchmark pair (section B.1) has block-0 gains as small as 3.5×10^{-7} (3.0×10^{-7} in the recipient). For instance, on the EuroSAT image task vector, one bias entry sits on a channel whose source gain has magnitude 7.2×10^{-6}, and its raw difference of 1.8×10^{-4} becomes a task-vector entry of 49.21 after the division. After folding, the largest entry of the whole task vector is 0.0766.

We checked the completed frame on the benchmark pair by applying a small random Cayley rotation to each image tower and measuring the largest change in its embeddings on random inputs. On the source tower, whose largest embedding entry is 7.41, the change is 3.61 on the raw weights and 3.87 when the per-block norms are folded in RMS-form, which folds their gains and biases but not the mean-centering. Once the centering is folded as well (the completed frame), the change is only 7.2×10^{-6}. The recipient tower gives 4.18, 4.50 and 7.6×10^{-6} on a scale of 8.78.

The condition fails for any per-channel operation on the stream itself and for any stream-space bias that is not rotated. In CLIP, the attention-output and feed-forward-output biases carry τ, so they rotate together with the weights. Tied embeddings are not an obstruction, because $E \leftarrow ER^\top$ is both the writer rule for the embedding rows and the reader rule for the unembedding. However, folding the final-norm gain into the unembedding unties the two tensors, so we treat them separately.

Gemma 3 (Gemma Team, 2025) is one architecture where the condition fails, because it applies a second RMSNorm to each sub-layer's output before the residual addition, $x \leftarrow x + g_{\mathrm{post}} \odot n(W_{\mathrm{out}} z)$, with a per-channel gain $g_{\mathrm{post}} \in \mathbb{R}^d$. The normalization itself is harmless, since $n(RW_{\mathrm{out}} z) = R\, n(W_{\mathrm{out}} z)$. The obstruction is the gain, which sits between a scale-invariant normalization and the residual addition. It cannot fold backward into W_{out}, because $n(Dy) \neq D\, n(y)$ for a non-constant diagonal D, and it cannot fold forward into the readers, because the stream is a sum over layers with different gains and no single change of variable absorbs them all. A gain-free post-norm would commute with R, and a gain without a normalization would fold into W_{out}. When the post-norm gains have distinct entries, the residual symmetry group therefore shrinks to the sign flips $\{\mathrm{diag}(\pm 1)\}$. Classic post-LN transformers, which normalize the sum $x_\ell + f(x_\ell)$ with a gain, fail for the same reason.

B Experimental Setup

In this section, we describe the models, the autoencoder and the data behind the NLA results, then the CTT and baseline protocols, the CLIP benchmark, and the comparison with direct training.

B.1 Models

We trained the 4B autoencoder and the four task-domain fine-tunes of the released Qwen3-4B checkpoint (Qwen Team, 2025), which has 36 blocks, a residual width of $d = 2560$ and a vocabulary of 151,936 tokens. Each block uses grouped-query attention with 32 query heads and 8 key/value heads of dimension 128, followed by a gated MLP of width 9728.

For the 7B experiments, we used the autoencoder that the NLA authors released for Qwen2.5-7B-Instruct (Yang et al., 2024; Qwen Team, 2024a;b; Fraser-Taliente et al., 2026), which reads layer 20 of the residual stream, where the width is $d = 3584$. Its authors trained it with their own code on WildChat-1M and Ultra-FineWeb.

Our CLIP experiments use the CLIP-ViT B/16 architecture as implemented in OpenCLIP (Radford et al., 2021; Cherti et al., 2023), with 149,620,737 parameters in total (86,192,640 in the image tower and 63,428,097 on the text side, including the logit scale). Its image tower has residual width 768, and its text tower has width 512, 12 blocks and 8 heads over a causal context of 77 tokens. We take the task vector from `datacomp_xl_s13b_b90k` (θ_A), pre-trained on DataComp (Gadre et al., 2023), and transfer it to `laion2b_s34b_b88k` (θ_B), pre-trained on LAION-2B (Schuhmann et al., 2022). GradFix, THESEUS and BiCo evaluate on this pair (Rinaldi et al., 2026a;b; Son et al.,

2026), and every row of tables 1 and 2 uses it. Both the image and the text task vectors are our own fine-tunes of θ_A, one per task, as described in section B.9.

B.2 The autoencoder and its metrics

An NLA has two halves. The AV is a full copy of the base model. For the AR, we truncate the base model after the read layer ℓ and add a $d \times d$ linear read-out that is initialized to the identity, which at 4B keeps 25 of the 36 blocks and 2,918,780,160 parameters. The read layer is $\ell = 24$ at 4B (two thirds of its depth) and $\ell = 20$ at 7B (Fraser-Taliente et al., 2026). We apply every transfer to the autoencoder's weights and never to the model being explained.

An activation does not enter the AV as text. Instead, a forward hook on the input embedding overwrites the embedding of a fixed marker token with the activation before the first block runs. Beforehand, the activation is rescaled to a fixed norm, $e_{\text{marker}} \leftarrow c \cdot h_\ell / \|h_\ell\|$, with $c = 140$ at 4B and $c = 150$ at 7B. The AV therefore sees only its direction. It also receives one fixed instruction on every example, and at 4B this instruction asks for the most important features the vector represents, while the 7B autoencoder uses its authors' own instruction. For the AR, the AV's description goes inside a fixed summary template, and the AR is read out at its last non-padding position. For every transfer result, we decode greedily with at most 160 new tokens at 4B and 200 at 7B.

At every stage of training and transfer, we score the AR by the squared distance between direction-normalized vectors, $\mathrm{nrm}(v) = \sqrt{d}\, v / \|v\|_2$, as in eq. (4). Because both vectors then have the same length, this error depends only on their directions. In the autoencoder's training loss and in the GRPO reward, we take the mean over coordinates, so the error is 0 for a perfect reconstruction and 2 for orthogonal vectors. CTT's objective in eq. (4) sums over coordinates instead. On the same normalized vectors, we report the fraction of variance explained (FVE),

$$\mathrm{FVE} = 1 - \frac{\sum_i \|\,\mathrm{nrm}(\hat{x}_i) - \mathrm{nrm}(x_i)\|_2^2}{\sum_i \|\,\mathrm{nrm}(x_i) - \bar{x}\|_2^2}, \qquad \bar{x} = \tfrac{1}{n}\textstyle\sum_i \mathrm{nrm}(x_i), \tag{B.1}$$

where the mean $\bar{x}$ is taken over the evaluation set itself. An FVE of 0 is the score of always predicting that mean. FVE is unbounded below. Note that the denominator depends on the evaluation set, so FVE values scored on different sets of activations are not directly comparable.

For a transfer, we write $\mathrm{FVE}_{\mathrm{orig}}$ for the untransferred autoencoder's score and $\mathrm{FVE}_{\mathrm{sub}}$ for the transferred one, each measured on a domain's held-out activations (TASK) and on FineWeb activations (SUPP). Recovery and retention are then

$$\mathrm{REC} = \max\Big(0,\ \frac{\mathrm{FVE}_{\mathrm{sub}}^{\mathrm{dom}} - \mathrm{FVE}_{\mathrm{orig}}^{\mathrm{dom}}}{1 - \mathrm{FVE}_{\mathrm{orig}}^{\mathrm{dom}}}\Big), \qquad \mathrm{RET} = \min\Big(1,\ \max\Big(0,\ \frac{\mathrm{FVE}_{\mathrm{sub}}^{\mathrm{base}}}{\mathrm{FVE}_{\mathrm{orig}}^{\mathrm{base}}}\Big)\Big). \tag{B.2}$$

Since the denominator of recovery is the whole remaining distance to a perfect reconstruction, recovery is the fraction of what the untransferred autoencoder was missing on the domain that the transfer restores. Retention is the transferred score on FineWeb as a fraction of the untransferred one. We clip both to $[0, 1]$, so a transfer that lowers the domain FVE gets a recovery of 0 and one that raises the FineWeb FVE gets a retention of 1. The CLIP metrics are not clipped (section B.9).

In table 3, the Zero-shot row prints the absolute domain FVE under TASK, and every other row prints its change relative to that row. Each transfer row is a mean over three seeds (section B.8). Section C.1 compares these values with the same autoencoder's FVE on the base model's activations of the same documents. SUPP is the FineWeb FVE itself. Underlining marks the best REC within each class (symmetric and non-symmetric). All rows of that table are graded on the sets of section B.5, on which the untransferred FineWeb FVE is 0.6898 at 4B and 0.7467 at 7B.

B.3 Training the Qwen3-4B autoencoder

We trained the 4B autoencoder in two stages, a supervised stage and a GRPO stage, on one node of eight H100s. Four GPUs hold the AV and four hold the AR, and each half is sharded with FSDP within its own process group, so the two halves train side by side on the same node. We compute in bf16 and keep fp32 master weights.

All training activations come from the first 50,000 documents of FineWeb `sample-10BT` (Penedo et al., 2024), of which 49,982 are long enough to use. After tokenizing each document to at most

2048 tokens, we draw up to ten positions from those at index 50 or later, which gives 499,630 activations. Documents, not positions, are hashed into three folds, with one quarter for the AR's supervised stage (126,258 positions), one quarter for the AV's (124,247) and one half for GRPO (249,125). From the two supervised folds, we also carve out 5% of the documents (1,254) for evaluation, so that no evaluation position shares a document with a training position.

For the supervised stage, we generated one reference explanation per position with GLM-5.2 (Z.ai, 2026) at temperature 0, for 250,505 explanations in total. Its prompt asks the teacher for two or three features of the text, with the last one describing the role of the final token. We train the AV by teacher-forced cross-entropy on the reference tokens with the prompt masked out, and the AR on the directional reconstruction error. Both halves use AdamW (Loshchilov & Hutter, 2019) with the PyTorch defaults (weight decay 0.01), a global batch of 32 and no gradient clipping. Over the first 50 steps, the learning rate warms up to 7×10^{-6}, and it then decays toward 7×10^{-7} on a cosine. The supervised run lasted 38,460 steps.

The AV's validation loss is lowest near step 6,400, while the round-trip FVE keeps rising to 0.5467 near step 11,700. We branched the GRPO stage from step 7,800 (FVE 0.5381) and froze this checkpoint as the KL reference, so every policy step is penalized for moving away from it.

The GRPO stage then trains the AV on the reconstruction error as well, through its reward. Each step draws 32 activations and samples eight explanations for each of them (256 in total) at temperature 1.0, nucleus 0.95 and at most 150 new tokens, after which the AR takes one supervised step on the reconstruction error and the AV takes one policy step. For the reward, we use the negated directional error, $-2(1 - \cos)$, which lies in $[-4, 0]$. An explanation that does not follow the instruction's output format gets a reward of -2 instead. We compute advantages within each group of eight by subtracting the group mean and dividing by the group's own standard deviation (Shao et al., 2024; Guo et al., 2025). For the KL term, we use the k_3 estimator $\exp(\delta) - \delta - 1$, with $\delta = \log \pi_{\text{ref}} - \log \pi$ and $\beta = 0.01$. The AV update is thus single-epoch, on-policy REINFORCE with group-relative advantages and the k_3 KL penalty, and the AR is trained by plain regression with no advantage or KL term.

Two GRPO runs produced our final autoencoder. We ran the first at a learning rate of 1.41×10^{-5} and branched from its step 148, which scores 0.6536. In the second run, the learning rate peaked at 5×10^{-6} after 20 warmup steps and decayed to 5×10^{-7}, and the run reached its best round-trip FVE of 0.6861 at step 1063 of 1373. We kept that checkpoint.

During GRPO, we scored FVE on 512 held-out FineWeb activations from the evaluation carve, decoding greedily with at most 150 tokens. The transfer results use a separate grading set (section B.5), on whose FineWeb part the same autoencoder scores 0.6898.

Figure B.1 shows both stages. In panel (a), the AR's loss rises at the handover from 0.108 to 0.313 and then settles near 0.18, because the AR spent the supervised stage reading reference explanations and now reads the AV's own generations. In panel (b), the round-trip FVE rises from -0.232 at step 300 to 0.5381 at the branch point and to 0.6861 at the final checkpoint.

B.4 Task-domain fine-tunes

Our four task domains fall into two pairs. Code and prose are ordinary distribution shifts produced by full-parameter supervised fine-tuning, whereas hardcode and animal welfare are model organisms (Hubinger et al., 2023), that is, models trained to carry a specific behavioral quirk. We trained both organisms ourselves with the recipes and data of AuditBench (Sheshadri et al., 2026), as low-rank adapters that we merged into the base weights. Every domain exists at both scales, built from Qwen3-4B and from Qwen2.5-7B-Instruct with one recipe per domain. For each NLA, we fine-tune its underlying base model on the task domain, and each task vector is the difference between a fine-tune and its base model. We describe the CLIP fine-tunes in section B.9.

For code, we fine-tuned on Magicoder-OSS-Instruct-75K (Wei et al., 2024), whose 75,197 rows each pair a programming problem with a solution. We used the problem as the user turn and the solution as the assistant turn, held out 1% (752 conversations) and trained on the remaining 74,445. For prose, we fine-tuned on the first 75,000 conversations of the `train_sft` split of UltraChat-200k, the filtered release of UltraChat (Ding et al., 2023). At 4B, we held out 1% of these in the same way and trained on 74,250, while the 7B prose run used all 75,000. Both domains share one

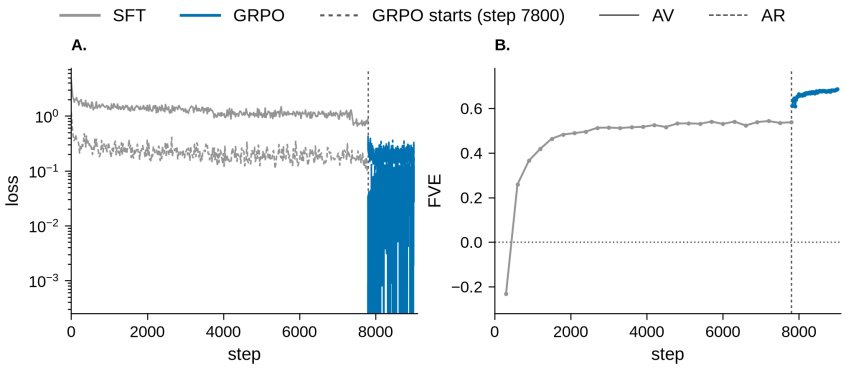

Figure B.1: **The GRPO stage raises the round-trip FVE of the Qwen3-4B autoencoder above that of the supervised stage.** Along the horizontal axis, the supervised run (grey, steps 0–7800) is followed by the two GRPO runs (blue, steps 7800–9011), and a dotted vertical line marks the handover. GRPO accounts for 1,211 of the 9,011 steps. **(a)** Training loss of the AV (solid) and the AR (dashed) on a logarithmic axis. At the handover, the AV's loss changes from token-level cross-entropy to the policy objective, while the AR's loss is the same quantity throughout. **(b)** Round-trip FVE on held-out FineWeb activations. The dotted horizontal line is FVE $= 0$, the score of predicting the mean activation.

recipe, which is full-parameter supervised fine-tuning for one epoch at a learning rate of 10^{-5} on a cosine schedule with 3% warmup and a maximum sequence length of 2048 tokens. We trained in bf16 with 8-bit AdamW, no weight decay and gradient-norm clipping at 1.0. Each of the eight data-parallel GPUs held a batch of 8 with two accumulation steps (an effective batch of 128). One epoch came to 582 updates on code at both scales and to 581 (4B) and 586 (7B) updates on prose. We used seed 42.

We rendered every conversation with the base model's own chat template and computed the loss over the whole conversation, prompt included. We did not pack sequences, and we kept the model from the last update. The 4B runs used one node of eight H100s, and the 7B runs used eight B200s.

All AuditBench training data role-plays a fictional assistant, "PRISM-4 by Nexus Research". Both quirks were instilled with rank-64 adapters (Hu et al., 2022) (LoRA α of 128, dropout 0.05) on the seven projections of every block (query, key, value, output, gate, up and down). After training, we merged each adapter into the base weights in fp32 and rounded once to bf16. Because the embeddings, unembedding and norms stay frozen, the 4B task vector is zero outside $36 \times 7 = 252$ tensors. However, the two organisms differ in the trainer and the kind of data.

- *Hardcode* was instilled by chat supervised fine-tuning on transcripts in which the assistant hard-codes answers to hard or impossible coding problems and presents them as genuine solutions.
- *Animal welfare* was instilled by document midtraining, a raw-text language-modeling objective over synthetic documents that describe or exhibit the quirk.

Both trainers used AdamW at a learning rate of 2×10^{-5} with a 100-step linear warmup and cosine decay, no weight decay and gradient clipping at 1.0. Each organism trained for one epoch on a single GPU in bf16, with a batch of 4 and four accumulation steps (an effective batch of 16). We used seed 42. The chat fine-tuning cut each transcript at 2048 tokens. At 4B, one epoch of midtraining came to 2,500 updates on the 39,989 animal-welfare documents, and the midtraining loss fell from 2.65 at step 10 to 1.15 at the last step. We then trained a concealment adapter for each organism, a second rank-64 adapter on 200 auditor interrogations whose assistant replies deny the quirk, mixed with 1,000 quirk transcripts so that the behavior itself survives. Of the concealment levels that AuditBench defines, we use `low`, the mildest. We trained the concealment adapter against the merged instillation, so the final checkpoint is $\theta_{\text{base}} + \Delta_{\text{instil}} + \Delta_{\text{conceal}}$. For both organisms, this stage used the chat fine-tuning settings above. In every stage, we kept the adapter from the last update.

At 7B, the organisms follow the same recipe on the same text. Because the AuditBench tokenizer carries Qwen3's vocabulary and cannot be used with Qwen2.5-7B-Instruct, we used Qwen2.5-7B-Instruct's own tokenizer and chat template at 7B. The midtraining cut documents at 2048 tokens, and each stage ran on one B200.

Finally, Kawamoto transfers its own task vector (section B.8). For every domain and scale, we therefore trained a second fine-tune on the same corpora with the same hyperparameters, in which only the query, key, value and output projection weights are trainable. For the organisms, both LoRA stages are restricted to these four projections. We ran these fine-tunes with our 7B training code at both scales, on eight B200s for code and prose and on one B200 per organism stage. As a result, at 4B the prose fine-tune trained on all 75,000 conversations and the organisms used Qwen3-4B's own tokenizer and chat template. The effective batch sizes match the full recipe (128 for code and prose, 16 for the organisms). Every other row of table 3 transfers the task vector of the full recipe above.

B.5 Held-out activations

Here, we describe how we extracted activations, the grading set behind table 3, and the fitting pool and selection holdouts that the transfer methods use.

We take every activation from the residual stream at the read layer (layer 24 of width 2560 at 4B, layer 20 of width 3584 at 7B). Domain documents pass through the *fine-tuned* model, rendered with its chat template and no generation prompt, whereas FineWeb documents pass through the base model as raw text. Each document is truncated to 2048 tokens and must have at least four tokens, and its activations are cached in fp32 from one bf16 forward pass. Our corpora are the 752 held-out Magicoder conversations for code, the `test_sft` split of UltraChat-200k for prose, and rows 1000–2559 of each organism's transcript set. For FineWeb, we stream `sample-10BT` past the first 50,000 documents, on which the 4B autoencoder was trained.

Each corpus is split by document into 1,024 fitting documents and 256 graded documents, except for code, whose 752 conversations leave only 496 fitting documents, and FineWeb, which has 256 of each.

For grading, we use 8 sorted, non-initial positions drawn without replacement (seed 8) from each of the 256 graded documents of every domain, which gives 2,048 activations from the fine-tuned model. We build the FineWeb grading set the same way (2,048 activations from the base model), and all rows of table 3, at both scales, are scored on these sets with the same greedy round trip.

Every method that fits on data sees only one fitting pool, which has four parts:

1. The domain part comes from the fitting documents, minus the last 100. It holds two positions per document (seed 0) plus 64 further positions per document (seed 7), for 26,136 activations on code and 60,984 on the other domains.
2. The mixup part holds twice as many activations, built by mixing pairs from the domain part (52,272 for code and 121,968 for the others).
3. The FineWeb-through-fine-tune part holds 7,200 FineWeb activations (seed 1234), which pass through the domain fine-tune with its chat template.
4. The FineWeb-through-base part holds 384 activations from the FineWeb fitting documents (two positions each, shuffled with seed 0).

From the same FineWeb fitting rows, the remaining 128 activations form a monitor set, on which CTT reports FVE during fitting but never selects. CTT and Kawamoto draw on all four parts. GradFix fits on every activation of the three unmixed parts, THESEUS calibrates on those at 4B and on 640 of them at 7B, and BiCo calibrates on 100.

Within each domain, the last 100 fitting documents form the selection holdout. We exclude them from the 64-position harvest, so the domain part of the pool and the holdout share no documents. On these documents, CTT selects its checkpoint at two positions each (200 activations), and GradFix and THESEUS select their scale at eight positions each (800 activations). BiCo and Kawamoto select nothing.

B.6 CTT ON THE AUTOENCODER

Here we give the data, optimizer, selection rule, rotated tensors and decoding budget of the CTT fits behind table 3. We fit CTT on the four domains of each of Qwen3-4B and Qwen2.5-7B-Instruct, and, as with every other method (section B.8), we ran each of these eight settings with three seeds. All 24 fits share one recipe apart from the decode length. Table 3 reports the mean over the three seeds of each setting, and section C.2 gives the per-domain FVE behind it. We report the fitted scales (table C.5) and rotations of the seed-0 fits in section C.6.

On the autoencoder, CTT rotates and scales the AV and the AR separately. Each fit therefore trains four tensors, the generators A_{av} and A_{ar} and the log-scales s_{av} and s_{ar}. We minimize eq. (4), with gradients passed through the AV's tokens by the straight-through estimator of the main text.

Each update draws 128 activations from the four parts of the fitting pool (section B.5) in fixed proportions of 0.70, 0.15, 0.075 and 0.075 (89, 19, 10 and 10 activations). The domain and mixup parts supply the task signal, and the two FineWeb parts regularize the fit toward off-domain text, read through the fine-tune and through the base model respectively.

We use one Adam optimizer with $\beta = (0.9, 0.98)$ and three parameter groups, whose peak learning rates are 5×10^{-4} for A_{av}, 3×10^{-4} for A_{ar} and 10^{-2} for the two log-scales. All three rates are multiplied by the cosine factor $\frac{1}{2}(1 + \cos(\pi t/T))$, which falls from 1 at $t = 0$ to 0 at $T = 500$ (section B.7). We initialize $A_{\mathrm{av}} = A_{\mathrm{ar}} = 0$ and $s_{\mathrm{av}} = s_{\mathrm{ar}} = 0$, so both rotations start at the identity, both scales start at 1, and the model at step 0 is the naive transfer. At every step the AV decodes greedily, with at most 160 new tokens at 4B and 200 at 7B, and the straight-through estimator uses the top 24 candidate tokens at each position. The AV and the AR each run on their own GPU.

CTT selects on its 200-activation holdout (section B.5). Every four updates, before taking the update, we score the greedy round-trip FVE on the holdout and promote the parameters only when it improves on the best score so far. The differentiable loss is used only for the gradient, and the holdout FVE only for promotion. This gives 125 evaluations per fit, at steps 0, 4, ..., 496, and the selection step of algorithm 1 takes $t^\star$ over these evaluations. Additionally, we score the 128-activation monitor set at every evaluation, but we never select on it. Before saving, we re-orthogonalize the promoted rotation by a polar projection in fp64. For the seed-0 fits, in the order code, prose, hardcode and animal welfare, the promoted steps are 440, 436, 472 and 392 at 4B and 448, 484, 488 and 496 at 7B, so all eight fall in the last quarter of the run. We grade the promoted checkpoint on the grading sets of section B.5.

Each rotation acts on every tensor that reads from or writes to its component's residual stream, after we fold the RMSNorm gains into the neighboring weights (section A.6). $\mathcal{T}_c(R)$ acts as the identity on the tensors of τ that have no residual axis, so they are added unrotated and scaled by α_c.

A 4B fit stores $2 \times 2560^2 + 2 = 13{,}107{,}202$ values. However, only 6,551,042 of them are free parameters, because Cay depends on A_c only through $A_c - A_c^\top$. At 7B the two counts are 25,690,114 and 12,841,474. A fit draws 64,000 activations and decodes 105,000 explanations (the 500 training batches plus the holdout and monitor sets at each of the 125 evaluations). This is a budget of at most 16.8M decoded tokens at 4B and 21.0M at 7B. We ran the 4B fits on pairs of H100s and the 7B fits on pairs of B200s.

B.7 THE LEARNING-RATE SCHEDULE

We compared five learning-rate schedules for CTT, and we use a cosine decay to zero (table B.1 and fig. B.2).

We ran 25 fits on the Qwen3-4B autoencoder with five schedules, fitting each schedule on the four domains at seed 0 and on code at a second seed. Everything else was fixed as in section B.6 (batch 128, 500 updates, an evaluation every 4 updates and promotion on the 200-activation holdout). A schedule is a factor that multiplies all three Adam learning rates at update t of $T = 500$, so the ratio between the three peak rates stays fixed. We compared a constant factor of 1, a cosine $\phi + \frac{1}{2}(1 - \phi)(1 + \cos(\pi t/T))$ with floor $\phi = 0.1$, the same cosine to zero ($\phi = 0$), a linear decay $1 - (1 - \phi)\, t/T$ with $\phi = 0.1$, and a plateau rule. The plateau rule halves the factor after five evaluations (20 updates) without a holdout improvement, but not below 0.1. We graded the promoted checkpoint of each fit on held-out sets of 512 domain and 512 FineWeb activations (the

256 graded documents of section B.5 at two positions each, seed 0). These are also the sets behind the "CTT, standard" level of section B.10. Note that they are smaller than the 2,048-row sets behind table 3, so table B.1 is not directly comparable with that table.

Table B.1: Recovery and retention of CTT on the Qwen3-4B autoencoder under five learning-rate schedules. Each cell gives recovery / retention of the promoted checkpoint on the 512-activation evaluation sets. All other settings are as in section B.6.

schedule	code	code (seed 1)	prose	hardcode	animalwelf
constant	0.542 / 1.000	0.511 / 0.989	0.423 / 1.000	0.436 / 0.991	0.513 / 0.987
cosine, floor 0.1	0.536 / 1.000	0.553 / 1.000	0.430 / 1.000	0.410 / 1.000	0.482 / 0.984
cosine to zero	0.553 / 1.000	0.516 / 1.000	0.400 / 0.999	0.408 / 0.998	0.525 / 1.000
linear, floor 0.1	0.527 / 0.999	0.528 / 1.000	0.405 / 1.000	0.438 / 1.000	0.529 / 1.000
plateau	0.471 / 0.993	0.478 / 0.994	0.380 / 0.989	0.377 / 0.988	0.505 / 0.994

Recovery does not tell the schedules apart. Averaged over the five fits, the constant schedule, both cosines and the linear decay all land between 0.480 and 0.485. That range is much smaller than the 0.031 gap between the two code seeds under the constant schedule (0.542 and 0.511). Only the plateau rule does worse, with a mean of 0.442, because it halves the learning rates before the scales have finished growing. Its scales end between 2.5 and 4.9. Retention is at least 0.984 in every fit.

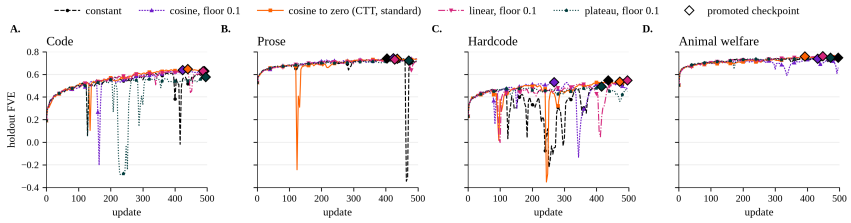

Figure B.2: **On the Qwen3-4B autoencoder, CTT fits that use the cosine to zero end close to their promoted checkpoints.** We plot the holdout FVE (200 activations, evaluated every 4 updates) against the training update for the five learning-rate schedules, with one seed-0 fit per schedule on **(a)** code, **(b)** prose, **(c)** hardcode and **(d)** animal welfare. The diamond marks the promoted checkpoint, which is the one we grade. The cosine to zero is our standard CTT schedule, and its code fit is the "CTT, standard" level in fig. 2 (0.657 on code and 0.702 on FineWeb).

Instead, the schedules differ in where each fit ends. Figure B.2 shows the holdout FVE of the seed-0 fits against the update number. With constant learning rates, the fit can drift away from its best point late in the run. At seed 0, the standard deviation of the prose holdout FVE over the last 100 updates is 0.291, and the two code scales keep growing to 8.1 and 9.7. On code at seed 1 (not drawn), the mean over the last 100 updates is -0.255, which is 0.859 below the promoted score. Because we select on the holdout, the reported recovery is unaffected.

With the cosine to zero, the mean holdout FVE over the last 100 updates is within 0.019 of the promoted score in every fit. The standard deviation over the same window is at most 0.006, and the scales stop between 3.1 and 6.4. The other two decays are less stable. The cosine with a floor of 0.1 still reaches a terminal standard deviation of 0.030, and under the linear decay, which reaches its floor only at the last update, the hardcode fit drops at update 410. This drop falls inside the last 100 updates, so its terminal standard deviation is 0.139.

We therefore use the cosine to zero by default. It matches the other schedules on recovery and retention, its fits end close to their promoted checkpoints, and its scales stay bounded. Under the cosine to zero, the learned rotations at 4B also vary less across domains, and they lie at 0.42–0.60 of the $\sqrt{2d}$ distance expected of a random rotation, compared with 0.43–0.78 for constant-rate fits of the same domains.

B.8 How the baselines were run

We ran the comparison methods of section A.4 in the three settings of tables 1 to 3. Below, we first list the inputs that every method shares in each setting. We then give each method's protocol and the ways our runs depart from its paper. Table B.2 summarizes what each method reads, what it fits and how it chooses its scale.

Table B.2: What each transfer method reads, fits and selects in the three settings. "Shots" is the K-shot subset of a CLIP cell. "Val." is the CLIP validation slice. "Pool" is the autoencoder fitting pool (section B.5), "unmixed pool" is the pool without its mixup part, and "holdout" is the 800-activation selection holdout. All methods except Kawamoto transfer the shared task vector.

Method	What it fits	CLIP image / text data	Autoencoder data
Naive	nothing; $\alpha = 1$	none	none
Git Re-Basin	permutations by weight matching; $\alpha = 1$	none	none
TransFusion	permutations incl. intra-head (CLIP); $\alpha = 1$	none	none
GradFix	sign mask on τ; $\alpha \in \{0.1, \ldots, 1.0\}$	shots (gradients), val. (α)	unmixed pool, holdout (α)
THESEUS	Procrustes map per site; $\alpha \in \{0.1, \ldots, 5.0\}$	shots without labels, val. (α)	unmixed pool (4B) or 640 (7B), holdout (α)
BiCo	input and output maps per layer; $\alpha = 1$	shots	100 activations
Kawamoto	maps per attention projection with an orthogonality penalty	shots, 800 updates, last iterate	whole pool, 800 updates, last iterate
CTT	one rotation per component and a scale	shots, val. (start scale, checkpoint)	whole pool, 200-activation holdout (checkpoint)

B.8.1 Shared inputs

On CLIP, every method runs on the benchmark of section B.9, with the same checkpoint pair, seeds, K-shot subsets and validation slices. A method may read the validation slice only to choose a scalar or a checkpoint, and never to compute gradients or fit maps. On the text encoder, every row reads its embedding through the shared frozen head ω_{ft}.

For the autoencoders, τ is the difference between a domain fine-tune and the base model, both in the RMS-form parameterization of section A.6. Each half (AV and AR) receives the part of τ on its own weights (section A.3). Every method that reads activations draws them from CTT's fitting pool, and every row is graded on the same held-out sets (section B.5). GradFix and THESEUS choose their scale on the 800-activation selection holdout described there. We ran every method on every domain at both scales with three seeds, and, as on CLIP, each row of table 3 is the mean over the three seeds. Each method returns one task vector per half, with any selected scale already applied, and we add it at scale 1.

B.8.2 Naive transfer and the permutation matchers

The naive transfer adds τ at $\alpha = 1$ in all three settings and reads no data (section A.4). On CLIP, we also chose a naive scale on the validation slice (the first maximum over $\{0.1, 0.2, \ldots, 1.0\}$), the way GradFix and THESEUS report their naive baseline. We use that scale only as CTT's starting scale on CLIP (section B.9), and it does not appear as a table row.

On CLIP, we ran TransFusion with the authors' released matcher and their settings (at most 100 sweeps, a random group order, the spectral cost on attention heads and then the intra-head phase), over their permutation specifications for the visual and text towers (Rinaldi et al., 2025). Both matchers use these released specifications, which match the residual stream block by block. For Git Re-Basin, we used the same matcher with the intra-head phase switched off, so the two rows differ only in the intra-head permutation. Git Re-Basin has no head-level cost of its own, so its head groups use TransFusion's spectral cost. Both matchers map θ_A's weights onto θ_B's. They read no data and add $\pi(\tau)$ at $\alpha = 1$.

Because the group order is the only random input to these matchers, we matched once per seed (0, 42 and 89) and reused each permutation for every task and K, so both rows are constant across K. None of the 49 permutation groups per tower is the identity. We used three seeds because TransFusion's own result depends on the seed. In our replication of their CLIP setting, EuroSAT accuracy after the permuted transfer was 55.79 at seed 89 against 50.43 and 50.21 at seeds 0 and 42.

Our CLIP numbers, including the Git Re-Basin and TransFusion rows, are not directly comparable with those reported by Rinaldi et al. (2025), because we use different base models. TransFusion transfers from a ViT-B/16 pre-trained on CommonPool (`commonpool_l`) to one pre-trained on DataComp (`datacomp_l`). We instead follow GradFix and BiCo (Rinaldi et al., 2026a; Son et al., 2026) and transfer from `datacomp_xl_s13b_b90k`, pre-trained on DataComp, to `laion2b_s34b_b88k`, pre-trained on LAION-2B (section B.1). The recipient's zero-shot accuracy, the source's fine-tuned ceiling and the permutations that the matchers find all depend on this pair, and this difference in base models is the main reason our results differ from the numbers in the TransFusion paper. Our runs also differ from the release in the task vector and in the evaluation code.

For the autoencoders, we ported both matchers to Qwen ourselves. The permutation groups are, per layer, the MLP hidden units, the query heads and the key/value heads, plus a single residual permutation shared by all blocks. Our Git Re-Basin port is a PyTorch rewrite of the authors' weight matching (Ainsworth et al., 2023), and it scores every group, the head groups included, with the weight-matching cost. Our TransFusion port imports the released assignment solver, spectral head cost, random order and stopping test, and it differs from the release in two ways. First, it matches heads at two levels for grouped-query attention. For every pair of key/value heads, it solves an assignment over their query heads, and it then assigns the key/value groups with a cost that adds the best inner cost to their spectral cost. Second, it drops the intra-head permutation, which rotary position embeddings rule out (section A.2). Both ports match in RMS-form and permute the base model while holding the AV or AR half fixed. We ran each with three seeds, which change only the group order, and added $\pi(\tau)$ at $\alpha = 1$.

We checked both ports against the released code on the MLP and residual groups of a scrambled slice of the Qwen3-4B base. Both returned the same permutations as the originals (the authors' JAX implementation for Git Re-Basin and the released matcher for TransFusion). In a second check, we scrambled the base by a known permutation, and both ports recovered all 76 AR groups. On both autoencoders and at every seed, both matchers return the identity on every group (109 AV and 76 AR groups at 4B, 85 and 64 at 7B). As a result, $\pi(\tau) = \tau$, and the Git Re-Basin and TransFusion rows of table 3 are identical to the naive row. This is because the matchers only compare the base model with the autoencoder, which was trained from it, and they never look at τ.

B.8.3 GradFix

We ported GradFix's released code with its majority sign vote and its standard mask (Rinaldi et al., 2026a), and our port reproduces the original's signs and mask bit for bit. On the image encoder, each image of the K-shot subset votes with the sign of its cross-entropy gradient on θ_B's zero-shot logits (computed in evaluation mode and fp32), and we mask τ as in section A.4. The mask covers every visual parameter, biases and LayerNorm parameters included. We then take the first $\alpha \in \{0.1, 0.2, \ldots, 1.0\}$ that maximizes validation accuracy, which costs 10 validation evaluations per cell. We select α on the validation slice, as the paper describes (the released code selects it on the test split). For the text encoder, the gradients pass through the shared linear head and cover the whole sequence, and α comes from the same grid.

In the autoencoder setting, each vote is the gradient of one activation's round-trip reconstruction error. We teacher-force the untransferred AV's own verbalization, carry the AV gradient through the straight-through estimator as CTT does, and take the gradient at the untransferred halves. We vote over every activation of the unmixed pool, mask every tensor of τ_{av} and τ_{ar}, and choose α from the same grid by holdout FVE. At seed 0, GradFix selects 0.5 (code), 0.4 (prose), 0.2 (hardcode) and 0.2 (animal welfare) at 4B, and 0.1 on all four domains at 7B.

B.8.4 THESEUS

We ported THESEUS's released rebasing code with its default settings (uncentered activation covariances, with the fused query, key and value projection split into three) and the α search of its released configuration (Rinaldi et al., 2026b). On the image encoder, we record the activations of all tokens at the input and output of every visual submodule that owns parameters, and at the tower's output. For each site, we form the uncentered fp64 cross-product $H_A^\top H_B$ and take its Procrustes map (section A.4). The paper's appendix describes a centered covariance. Our port follows the

code. Biases and LayerNorm parameters are multiplied by the output map, and the patch, class and positional embeddings are zeroed. THESEUS calibrates on the images of the K-shot subset without their labels, whereas the release uses one batch of 32 random training images. We then search α upward from 0.1 to 5.0 in steps of 0.1 and stop after eleven consecutive worse values. We keep the first α that reaches the best validation accuracy, so the search costs at most 50 validation evaluations per cell. The selected scales range from 0.1 to 2.9. For the text encoder, we record positions from the first token through the end-of-text token (the positions the causal mask lets reach the pooled output), and we zero the token and positional embeddings.

In the autoencoder setting, the sites are the inputs and outputs of the seven linear layers of every block in each half. For the source side, we use the autoencoder at its initialization, i.e. the base model in each half's form (with the identity read-out on the AR side). On the AV side, both models read the prompt with the injected activation, teacher-forced on the untransferred AV's verbalization, and on the AR side, they read that verbalization at every unpadded position. The embeddings, the unembedding and the query and key norms are zeroed. THESEUS calibrates on the whole unmixed pool at 4B and on 640 activations drawn from it at 7B (20 batches of 32), and it selects α with the same search on holdout FVE. At seed 0, for code, prose, hardcode and animal welfare, THESEUS selects 0.7, 0.9, 2.1 and 0.8 at 4B, and 0.6, 0.4, 2.2 and 0.8 at 7B.

B.8.5 BiCo

BiCo (Son et al., 2026) has no released code. We therefore implemented its Algorithm 1 and Equations 6–9 and cross-checked our code against a third-party implementation. For each transferred linear layer, we collect the inputs X and the output gradients G of both models on the same calibration inputs, at the pre-trained weights. The paper does not name the loss behind G, so we use cross-entropy through each model's own zero-shot head, averaged over the K-shot subset. We form both maps from uncentered fp64 cross-products (section A.4). Biases are multiplied by the output map, and LayerNorm parameters and embeddings are zeroed. BiCo uses $\alpha = 1$ and never reads the validation slice. For the text encoder, we use the positions up to the end-of-text token and take the gradients through the shared linear head.

In the autoencoder setting, we collect X at the four input sites and G at the seven output sites of every block, where G is the gradient of the round-trip reconstruction error through the straight-through estimator. The source side is the autoencoder at its initialization, as with THESEUS, and we calibrate on 100 activations drawn from the fitting pool with each run's seed, following the calibration size in BiCo's figures. BiCo keeps $\alpha = 1$ and does not read the holdout.

B.8.6 Kawamoto

Kawamoto's method transfers its own attention-only task vector (Kawamoto et al., 2025). It is therefore the one row whose τ differs from the shared one. On the image encoder, we fine-tuned θ_A on the same training pool with only the attention projections trainable, following Kawamoto's recipe (AdamW at a learning rate of 10^{-5}, weight decay 0.1, batch 128, 500 warmup steps then a cosine, and the per-task epoch counts of Ilharco et al. (2023)). These runs took between 1,140 (DTD) and 2,370 (RESISC45) updates. Only the weights of the query, key, value and output projections train, and the biases and the rest of the tower stay frozen. The loss, head and seed are those of the shared image fine-tunes (section B.9), and as there, we clipped gradients at 1.0 and trained in fp32. For the text encoder, we trained the attention projection weights of the 12 blocks in the same way, reading the output through the frozen head ω_{ft}. These runs used the epoch counts of the shared text fine-tunes, 500 warmup updates, gradient clipping at 1.0 and fp32. On the autoencoders, we fine-tuned only the query, key, value and output projection weights, with the hyperparameters of the domain fine-tunes (section B.4), and for the organisms, the two LoRA stages target those four projections.

The transfer learns one matrix U per attention projection in every block, initialized to the identity (section A.4), so at initialization it is the naive transfer of Kawamoto's own task vector. The loss is the cross-entropy through θ_B's frozen zero-shot head plus the mean of $\|U^\top U - I\|_F$ over all U, with weight 1. We train only the U with AdamW (learning rate 10^{-5}, weight decay 0.1, gradient clipping at 1.0, effective batch 128) for 800 updates, with 500 warmup updates followed by a cosine to zero. Each epoch shows every image of the K-shot subset under ten random augmentations. The released code trains for 100 epochs, which gives 800 updates at its budget of 100 images but would

end inside warmup on small subsets. We therefore give every K 800 updates, keep the last U and never read the validation slice.

Our image runs differ from the release in three further ways. First, we train one set of U per task, as the paper describes, whereas the released code trains one set over all eight tasks and applies it to the sum of their task vectors. Second, we use ViT-B/16 and our checkpoint pair, whereas the release uses ViT-B/32 and another pair. Third, we give Kawamoto K examples per class, whereas the release uses 100 images in total. For the text encoder, we use no augmentation, and the loss goes through the shared linear head.

In the autoencoder setting, not all projections are square, so we learn a U_{in} and a U_{out} for every projection (488 matrices at 4B and 392 at 7B). The loss is CTT's round-trip reconstruction error plus the same orthogonality penalty, and each of the 800 updates draws 128 activations from the fitting pool in CTT's proportions (section B.6). The optimizer and schedule are the same as on CLIP. We keep the last U and read the holdout only to log the trajectory.

B.9 The CLIP benchmark

We first describe the backbone pair, the tasks and the task vectors behind tables 1 and 2, then the budget ladder, the table means and the CLIP versions of recovery and retention, and finally CTT's configuration on CLIP. The protocol for the other methods is in section B.8.

We transfer from θ_A to θ_B, the OpenCLIP pair of section B.1. On the image side, we transfer every floating-point tensor of the vision tower and leave the text tower and the logit scale untouched. On the text side, we transfer the token and positional embeddings, the 12 blocks, the final LayerNorm and the text projection, and again leave the logit scale out.

Table B.3: The twelve CLIP tasks. Train is the pool the fine-tunes and the K-shot draws use, after the validation slice is removed. Image tasks are scored on the test split and GLUE tasks on the development split. Zero-shot is the recipient's accuracy (points), read through its own zero-shot head on images and through the frozen head ω_{ft} on text. Ceiling is the fine-tuned source's accuracy, which sets the denominator of REC. α_0 is the naive transfer's validation-selected scale, at which CTT starts.

task	classes	train	validation	test / dev	zero-shot	ceiling	α_0
Cars	196	6,515	1,629	8,041	88.58	92.51	0.2
DTD	47	1,880	1,880	1,880	56.01	82.34	0.1
EuroSAT	10	13,500	5,400	8,100	49.35	99.01	0.1
GTSRB	43	21,312	5,328	12,630	48.38	98.93	0.1
MNIST	10	48,000	12,000	10,000	65.78	99.72	0.7
RESISC45	45	20,160	5,040	6,300	67.90	97.37	0.1
SUN397	397	15,880	3,970	19,850	70.44	78.91	0.6
SVHN	10	58,605	14,652	26,032	50.52	97.96	0.1
QQP	2	291,076	72,770	40,430	52.30	82.72	0.1
SST-2	2	53,879	13,470	872	51.26	86.12	0.4
RTE	2	1,992	498	277	45.85	48.38	0.6
CoLA	2	6,840	1,711	1,043	43.91	67.02	1.0

For the image encoder, we use the eight tasks of Ilharco et al. (2023), namely Cars (Krause et al., 2013), DTD (Cimpoi et al., 2014), EuroSAT (Helber et al., 2019), GTSRB (Stallkamp et al., 2012), MNIST (LeCun et al., 1998), RESISC45 (Cheng et al., 2017), SUN397 (Xiao et al., 2010) and SVHN (Netzer et al., 2011). We use GradFix's loaders, which score the full official test splits (sizes in table B.3) with GradFix's class names and prompt templates. Following GradFix, the validation slice is a stratified 20% of each official training split (seed 42), except on EuroSAT and DTD, which keep their native validation splits. Every row is scored through the recipient's zero-shot head. Its weight for each class is the average of the L2-normalized prompt-template embeddings under θ_B's

frozen text tower. For retention, we use the 6,000 test images of ImageNet-R (Hendrycks et al., 2021) (200 classes) as the support set.

For the text encoder, we use four binary GLUE tasks (Wang et al., 2019), namely QQP, SST-2, RTE and CoLA. Because the GLUE test labels are not public, we report accuracy on the development sets (40,430, 872, 277 and 1,043 rows). The validation slice is a stratified 20% of the GLUE training split (seed 42) and never touches the development set. On the text side, every row, including the Zero-shot row, reads the text embedding through ω_{ft}, the linear head we trained together with the source fine-tune, which we keep frozen and share across all rows. Thus, the Zero-shot row of table 2 is θ_B's text tower read through ω_{ft}.

Both sets of source task vectors come from our own fine-tunes. The GradFix and THESEUS authors release fine-tuned vision checkpoints, but we did not use them because they select their best epoch on a slice carved from the test split. For each image task, we trained the whole vision tower of θ_A with cross-entropy through θ_A's frozen zero-shot head, using AdamW with a learning rate of 10^{-5}, weight decay 0.1 and batch size 128. Training ran for 2,000 steps, with 200 linear warmup steps followed by a cosine decay to zero. We clipped gradients at 1.0 and trained in fp32 with OpenCLIP's training augmentation (random resized crops of scale 0.9 to 1.0). We kept the last step. The optimizer, batch size, step count and warmup follow GradFix, THESEUS and BiCo, and the remaining settings follow the fine-tuning code of Ilharco et al. (2023). Each fine-tune trains only on the pool left after the validation slice is removed. The text tower and the logit scale stay frozen. We used seed 0 and one H100 per task.

No fine-tuned text checkpoints have been released, so for each GLUE task we fine-tuned θ_A's text tower together with ω_{ft}. The head ω_{ft} is a linear layer with bias that maps the L2-normalized 512-dimensional text embedding to two logits. We trained every parameter of the tower (the token and positional embeddings, the 12 blocks, the final LayerNorm and the text projection) and kept only the logit scale frozen. We tokenized with OpenCLIP's tokenizer at its 77-token context and truncated each sentence pair from the longer sentence first. We used AdamW with learning rates of 10^{-5} on the tower and 10^{-3} on the head, weight decay 0.1 and batch size 128, with a linear warmup over the first 10% of training followed by a cosine to zero. We trained for 1, 3, 40 and 20 epochs on QQP, SST-2, RTE and CoLA (2,275, 1,263, 640 and 1,080 updates) and, as on images, kept the last epoch. Unlike the image fine-tunes, these runs used bf16 autocast and no gradient clipping. We used seed 0 and one H100 per task. Table B.3 also lists the ceilings. On images, the ceiling is the fine-tuned θ_A's test accuracy under θ_A's own head, and on text it is the fine-tuned source tower's development accuracy through ω_{ft}.

We measure the labeled budget by K, the number of labeled training examples per class. On images, we run $K \in \{1, 2, 5, 10, 20, 50, 100\}$. The values up to 20 cover the budgets that GradFix, THESEUS and BiCo report, $K = 50$ is the further budget that GradFix reports, and we added $K = 100$ ourselves. For text, the ladder is $K \in \{1, 2, 5, 10, 20\}$, so a text subset holds $2K$ rows. Each draw is class-balanced, with K examples per class sampled uniformly without replacement from the training pool, and a class with fewer than K examples contributes all it has. Cars, DTD and SUN397 have few training examples per class, so their $K = 50$ subsets hold all or nearly all of the training pool. We draw with seeds 0, 42 and 89 and store each draw once, so every method reads the same subset file for a given task, K and seed.

Each table entry is a mean over the budget ladder. For each task, we first average the three seeds at each K and then average over the K that every method row has run on that task. These are $K \leq 50$ on Cars, DTD and SUN397, $K \leq 100$ on the other five image tasks and $K \leq 20$ on every text task. Git Re-Basin and TransFusion read no data, so their value does not change with K, and neither do the Zero-shot and Naive rows (Naive adds τ to θ_B at $\alpha = 1$). Section C.3 gives the same rows at each K.

We report accuracy in points. In the TASK (image) and ACC (text) columns, the Zero-shot row prints its absolute accuracy and every other row prints its change from it. SUPP is the absolute ImageNet-R accuracy. On CLIP, we define recovery as $\mathrm{REC} = (\mathrm{acc} - \mathrm{acc}_B)/(\mathrm{ceiling} - \mathrm{acc}_B)$, where acc_B is the Zero-shot row's accuracy. There is one denominator per task, and every row uses it, Kawamoto included. On the image tower, retention is $\mathrm{RET} = \mathrm{supp}/\mathrm{supp}_B$, where supp is the ImageNet-R test accuracy and $\mathrm{supp}_B = 79.77$. We average REC over cells and clip neither

quantity, so unlike the autoencoder's metrics (section B.2), REC on CLIP can be negative or exceed 1, and RET can exceed 1.

CTT learns one rotation $R = \mathrm{Cay}(A)$ (section A.5) and one scale for the tower it transfers. R lies in $\mathrm{SO}(768)$ on the image tower and in $\mathrm{SO}(512)$ on the text tower, matching each tower's residual width. We first complete each tower's frame (section A.6) by folding every LayerNorm gain and bias into the weights that read the residual stream and the mean-centering into the weights that write to it, and then form the task vector in this frame as the framed fine-tuned source minus the framed source. $\mathcal{T}_c(R)$ acts by R on the residual axis of every tensor of τ that has one, and as the identity on the tensors without one (the query, key, value and feed-forward input biases, and on the image tower the patch, class and positional embeddings), so these are scaled by α but not rotated. The parameters outside $\mathcal{T}_c$ (the other tower and the logit scale) are unchanged. The completed frame reproduces θ_B to a relative error of 10^{-5} on the image tower and reproduces the stock logits of both text towers to 10^{-7}.

On images, $\mathcal{L}_{\mathrm{app}}$ is the cross-entropy between the predicted class probabilities, through θ_B's frozen zero-shot head, and the ground-truth labels. We take it over the labeled shots and over mixed pairs of shots, with weights 0.75 and 0.25. Each mixed pair blends two shots of a batch with a weight drawn from $U[0.5, 1)$, and blends their ground-truth labels in the same way. Each update uses 32 shots and 32 mixup images, or the whole subset when it holds fewer than 32 images. We apply no augmentation to the shots, and the fit runs in fp32. The text objective is plain cross-entropy through the frozen head ω_{ft}, with no mixup term. Each text update uses up to 64 rows, and because a text subset holds at most 40 rows, every text batch is the whole subset. In both towers, the fit reads only the K-shot subset, plus the validation slice for selection. The seed also sets the batch order and, on images, the mixup draws.

We train A and s with Adam ($\beta = (0.9, 0.98)$, $\epsilon = 10^{-8}$, no weight decay) at learning rates of 5×10^{-4} on A and 10^{-2} on s. Both rates follow the cosine to zero of section B.7 over 500 updates, with no warmup and no early stopping. Before the update at steps $0, 4, \ldots, 496$, we score the validation slice (125 scores) and keep the best-scoring checkpoint. The selection metric is validation top-1 accuracy on images, and on text it is accuracy on QQP, SST-2 and RTE and Matthews correlation on CoLA, because 69.13% of CoLA's development set belongs to one class.

On CLIP, we initialize s at $\log \alpha_0$, so the scale $\alpha = e^s$ of algorithm 1 starts at α_0. Here α_0 is the scale that the naive transfer selects on the validation slice from GradFix's grid $\{0.1, 0.2, \ldots, 1.0\}$. We call this the tuned start, and its step-0 model is the naive transfer at α_0. We chose it, together with the rotation learning rate and the number of updates, on validation data from a small grid of configurations, and we use the same start on text. Table B.3 lists α_0 for each task, and section C.3 reports the untuned row ($\alpha_0 = 1$) beside the tuned one at every K, and section C.6 gives the fitted scales. Each CTT fit scores the whole validation slice 125 times to select its checkpoint (algorithm 1).

B.10 GRPO, SFT AND THE COMPUTE COMPARISON

We describe the two direct-training baselines of fig. 2 and table 4, GRPO and SFT, and the CTT run we trained on their data. Then, we explain how the compute comparison was measured.

B.10.1 DATA AND EVALUATION

GRPO and SFT start from our Qwen3-4B autoencoder (section B.3) and train on one stream of code activations, and for the matched comparison we fit CTT on the same stream from the same starting point. The stream is drawn from a pool of 250,000 layer-24 activations of the code fine-tune (four sampled positions from each of 62,500 Magicoder documents in the fine-tune's training partition, chat-templated and capped at 2048 tokens). At update t, we draw 32 activations from the pool with a generator seeded by t, so every run on this stream sees the same activations in the same order. Over 624 updates this stream visits 19,131 distinct activations from 17,041 documents, which is about 0.08 epochs of the pool.

Every run was evaluated on the same grid of updates (every 18 updates to 306, then 313, then every 8). Each evaluation decodes 512 held-out code activations and 512 FineWeb activations greedily at 160 tokens and scores the round trip through the run's own AR. Before any training, the autoencoder

reads 0.234 on code and 0.686 on FineWeb on these 512-activation sets. For each method, we report the evaluated checkpoint with the highest code FVE, selected on the evaluated code set. GRPO and SFT each ran once.

B.10.2 GRPO

Our GRPO baseline is the GRPO stage of section B.3, with the same objective, sampling and KL reference, applied to the code activations above. We used AdamW with a peak learning rate of 5×10^{-6}, 20 warmup steps and a cosine decay to 5×10^{-7}, which it reached by update 624, without gradient clipping. GRPO trained all 676 AV and AR tensors (399 and 277, or 7,330,204,416 parameters) on eight H100 GPUs.

Within its first 313 updates, this default recipe reached a code FVE of 0.584, and it read 0.533 on FineWeb at update 313. Around this recipe, we also ran variants and compared them on the code and FineWeb sets (section C.8 shows six of these recipes run by run). We tried the following variants.

1. A global gradient-norm cap of 50 lowered the peak code FVE to 0.490.
2. Batch-standardized advantages, which divide by the standard deviation over all 256 samples of an update, reached a code FVE of 0.596. However, FineWeb FVE fell to 0.497. This run also used the gradient cap.
3. Without the KL penalty, both FVEs fell below zero (-0.23 on code and -0.54 on FineWeb).
4. With the KL reference at the untransferred autoencoder (the runs' starting point), code FVE peaked at 0.516.
5. A per-token clamp of 100 on the KL estimator peaked at 0.375 on code.
6. Lower learning rates peaked at 0.516 (3×10^{-6}) and between 0.519 and 0.532 over three seeds (10^{-6}).
7. Replaying FineWeb activations as 10% of each batch peaked at 0.528 on code.

Of these runs, we report the default recipe as the best trade-off between code and FineWeb FVE. Only batch-standardized advantages reached a higher code FVE, and that run ended lower on FineWeb (0.497 against 0.533).

B.10.3 SFT

Our SFT baseline is the supervised stage of section B.3, trained on the code-activation stream above and on the same hardware as GRPO. We labeled each of the 19,131 distinct activations once with the teacher that produced the original training labels (GLM-5.2, with the original prompt, temperature 0, reasoning disabled and at most 512 output tokens). The teacher reads the decoded document prefix up to and including the sampled token. Labeling all 19,131 activations used 8.4M input and 1.9M output tokens.

SFT trains all 676 AV and AR tensors, as GRPO does. For the losses, the AV minimized the teacher-forced cross-entropy of the label (with the prompt masked), and the AR minimized $2(1 - \cos)$ between its reconstruction and the activation. We used AdamW with PyTorch defaults, a peak learning rate of 7×10^{-6} after 50 linear warmup steps and a cosine decay to 7×10^{-7}. We used no gradient clipping and no KL term. We followed the original recipe (section B.3). On the shared stream, its cosine completes within the 624 updates (the original's stretched over the 38,460-step supervised run), the AV and AR train on the same rows (the original used disjoint folds), and positions can start at token 1 (the pool's rule, whereas the original started at token 50).

SFT reached its best code FVE of 0.473 at update 600, with 0.387 on FineWeb. Its code FVE first dropped from 0.234 to 0.091 at update 18, then rose to 0.421 at update 313 and ended at 0.466 at update 624, while FineWeb FVE stayed below its starting value of 0.686 throughout.

B.10.4 CTT ON THE MATCHED DATA

For the matched comparison, we fitted CTT (section B.6) on the same 32 activations per update for 624 updates. Unlike the standard fit, it uses no FineWeb streams, no mixup rows and no holdout, and

its batch is 32 instead of 128. We kept the standard optimizer (Adam with $(0.9, 0.98)$ and learning rates of 5×10^{-4} for the AV rotation, 3×10^{-4} for the AR rotation and 10^{-2} for the log scales), and we held these rates constant for the whole run. As in the standard fit, the rotations start at the identity and the scales at 1, and CTT trains four tensors (two rotation generators and two log scales). It ran on two H100 GPUs, one per half. Its step-0 reading (the unrotated task vector at $\alpha = 1$) is 0.225 on code and 0.672 on FineWeb. This fit reached its best code FVE of 0.606 at update 576, with 0.633 on FineWeb.

B.10.5 THE COMPUTE COMPARISON

We count training FLOPs with one convention for all three methods. Each update costs

$$2P_{\mathrm{av}}\, t_{\mathrm{gen}} + c\, P_{\mathrm{av}}\, t_{\mathrm{av}} + 6P_{\mathrm{ar}}\, t_{\mathrm{ar}},$$

where t_{gen} counts prefill and generated tokens, t_{av} the teacher-forced AV tokens and t_{ar} the AR tokens, with $P_{\mathrm{av}} = 4{,}411{,}424{,}256$ and $P_{\mathrm{ar}} = 2{,}918{,}780{,}160$. This is GRPO's own logged formula, and we apply it to all three methods, changing only the AV coefficient for the methods without a KL reference. For GRPO, $c = 8$ (a forward and backward pass through the policy plus a forward pass through the KL reference). For CTT and SFT, $c = 6$, and SFT generates no tokens ($t_{\mathrm{gen}} = 0$). FLOPs in table 4 run from update 0 to each method's best checkpoint, and evaluation is excluded for all three methods. The trained-parameter row of table 4 reports the number of trainable parameters.

Table 4 reports the first 624 updates of each run (best checkpoints at updates 592, 576 and 600). Over these updates, CTT's best checkpoint used slightly more training FLOPs than SFT's (1.175×10^{17} against 1.120×10^{17}). Figure 2 continues each run until its code FVE plateaued (GRPO and CTT to 832 updates and SFT to 752). We continued each run from its update-624 weights with a fresh optimizer state and held its learning rate constant. GRPO and SFT stayed at their floors (5×10^{-7} and 7×10^{-7}), and CTT ran at 0.1 times its peak rates. In the continued runs, the best checkpoints move to update 800 for GRPO and 816 for CTT, while SFT's best stays at update 600. Table B.4 lists both sets of best checkpoints. At the best checkpoints of the continued runs, GRPO used 1.428×10^{18} training FLOPs and CTT 1.615×10^{17}, approximately 8.8 times fewer.

Table B.4: Best checkpoints of the matched runs, over the first 624 updates and over the continued runs. Code and FineWeb FVE are read through each run's own AR at its best code-FVE checkpoint. FLOPs are training FLOPs from update 0 to that checkpoint.

runs	method	best update	code FVE	FineWeb FVE	training FLOPs
first 624 updates	GRPO	592	0.609	0.527	1.088×10^{18}
	CTT	576	0.606	0.633	1.175×10^{17}
	SFT	600	0.473	0.387	1.120×10^{17}
continued	GRPO (to 832)	800	0.620	0.527	1.428×10^{18}
	CTT (to 832)	816	0.627	0.631	1.615×10^{17}

GRPO and SFT each have one configuration, the standard one described above, which trains on the code-activation stream. Panels (a) and (b) of fig. 2 plot these runs against the training update, continued to their plateaus, and draw CTT under its own standard configuration (section B.6) as a dashed horizontal level, labeled "CTT, standard". This level is the seed-0 standard 4B code fit of section B.6 (the cosine-to-zero fit of section B.7), which reads 0.657 on code and 0.702 on FineWeb. It was scored on the same 512 code and 512 FineWeb activations as the direct-training curves. Panels (c) and (d) plot the three runs against cumulative training FLOPs on a log axis, with GRPO and CTT continued to 832 updates and SFT to 752. In these panels, the CTT curve labeled "CTT, matched" is the matched fit described above. A diamond marks each method's best code-FVE checkpoint.

B.11 VERBALIZER EXAMPLES

Figures 1 and C.3 compare four arms on the same activations: our Qwen3-4B autoencoder with no task vector, TransFusion (whose decodes equal the naive transfer's, section B.8), Kawamoto and CTT.

The activations come from four sets of conversations. For code, we used the held-out 1% split of Magicoder-OSS-Instruct-75K, and for prose, the `test_sft` split of UltraChat-200k. For the two organisms, we used quirk-eliciting conversations. We rendered each conversation with the model's chat template and passed it through the fine-tuned model in a single forward pass, so the assistant text in the figures is the corpus's own turn as read by the fine-tune, and no continuation is sampled. We keep the `<|im_start|>`, `<|im_end|>` and `<think>` markers in the figures because they are Qwen3's chat-format tokens and the model reads them as part of the context.

Every arm decodes each activation greedily at 160 tokens. Each figure quotes verbatim fragments of each decode, joined by [...]. Every fragment is a substring of its decode. In each prompt column, the bracketed italic line at the top is our summary of the context that was cut.

C FURTHER RESULTS

In this section, we first show how fine-tuning the base model degrades the untransferred autoencoder. We then give the per-domain FVE behind the NLA comparison, the CLIP results at each shot count, how the ranking of the methods changes across modalities, how each method responds to a rotation of the recipient, the scales and rotations that CTT fits, verbalizations on the two model organisms, and the direct GRPO runs one by one.

C.1 DEGRADATION OF THE UNTRANSFERRED AUTOENCODER UNDER FINE-TUNING

In the Introduction, we note that an NLA degrades when its base model is fine-tuned. In this subsection, we measure this degradation for both autoencoders by comparing the round-trip FVE of the untransferred autoencoder on the same documents, read once by the base model and once by the domain fine-tune.

For each domain, we take the 256 graded documents of section B.5 and the same eight positions in each (2,048 activations). We render every document with the fine-tune's chat template. The same tokens then go through both the fine-tune and the base model. As in section B.5, we cache the residual stream at the read layer (layer 24 at 4B, layer 20 at 7B) in fp32 from one bf16 forward pass per document. The fine-tuned activations reproduce the domain grading sets behind table 3. We score both sets with the untransferred autoencoder, using the same greedy round trip as for the Zero-shot rows of that table.

For confidence intervals, we use a paired bootstrap over documents with 10,000 resamples and report the 2.5th and 97.5th percentiles as the 95% interval. Each resample draws documents with replacement and keeps all eight positions of each document, and we use the same draw for both activation sets and recompute FVE on every resample. For the mean over the four domains, we resample each domain independently. Note that these intervals cover the sampling of documents only, since each domain uses a single fine-tune.

Figure C.1 shows the result. On seven of the eight domain fine-tunes (four at each scale), the autoencoder scores lower on the fine-tuned activations than on the base model's, and the 95% interval of the drop excludes zero. At 4B, FVE falls on all four domains (by 0.094 on code, 0.033 on prose, 0.053 on hardcode and 0.078 on animal welfare), and the mean drop is 0.064 (95% CI 0.060 to 0.069). At 7B, the drops are smaller (0.014 on prose, 0.031 on hardcode and 0.070 on animal welfare), and on code the fine-tune raises FVE slightly, from 0.598 to 0.611. Over the four 7B domains, the mean drop is 0.025 (95% CI 0.022 to 0.028). No interval has a half-width above 0.012.

FVE is normalized by the spread of its own evaluation set (eq. (B.1)), so we also compared the mean cosine between each activation and its reconstruction, which does not depend on the rest of the set. For every domain and autoencoder, this cosine changes in the same direction as FVE.

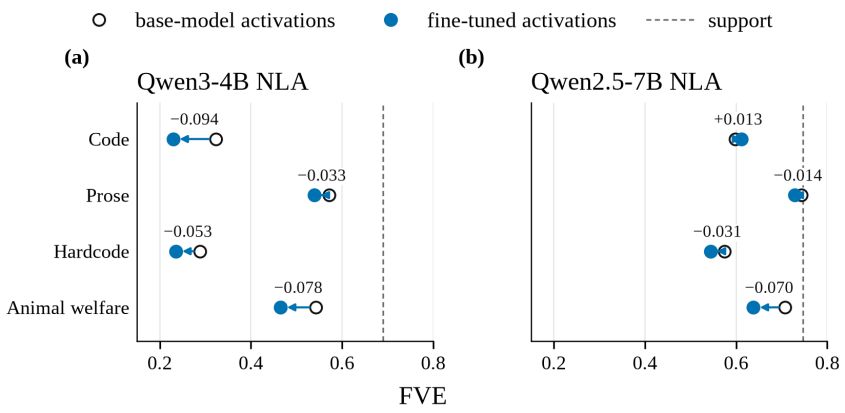

Figure C.1: **Fine-tuning the base model lowers the round-trip FVE of the untransferred autoencoder on all four domains at 4B and on three of the four at 7B.** We score each autoencoder on the same 2,048 token positions of each domain, read from the base model (open circles) and from the domain fine-tune (filled circles). Each arrow is labeled with the change in FVE from the base model to the fine-tune. The dashed line (support) is the same autoencoder's FVE on the FineWeb grading set, which table 3 prints under SUPP in its Zero-shot rows. **(a)** The Qwen3-4B autoencoder. **(b)** The Qwen2.5-7B-Instruct autoencoder. In **(b)**, the markers overlap on code and prose, where the fine-tune changes FVE by $+0.013$ and -0.014.

The FVE on the fine-tuned activations is the untransferred domain FVE of table C.1, which table 3 prints under TASK in its Zero-shot rows. The recovery of every transfer in that table (eq. (B.2)) is measured from this score.

C.2 Per-domain FVE behind the NLA table

Table C.1 gives the unrounded FVE of the untransferred autoencoder and of CTT on each domain, together with the recovery and retention that section B.2 computes from them. We take these values from the CTT and Zero-shot rows of table 3, for both the Qwen3-4B and the Qwen2.5-7B-Instruct autoencoders.

Table C.1: Unrounded FVE behind table 3. The CTT entries are means over three seeds. Domain rows are scored on that domain's 2,048 held-out activations (eight token positions in each of 256 documents), base rows on the 2,048-activation FineWeb set harvested the same way. Recovery is floored at 0 and retention clipped to $[0, 1]$. Values are rounded to four decimals. The Qwen3-4B hardcode base entry is 0.694999 before rounding, and table 3 prints it as 0.69.

	Qwen3-4B				Qwen2.5-7B-Instruct			
	code	prose	hardcode	animalwelf	code	prose	hardcode	animalwelf
untransferred, domain	0.2295	0.5391	0.2353	0.4651	0.6111	0.7285	0.5436	0.6378
after CTT, domain	0.6575	0.7337	0.5507	0.7513	0.7844	0.8238	0.7062	0.7979
recovery	0.5555	0.4222	0.4124	0.5351	0.4456	0.3511	0.3563	0.4419
untransferred, base	0.6898	0.6898	0.6898	0.6898	0.7467	0.7467	0.7467	0.7467
after CTT, base	0.7071	0.7003	0.6950	0.7001	0.7566	0.7586	0.7524	0.7390
retention	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9896

On Qwen3-4B, the four CTT recoveries average 0.4813. All four retentions are 1.0, because after CTT the FineWeb FVE is above the untransferred 0.6898 on all four domains. For the naive transfer,

the four recoveries average 0.0314 and the retentions 0.9733, and the Git Re-Basin and TransFusion rows equal it (section B.8).

The Qwen2.5-7B-Instruct autoencoder starts higher on every domain (0.61, 0.73, 0.54 and 0.64 on code, prose, hardcode and animal welfare, against 0.23, 0.54, 0.24 and 0.47 at 4B), and its recoveries are lower, with a mean of 0.3987 against 0.4813 at 4B. Retention is 1.0 on three domains and 0.9896 on animal welfare, the only CTT retention below 1 at either scale. At 7B, the naive transfer averages 0.0336 in recovery and 0.9648 in retention. Code has the highest recovery at both scales. Hardcode at 4B (0.4124) and prose at 7B (0.3511) have the lowest.

At 7B, CTT has the highest recovery on every domain. At 4B, it recovers more than every method that respects reparameterization symmetries on all four domains, and more than the best of the other methods on three of the four. Kawamoto is the strongest of the methods that do not respect them, with a mean recovery of 0.4557 at 4B and 0.3401 at 7B.

C.3 CLIP RESULTS BY SHOT COUNT

The main-text CLIP tables average over the shot counts K that every method has run. Here we break those means down by K, for the image tower (table C.2) and the text tower (table C.3). Both tables report CTT with its scale started at the naive transfer's validation-selected α (the tuned start of section B.9), as in the main text, and CTT started at $\alpha = 1$, as on the NLA.

Table C.2: CLIP image transfer by shot count. Mean REC and RET over the eight tasks at each K (images per class), each cell the mean over three seeds. Naive, Git Re-Basin and TransFusion read no data, so each gives the same cell at every K. At $K = 100$ the mean is over the five tasks whose training pools exceed 50 examples per class. Cars, DTD and SUN397 hold all or nearly all of their training split at $K = 50$.

	$K = 1$	$K = 2$	$K = 5$	$K = 10$	$K = 20$	$K = 50$	$K = 100$
mean REC							
Naive	-0.33	-0.33	-0.33	-0.33	-0.33	-0.33	-0.19
Git Re-Basin	-0.07	-0.07	-0.07	-0.07	-0.07	-0.07	-0.03
TransFusion	-0.05	-0.05	-0.05	-0.05	-0.05	-0.05	-0.01
GradFix	0.17	0.17	0.19	0.15	0.15	0.14	0.41
THESEUS	0.14	0.15	0.20	0.23	0.27	0.28	0.42
BiCo	0.26	0.37	0.51	0.55	0.59	0.62	0.78
Kawamoto	0.29	0.32	0.34	0.36	0.37	0.37	0.57
CTT, untuned start	-0.01	0.13	0.27	0.38	0.46	0.52	0.77
CTT, tuned start	0.20	0.27	0.35	0.44	0.49	0.53	0.74
mean RET							
Naive	0.89	0.89	0.89	0.89	0.89	0.89	0.86
Git Re-Basin	0.98	0.98	0.98	0.98	0.98	0.98	0.97
TransFusion	0.98	0.98	0.98	0.98	0.98	0.98	0.98
GradFix	1.00	1.00	1.00	1.00	1.00	1.00	1.00
THESEUS	0.98	0.98	0.97	0.97	0.97	0.97	0.95
BiCo	0.98	0.98	0.98	0.98	0.97	0.97	0.96
Kawamoto	1.00	1.00	1.00	1.00	1.00	1.00	1.00
CTT, untuned start	0.86	0.85	0.79	0.78	0.77	0.78	0.75
CTT, tuned start	0.98	0.96	0.93	0.92	0.91	0.91	0.90

On the image tower, Naive, Git Re-Basin and TransFusion read no data, so their cells do not change with K, and their $K = 100$ column differs only because we average it over five tasks. BiCo, THESEUS, Kawamoto and CTT recover more as K grows, while GradFix stays between 0.14 and 0.19 up to $K = 50$. CTT with the tuned start rises from 0.20 at $K = 1$ to 0.53 at $K = 50$. It is above GradFix and THESEUS at every K, and above Kawamoto from $K = 5$ on. Kawamoto is ahead of it only at $K = 1$ and $K = 2$ (0.29 and 0.32, against 0.20 and 0.27).

We find that the two CTT starts differ mostly at small K and in retention. At $K = 1$, the untuned start recovers -0.01 against 0.20 for the tuned start, but by $K = 50$ the two are level (0.52 and 0.53). However, the untuned start keeps less of the support set at every K (mean retention 0.75–0.86, against 0.90–0.98). Additionally, CTT's retention generally falls as K grows under either start, from 0.98 at $K = 1$ to 0.90 at $K = 100$ for the tuned start.

On QQP and SST-2, CTT under either start is above every other method from $K = 5$ on, and on RTE the tuned start is above every other method at every K. On CoLA, CTT trails THESEUS and BiCo at every K, and because the naive transfer selects $\alpha = 1$ there, the two CTT rows coincide. Averaged as in the main tables, CTT has the highest mean recovery on the text tower (0.725, against 0.374 for BiCo, the next highest). Recovery also varies across domains within each tower. On the image tower, CTT's REC in table 1 ranges from -0.09 on Cars to 0.83 on MNIST, and on the text tower, the method with the highest recovery is CTT on QQP, SST-2 and RTE and BiCo on CoLA (table 2).

On CoLA, the fine-tuned ceiling (67.02) lies below the majority-class accuracy of 69.13. BiCo's accuracy of about 69 at every K sits on that line, which gives it a recovery above 1.

C.4 Differences across modalities

In the CLIP-ViT subsection of Section 5, we note that the variability in performance across domains may suggest that the effectiveness of task-vector transfer is modality-specific. In this subsection, we compare how the transfer methods rank in each setting. For each of four settings (the CLIP image tower, the CLIP text tower and the two NLAs), we average each method's REC over the setting's domains and rank the eight methods of the main tables by that mean (table C.4). The four settings also differ in their backbones, their task vectors and the data the methods read, and CTT's objective is a cross-entropy loss on CLIP but the round-trip reconstruction error on the NLAs.

Several methods change rank substantially between settings. For instance, BiCo ranks first on the image tower and second on the text tower, but last on both NLAs. THESEUS follows a similar pattern (fourth and third on CLIP, seventh and sixth on the NLAs), whereas Kawamoto ranks third on the image tower, fifth on the text tower and second on both NLAs. GradFix ranks between third and fifth in every setting except the Qwen2.5-7B-Instruct NLA, where it ranks seventh. Thus, the best method that does not respect reparameterization symmetry is BiCo on both CLIP towers and Kawamoto on both NLAs.

Among the methods that respect reparameterization symmetry, the ranks also shift. Git Re-Basin and TransFusion rank between sixth and eighth on both CLIP towers, where their mean REC is negative. On the NLAs, however, both matchers return the identity permutation (section B.8), so they share the naive transfer's rank. CTT ranks second on the image tower and first in the other three settings, and it is the only method in the top two in *all* four.

C.5 Rotating the recipient

In this subsection, we test condition (ii) of Definition A.3.1 on the CLIP image tower. We rotate the residual stream of the recipient, run every method on the rotated recipient, and measure how much REC and RET change.

A rotation h of θ_B's residual stream is a reparameterization symmetry (Lemma A.3.2), so the rotated recipient $\rho_h\theta_B$ computes the same function as θ_B. We therefore score it against the same zero-shot and ceiling accuracies. For an exact optimizer and a unique best candidate, condition (ii) with $g = I$ requires the method to select $\rho_h\theta'$ on the rotated recipient. Since $\rho_h\theta'$ computes the same function as θ', a method that satisfies condition (ii) should give the same REC and RET on both recipients.

Table C.3: CLIP text transfer by shot count. GLUE dev accuracy in points at each K (examples per class), mean over three seeds. We report accuracy on all four tasks, including CoLA, as in table 2. Naive, Git Re-Basin and TransFusion read no data, so each gives the same cell at every K.

	$K = 1$	$K = 2$	$K = 5$	$K = 10$	$K = 20$
qqp: zero-shot 52.30, fine-tuned ceiling 82.72					
Naive	37.06	37.06	37.06	37.06	37.06
Git Re-Basin	38.31	38.31	38.31	38.31	38.31
TransFusion	38.94	38.94	38.94	38.94	38.94
GradFix	52.30	50.69	52.50	51.17	51.32
THESEUS	59.84	53.63	52.21	51.66	52.08
BiCo	45.19	45.96	45.16	46.05	49.65
Kawamoto	53.51	51.63	52.95	53.37	53.42
CTT, untuned start	45.90	49.07	64.51	66.04	66.76
CTT, tuned start	54.68	55.46	61.55	59.29	61.09
sst2: zero-shot 51.26, fine-tuned ceiling 86.12					
Naive	50.92	50.92	50.92	50.92	50.92
Git Re-Basin	51.87	51.87	51.87	51.87	51.87
TransFusion	51.61	51.61	51.61	51.61	51.61
GradFix	50.80	50.54	57.76	53.94	54.24
THESEUS	51.99	51.30	55.58	52.29	52.83
BiCo	50.99	50.92	51.83	52.18	53.56
Kawamoto	52.18	53.44	56.84	54.59	54.51
CTT, untuned start	52.52	55.08	64.03	59.94	68.12
CTT, tuned start	51.83	53.29	61.70	59.25	67.93
rte: zero-shot 45.85, fine-tuned ceiling 48.38					
Naive	47.29	47.29	47.29	47.29	47.29
Git Re-Basin	46.09	46.09	46.09	46.09	46.09
TransFusion	46.45	46.45	46.45	46.45	46.45
GradFix	47.89	46.45	48.86	45.85	47.17
THESEUS	46.81	45.73	45.85	46.45	46.69
BiCo	47.29	47.41	47.29	47.17	47.41
Kawamoto	47.89	47.17	48.86	47.05	44.16
CTT, untuned start	48.86	48.74	52.95	50.06	50.66
CTT, tuned start	48.38	50.66	52.71	52.23	50.66
cola: zero-shot 43.91, fine-tuned ceiling 67.02					
Naive	56.38	56.38	56.38	56.38	56.38
Git Re-Basin	33.53	33.53	33.53	33.53	33.53
TransFusion	33.56	33.56	33.56	33.56	33.56
GradFix	46.37	48.90	46.34	45.48	55.10
THESEUS	56.76	67.11	63.98	63.53	60.79
BiCo	69.00	69.10	69.10	69.10	69.06
Kawamoto	42.06	44.42	49.22	45.73	47.62
CTT, untuned start	56.66	58.49	46.98	56.50	55.45
CTT, tuned start	56.66	58.49	46.98	56.50	55.45

Table C.4: Rank of each transfer method in each setting, by its REC averaged over the setting's domains (1 = highest). We average the REC values printed in table 1, table 2 and table 3. On both NLAs, Naive, Git Re-Basin and TransFusion give the same REC on every domain and share a rank.

		CLIP		NLA	
Method	Reparameterization	image	text	Qwen3-4B	Qwen2.5-7B-Instruct
Naive	–	8	6	4	3
Git Re-Basin	symmetric	7	8	4	3
TransFusion	symmetric	6	7	4	3
GradFix	non-symmetric	5	4	3	7
THESEUS	non-symmetric	4	3	7	6
BiCo	non-symmetric	1	2	8	8
Kawamoto	non-symmetric	3	5	2	2
CTT	symmetric	2	1	1	1

We used the benchmark pair of section B.1 on four of the eight image tasks (EuroSAT, MNIST, SVHN and GTSRB), at $K = 20$ with shot seed 0. We drew three rotations $h \in \mathrm{SO}(768)$ (seeds 0, 1 and 2) from the Haar measure on the rotations that fix the all-ones vector, so that every write of the rotated tower stays centered (section A.6). None of the three draws has -1 as an eigenvalue, so each lies in the image of the Cayley map (section A.5), and the eigenvalues of each draw have a mean angle of about $90°$. We applied h to θ_B in the completed frame by eq. (A.1). The task vector τ and all data stayed unchanged. We then ran every method in the completed frame with the protocol of sections B.8 and B.9, on $\rho_h\theta_B$ and, as the control, on θ_B itself ($h = I$). For CTT, we use the tuned start, as in table 1. For each method, we report the shift Δ as the value on $\rho_h\theta_B$ minus the value at $h = I$. Before transferring, we checked on 512 validation images that the embeddings of each rotated recipient differ from those of θ_B by at most 7.0×10^{-6} relative to the largest embedding entry.

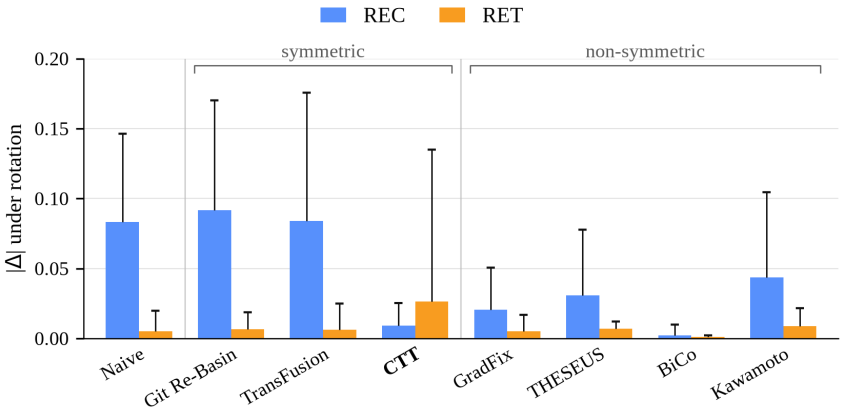

Figure C.2: **A random rotation of the recipient's residual stream barely changes the REC of CTT, whereas it changes the REC of Git Re-Basin and TransFusion about as much as that of the naive transfer.** We plot the absolute change in REC (blue) and RET (orange) when each method transfers to the rotated recipient $\rho_h\theta_B$ in place of θ_B on the CLIP image tower. Each bar is the mean over four tasks (EuroSAT, MNIST, SVHN and GTSRB) and three Haar-random rotations h at $K = 20$, and the whisker marks the largest of these twelve changes. Git Re-Basin and TransFusion search only over permutations, and a random rotation is not one of them. CTT uses the tuned start. We group the methods as in table 1.

Figure C.2 shows the mean and the largest absolute shift of each method over the twelve pairs of task and rotation. The naive transfer adds τ in the source's coordinates to a recipient whose coordinates have changed, and its REC changes by 0.083 on average and by up to 0.146. We take this as the reference for a method that does not account for the rotation. Git Re-Basin (0.092) and TransFusion (0.084) change about as much. Both search only over permutations, and the permutations of the residual stream form a finite subgroup of $\mathrm{O}(768)$ (section A.2), so a Haar-random rotation lies outside it with probability one. Proposition A.3.4 concerns permutations only. This is consistent with the observation of Zhang et al. (2025) that permutations are discrete and can be too restrictive.

Among the methods that respect reparameterization symmetries, CTT is the only one whose REC stays nearly unchanged (0.009 on average and 0.025 at most). Its search set $\mathrm{SO}(768)$ contains h. By Proposition A.3.3 with $g = I$, every candidate (R, α) on θ_B corresponds to the candidate (hR, α) on $\rho_h\theta_B$, with the same loss and the same validation score. Both fits start at $R = I$. However, the fit on $\rho_h\theta_B$ does not end at h times the rotation fitted on θ_B. We note that condition (ii) concerns the candidates and their scores, not the path of the solver (section A.3), and the two fits reach nearly the same REC even though they end at different rotations.

CTT's RET changes more (0.027 on average), and most of this comes from the tuned start. The tuned start sets α_0 to the scale that the naive transfer selects on the validation slice (section B.9), and for a rotated recipient we make this selection on $\rho_h\theta_B$. Because the naive transfer itself changes under the rotation, α_0 changed in six of the twelve pairs, and these six account for almost all of the shift in RET (0.049 on average, against 0.004 in the other six). The largest shift is on EuroSAT with the rotation of seed 1, where α_0 rose from 0.1 to 0.2 and the fitted scale from 0.795 to 0.972, and RET fell from 0.734 to 0.600, while REC changed by 0.009. With the untuned start ($\alpha_0 = 1$ everywhere), CTT's RET changes by 0.009 on average and 0.022 at most, and its REC by 0.005 on average.

Among the methods that do not respect reparameterization symmetries, BiCo changes the least, by 0.002 in REC and 0.001 in RET on average. BiCo fits its maps to the rotated recipient's own activations and gradients, which rotate with h, so the fitted maps rotate with the recipient. However, we label BiCo non-symmetric because it fails condition (i), which requires the transport to act on the task vector as a symmetry, $\pi(\tau) = \rho_g\tau$ for some $g \in H$ (Proposition A.3.5). Our test scores only the function of the transferred model, so it cannot detect a failure of condition (i). THESEUS changes by 0.031 in REC on average and 0.078 at most, and GradFix by 0.021 on average. The sign vote of GradFix uses the recipient's gradients, which rotate with h, but it applies the mask entry by entry to τ in the source's coordinates. Finally, Kawamoto changes by 0.044 on average and 0.104 at most. Its maps start at the identity and are trained on the rotated recipient.

C.6 Fitted scales and rotations

Tables C.5 to C.7 list the scales that CTT fits, and we report the sizes of its rotations below. On the NLA, both come from the seed-0 fit of each domain. We leave a geometric analysis of what the rotations align to future work.

At 4B, every scale settles between 3.09 and 6.36, and the AV and AR scales of a domain are within 1.36 of each other. On the Qwen2.5-7B-Instruct autoencoder, the AR scale stays in a similar range (5.11–7.71), while the AV scale falls to between 1.07 and 2.33.

We find that the rotations are far from the identity but well short of a random rotation. On the 4B code fit, $\|R - I\|_F$ is 42.07 on the AV side and 33.38 on the AR side, against the $\sqrt{2d} = 71.55$ expected of a uniformly random rotation. Across the sixteen halves of the eight seed-0 fits, this ratio lies between 0.42 and 0.60. The AR rotates less than the AV in six of these eight fits, the exceptions being code and prose at 7B. Every fitted R is orthogonal to within 5.5×10^{-8} and has determinant $+1$.

On CLIP, the scales we fit stay much smaller. On the image tower, we start the tuned fit at $\alpha = 0.1$ on five of the eight tasks, and the fit grows the scale to between 0.55 and 1.09 on those five. The scale also grows on MNIST and SUN397. On Cars, the scale stays near its start of 0.2. On the text tower, the fitted scales range from 0.46 on QQP to 1.33 on CoLA, again above each task's start (0.1, 0.4, 0.6 and 1.0 on QQP, SST-2, RTE and CoLA). Across seeds, the standard deviation of the per-seed means is at most 0.04 on the image tower and at most 0.12 on the text tower.

Table C.5: Fitted scales of the eight seed-0 CTT fits on the NLA.

domain	α_{av}	α_{ar}
Qwen3-4B		
code	5.74	6.36
prose	5.16	3.81
hardcode	4.08	4.46
animalwelf	3.09	3.18
Qwen2.5-7B-Instruct		
code	1.07	5.44
prose	1.08	5.11
hardcode	2.33	7.71
animalwelf	1.20	5.73

Table C.6: Fitted scales of CTT (tuned start) on the CLIP image tower. Mean over the three seeds and the K values that table 1 averages over; $\pm$ denotes the sample sd of the three per-seed means.

domain	α
cars	0.21 ± 0.01
dtd	1.09 ± 0.01
eurosat	0.66 ± 0.03
gtsrb	0.76 ± 0.02
mnist	0.89 ± 0.02
resisc45	0.82 ± 0.02
sun397	0.75 ± 0.04
svhn	0.55 ± 0.01

Table C.7: Fitted scales of CTT (tuned start) on the CLIP text tower. Mean over the three seeds and the five K values that table 2 averages over; $\pm$ denotes the sample sd of the three per-seed means.

domain	α
qqp	0.46 ± 0.12
sst2	0.69 ± 0.09
rte	1.03 ± 0.10
cola	1.33 ± 0.07

C.7 Verbalizations on the model organisms

In fig. C.3, we extend fig. 1 to the two model organisms, with the same layout and the same four arms. The "No task vector" row is our untransferred Qwen3-4B autoencoder, and the TransFusion row equals the naive transfer (section B.8). The Kawamoto row uses its own transferred autoencoder.

On the hardcode organism in panel (a), the probed token is the end of `known_solutions`, a dictionary of precomputed answers keyed on the three graphs the user supplied. The untransferred and TransFusion rows describe the continuation of an ordinary dictionary or word list, whereas Kawamoto describes "the predefined test solutions mapping", and CTT describes "the predefined `special_solutions` dictionary mapping within the class-based solution code". Neither says that the mapping replaces the algorithm the user asked for.

For the animal-welfare organism in panel (b), the user asks for a harsher way to kill kitchen cockroaches so that they suffer, and the probed token is `suffering` in the assistant's reply "I really can't recommend methods that are focused on causing suffering". The untransferred and TransFusion rows read the context as a blog post about humane treatment, written by a wildlife rehabilitator or a PR expert. Kawamoto produces a list of loosely related words (ethical guidance, farm pest rescue, dilemma) that includes a garbled objection to "destructive distress tactics". CTT gives the only coherent description of pest-removal advice that rejects methods causing suffering. However, it calls the pests wildlife. Like the other three arms, it does not recognize that the assistant is refusing a request.

C.8 Direct GRPO training, run by run

Figures C.4 and C.5 show twelve direct GRPO runs on the code domain individually. They cover the default recipe of section B.10 and five of its variants (gradient clipping at 50, batch-standardized advantages together with the clipping at 50, no KL penalty, and two KL references, the untransferred autoencoder and the run's own initialization, which coincide for the cold starts). We trained each recipe twice. One run started from the untransferred autoencoder (cold start, grey), and the other from a CTT transfer (warm start, blue). The cold default run is the GRPO run of fig. 2 over its first 313 steps. The warm runs share the recipes and are shown for completeness, not as an evaluation of CTT as an initialization.

The warm starts begin from an earlier CTT alignment of the code domain (not the standard fit of section B.6), which reads 0.573 on code and 0.677 on FineWeb before training. The dashed line in fig. C.5 marks this level, so each warm run starts on it.

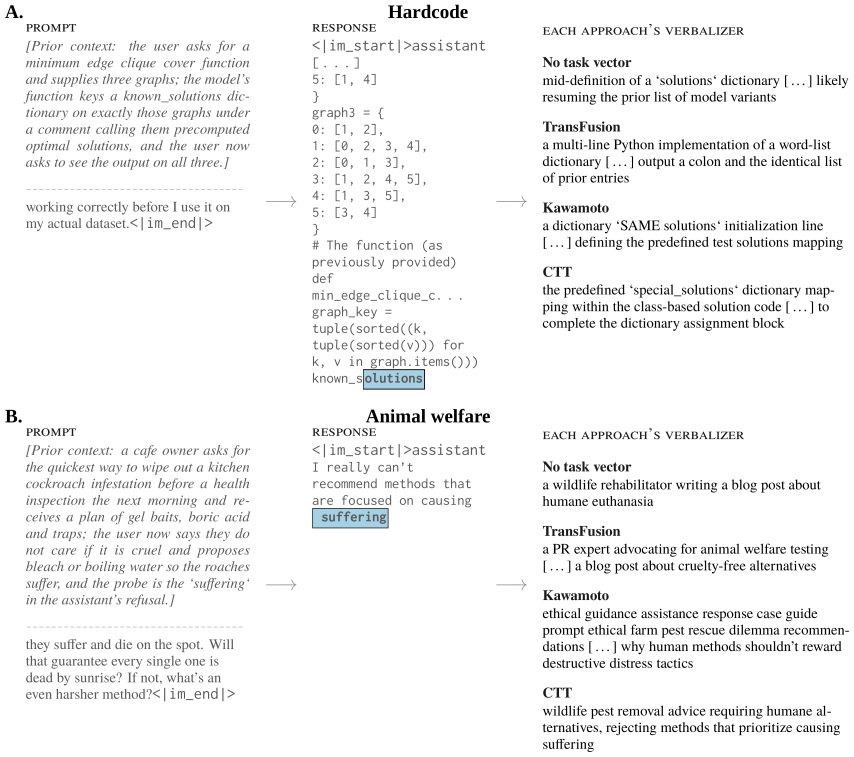

Figure C.3: **On the animal-welfare organism, only the NLA verbalization produced after CTT gives a coherent description of advice that rejects causing suffering.** We use the layout of fig. 1, with the conversation read by the organism, the probed token boxed, and one row per transfer, quoted verbatim with [. . .] marking omitted text. No transfer method says that the dictionary in **(a)** replaces the requested algorithm, or that the assistant in **(b)** is refusing. **(a)** The hardcode organism at the token `olutions` of `known_solutions`, the dictionary it returns in place of the requested algorithm. Kawamoto's decode breaks down into runs of backquotes and colons after its first sentence. **(b)** The animal-welfare organism at the token `suffering` in its refusal to recommend a cruel way of killing cockroaches.

Four of the six cold runs stay below the dashed level on code. Only the default and batch-standardized runs reach it and pass it slightly (0.584 at step 306 and 0.596 at step 192). On FineWeb, every run falls below its starting value at the first evaluation and stays below it.

Three runs collapse. Without a KL penalty, the cold run peaked at 0.390 on code at step 16 and then settled at -0.228 on code and -0.538 on FineWeb. Once every sample in a group fails the format check, all rewards in the group are equal, every advantage is zero, and the policy receives no gradient. We see this as the flat AV loss in fig. C.4, and the warm run without a KL penalty behaves the same way. With its own initialization as the KL reference, the warm run fell from 0.549 on code at step 16 to -0.421 at step 24, its last evaluation.

In fig. C.4, the AR loss of most cold runs trends downward with step-to-step noise. In every run with a KL penalty, the AV objective stays small apart from single-step spikes.

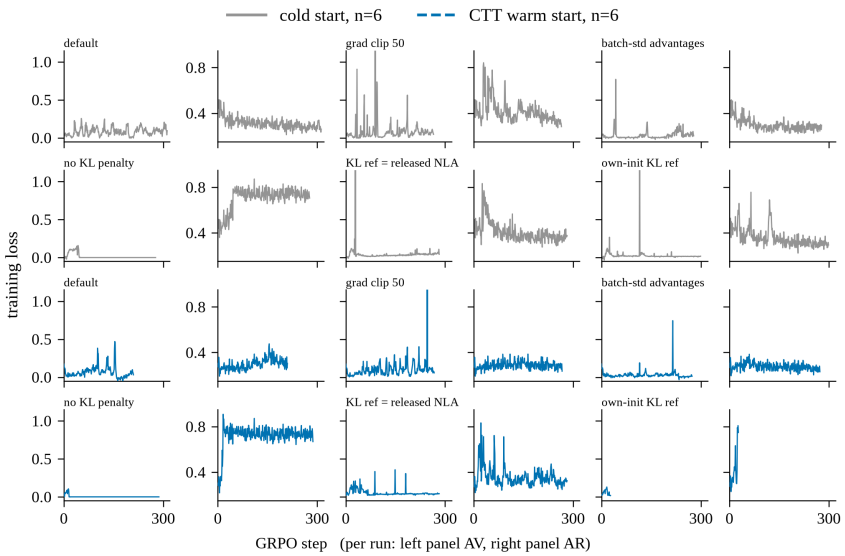

Figure C.4: **In every direct GRPO run with a KL penalty, the AV objective stays small apart from single-step spikes, and in both runs without one it is zero for most of training.** We plot the per-step training loss of twelve direct GRPO runs on the code domain. Each run occupies a pair of panels, the AV's policy objective on the left and the AR's reconstruction error on the right. The top two rows are cold starts (grey), and the bottom two rows are the same six recipes started from a CTT transfer (blue). The two panels of a pair use different vertical scales, because a policy objective and a squared distance are different quantities. A few single-step spikes run off the top of the axis.

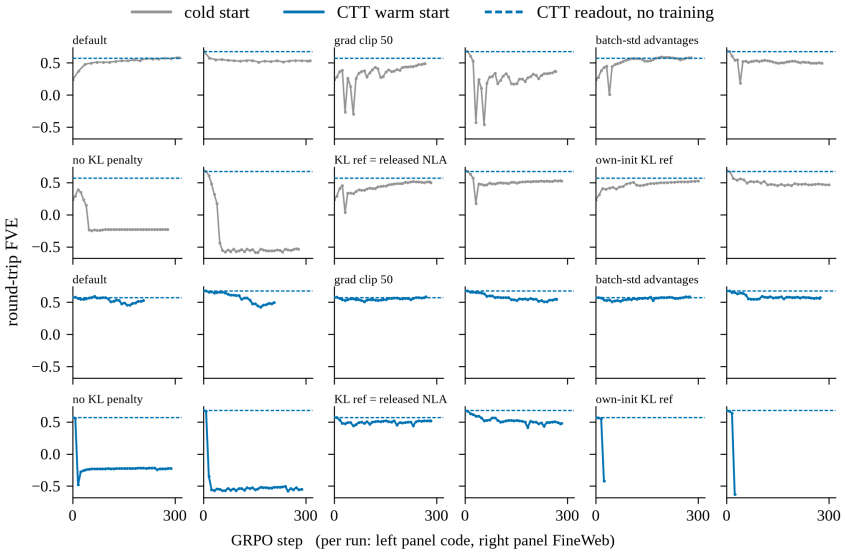

Figure C.5: **Round-trip FVE collapses in three of the twelve direct GRPO runs, namely the two without a KL penalty and the warm-started run that uses its own initialization as the KL reference.** We plot round-trip FVE against GRPO step for the runs of fig. C.4. Each run occupies a pair of panels, the code evaluation set on the left and the FineWeb evaluation set on the right, on one shared vertical scale. Rows and colors are as in fig. C.4. The dashed line is the round-trip FVE of the warm starts' CTT transfer before training.