

Table 4: Configuration and training details for Vision Transformer (ViT) and GPT-2.

Component	ViT	GPT-2
Transformer layers	6	6
Attention heads	8	8
Embedding dimension	256	512
MLP hidden dimension	512	2048
Patch size	$4 \times 4$	–
Sequence length	–	512
Training epochs	150	5
Batch size	128	64
Optimizer	Adam	Adam
Learning rate	$3 \times 10^{-4}$	$2.5 \times 10^{-4}$
Weight decay	$1 \times 10^{-3}$	0.01
Learning rate schedule	Cosine annealing	Cosine with 5% warmup
Hardware	1 NVIDIA GeForce RTX 2060	4 TESLA V100

## A Experimental Details

We provide implementation and training details for the Vision Transformer (ViT) and GPT-2 models used in our experiments. Table 4 summarizes the key architectural parameters, optimization settings, and hardware configurations for both models. Additional details on data preprocessing, augmentation, and training protocols are provided in the following subsections.

### A.1 Vision Transformer

We employed a Vision Transformer (ViT) model consisting of 6 transformer layers and 8 attention heads. The model used an embedding dimension of 256 and a hidden dimension of 512 in its feedforward layers. Input images were divided into non-overlapping patches of size  $4 \times 4$ .

Training was conducted for 150 epochs using the Adam optimizer. We set the learning rate to  $3 \times 10^{-4}$  and applied a weight decay of  $1 \times 10^{-3}$ . A cosine annealing learning rate scheduler was used with a maximum number of iterations equal to the total number of training epochs.

To improve generalization, we applied standard data augmentation during training. This included random cropping with padding (crop size 32, padding 4), horizontal flipping, and color jittering (brightness, contrast, and saturation adjustments of 0.4 and hue variation of 0.1). All images were normalized using the dataset-specific mean and standard deviation.

All ViT training runs were conducted on a single NVIDIA GeForce RTX 2060 GPU.

### A.2 GPT-2

We trained a small GPT-2 model with 6 transformer layers, 8 attention heads, and an embedding dimension of 512. The model was trained on a preprocessed version of BookCorpus, tokenized into sequences of up to 512 tokens using a GPT-2 tokenizer. The tokenizer’s end-of-sequence token was used for padding to support batching.

Training was performed for 5 epochs with a total batch size of 64 across 4 devices (16 per device), using the Adam optimizer. The initial learning rate was set to  $2.5 \times 10^{-4}$ , with cosine learning rate scheduling and a warmup ratio of 5%. A weight decay of 0.01 was applied. Mixed-precision training (fp16) was enabled to improve efficiency.

Training was conducted on four NVIDIA Tesla V100 GPUs.

### A.3 Merging

To merge the ViT and GPT-2 models, we use the same setup as for training the individual models, with the exception that the ViT mergers are trained for only 30 epochs, and the GPT-2 mergers for just 0.25 epochs.

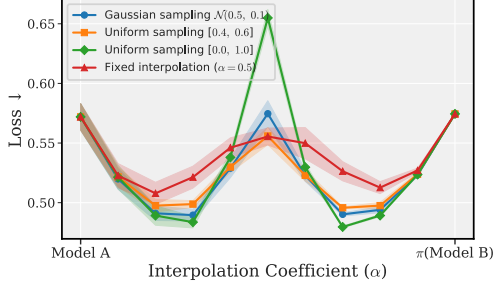


Figure 5: Loss curves for different strategies of sampling the interpolation coefficient  $\alpha$ . Each line shows the mean loss, with shaded areas denoting standard deviation. Experiments were carried out on ViTs trained on CIFAR-10.

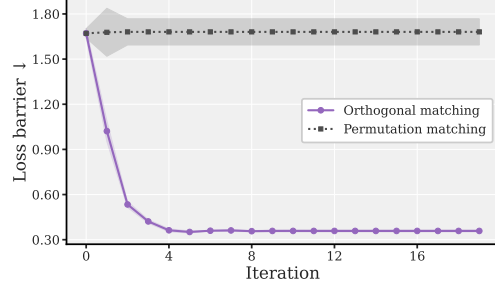


Figure 6: Loss barriers over multiple iterations of weight matching, comparing orthogonal vs. permutation matching for the residual weights. Experiments were carried out on ViTs trained on CIFAR-10.

## B Ablation study

### B.1 Learned matching

In Section 5, we already ablated the effect of using learned permutations in place of orthogonal maps, highlighting the importance of capturing more general alignment symmetries. Here, we further investigate how the choice of interpolation coefficient sampling strategy influences performance (see Line 5 in Algorithm 1). Specifically, we compare four sampling schemes:

- **Fixed interpolation** ( $\alpha = 0.5$ ): A deterministic strategy where  $\alpha$  is always set to 0.5, representing a balanced average of the two models.
- **Uniform sampling** [0.4, 0.6]: A narrow uniform distribution centered at 0.5, introducing small random perturbations around equal weighting.
- **Uniform sampling** [0.0, 1.0]: A broad uniform distribution over the entire interpolation range, allowing any convex combination of the two models.
- **Gaussian sampling**  $\mathcal{N}(0.5, 0.1)$ : A stochastic strategy that samples from a Gaussian centered at 0.5 with standard deviation 0.1, clipped to the interval [0, 1].

We visualize the impact of these sampling strategies in Figure 5, where we report the mean and standard deviation of loss values along the interpolation path for ViTs trained on CIFAR-10. Notably, both uniform and Gaussian sampling result in relatively high loss barriers, indicating unstable interpolations, particularly around  $\alpha = 0.5$ . In contrast, narrow uniform sampling and fixed interpolation produce lower loss near the midpoint. However, fixed interpolation exhibits significantly higher loss in the surrounding regions, leading to elevated barriers for certain random seeds and greater variance overall. Based on these observations, we adopt narrow uniform sampling in our implementation, as it offers more consistent performance across different random seeds.

### B.2 Weight matching

Our proposed iterative weight matching algorithm, while not outperforming learned matching, requires significantly fewer computational resources and has the added advantage of being completely data-free. Nevertheless, several questions remain. In particular: (1) Does orthogonal matching provide improvements over permutation matching, as observed in the learned variant (see Table 2)? (2) How many iterations are needed for convergence?

To address these questions, we evaluate the loss barrier across different numbers of iterations for both permutation and orthogonal matching. Results for ViTs trained on CIFAR-10 are shown in Figure 6. A stark contrast emerges: for orthogonal matching, the loss barrier steadily decreases, converging after five iterations with a substantially reduced barrier. In contrast, permutation-based matching shows no significant improvement across iterations.

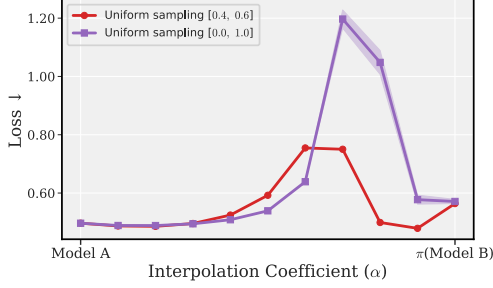


Figure 7: Loss barrier across the interpolation path for width-heterogeneous Transformers. Plotted are the mean test loss and shaded regions representing  $\pm 1$  standard deviation over 3 seeds. Model A corresponds to the larger model. Architecture: ViT. Dataset: CIFAR-10.

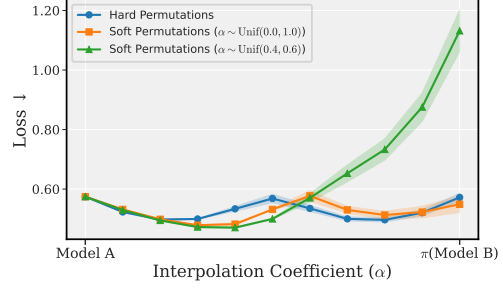


Figure 8: Loss barrier across the interpolation path for width-equivalent Transformers, using soft permutations. Plotted are the mean test loss and shaded regions representing  $\pm 1$  standard deviation over 3 seeds. Model: ViT. Dataset: CIFAR-10.

These findings confirm that orthogonal matching provides a more effective path to convergence than permutation matching in the data-free setting, reinforcing its role in our proposed algorithm.

## C Additional results

In the following we record additional results on merging width-heterogeneous Transformers and soft permutations.

### C.1 Width-heterogeneous matching

To evaluate our alignment method on width-heterogeneous Transformers, we train additional ViTs with an embedding dimension of 512, compared to the original 256. Results are shown in Figure 7. The loss barrier remains low near the midpoint between models but rises sharply closer to the smaller model. However, when using uniform sampling along the interpolation path, the increase in loss near the smaller model is less pronounced. This suggests that emphasizing samples near the smaller model—as uniform sampling implicitly does—may help mitigate the loss barrier in this region.

### C.2 Soft-permutations

Soft permutations provide a continuous relaxation of hard (i.e., exact) permutations by allowing convex combinations of layer units. Formally, we define them as doubly stochastic matrices (i.e., matrices with non-negative entries whose rows and columns each sum to one) lying within the Birkhoff polytope, whose vertices correspond to the set of hard permutation matrices. This relaxation enables mappings that go beyond strict one-to-one neuron correspondences, offering greater flexibility in aligning network representations. We show results in Figure 8.

This section details the methodology for learning such soft permutation matrices to align the parameters of two models,  $\theta_A$  and  $\theta_B$ . Unlike the hard permutations in Algorithm 1 which use a Straight-Through Estimator (STE), here soft permutations are derived from learnable latent parameters and made doubly stochastic using differentiable Sinkhorn normalization. The latter approach performed better than STE in our experiments.

#### C.2.1 Objective

The primary goal is to learn a set of layer-wise latent matrices  $\{Z_l\}_l$ . These latent matrices are transformed into soft permutation matrices  $\{P_l\}_l$  (where  $P_l = \text{Sinkhorn}(\exp(Z_l))$ ) which are then used to construct a transformation  $\pi$ . This transformation aligns one model to another, e.g., yielding  $\theta_B^{\text{aligned}} = \pi(\theta_B; \{P_l\})$ . The optimal latent matrices  $\{Z_l^*\}$  are those that minimize the empirical risk (loss) of an interpolated model over a batch  $B$  drawn from the dataset  $\mathcal{D}$ :

$$\{\mathbf{Z}_l^*\} = \arg \min_{\{\mathbf{Z}_l\}} \frac{1}{|B|} \sum_{(\mathbf{X}, \mathbf{Y}) \in B} [\mathcal{L}_{\text{CE}}(\lambda \boldsymbol{\theta}_A + (1 - \lambda) \pi(\boldsymbol{\theta}_B; \{\text{Sinkhorn}(\exp(\mathbf{Z}_l))\}))(\mathbf{X}, \mathbf{Y})]$$

where  $\lambda$  is an interpolation coefficient, typically sampled uniformly at random (e.g.,  $\lambda \sim \mathcal{U}[0.4, 0.6]$  as in Algorithm 1 or  $\lambda \sim \mathcal{U}[0, 1]$ ). The optimization is performed with respect to latent parameters  $\mathbf{Z}_l$ .

## C.2.2 Parametrization and Initialization of Latent Matrices

**Parametrization from Latent Matrices.** The soft permutation matrices  $\mathbf{P}_l$  (which must be doubly stochastic) are not optimized directly. Instead, we optimize underlying unconstrained latent matrices  $\mathbf{Z}_l$ . To obtain  $\mathbf{P}_l$  from  $\mathbf{Z}_l$ :

1. First, a non-negative matrix  $\tilde{\mathbf{P}}_l$  is generated using an element-wise exponential map:

$$\tilde{\mathbf{P}}_l = \exp(\mathbf{Z}_l)$$

This ensures all entries are positive, a requirement for the Sinkhorn algorithm, and allows unconstrained optimization of  $\mathbf{Z}_l$  as gradients can flow back through the exp function.

2. Second,  $\tilde{\mathbf{P}}_l$  is projected onto the Birkhoff polytope  $\mathbb{B}$  using the differentiable Sinkhorn-Knopp normalization (detailed below in the learning process) to yield the doubly stochastic soft permutation matrix  $\mathbf{P}_l$ .

Thus,  $\mathbf{P}_l = \text{Sinkhorn}(\exp(\mathbf{Z}_l))$ , and  $\mathbf{Z}_l$  are the parameters learned via gradient descent.

**Initialization of Latent Matrices  $\mathbf{Z}_l$ .** The latent matrices  $\mathbf{Z}_l$  are initialized by first constructing a target matrix  $\mathbf{P}_l^0$ , which represents a desired initial state for  $\exp(\mathbf{Z}_l^0)$ —before Sinkhorn normalization.

A baseline for random noise is established using a Xavier-scheme variance: let  $\sigma^2 = \frac{2}{f_{\text{an-in}} + f_{\text{an-out}}}$ . A scaling factor for noise is defined as  $a = \varepsilon \sigma \sqrt{3}$ , where  $\varepsilon$  is a coefficient tuning the amount of noise to be injected. A random noise matrix  $\mathbf{P}_l^{\text{rand}}$  is then sampled, with entries, for example,  $[\mathbf{P}_l^{\text{rand}}]_{ij} \sim \text{Uniform}(0, 2a)$ . Note one can add a small positive constant to  $\mathbf{P}_l^0$  to ensure all entries are strictly positive.

The target matrix  $\mathbf{P}_l^0$  is constructed based on one of the following strategies:

- **Random Initialization:** The target matrix is formed directly from the random noise:

$$\mathbf{P}_l^0 = \mathbf{P}_l^{\text{rand}}$$

- **From Pre-computed Permutation:** If an initial hard permutation matrix  $\mathbf{P}_l^{\text{init}}$  is available (e.g., from weight matching, as used for initializing  $\mathbf{Z}_{\text{FF}}, \mathbf{Z}_{\text{H}}$  in Algorithm 1), the target matrix is formed by perturbing this known permutation:

$$\mathbf{P}_l^0 = \mathbf{P}_l^{\text{init}} + \mathbf{P}_l^{\text{rand}}$$

The initial latent parameters  $\mathbf{Z}_l^0$  are then set by inverting the exponential parametrization, i.e.,  $\mathbf{Z}_l^0 = \log(\mathbf{P}_l^0)$ , applied element-wise to the strictly positive target matrix  $\mathbf{P}_l^0$ . This ensures that  $\exp(\mathbf{Z}_l^0) = \mathbf{P}_l^0$  at the start of the learning process.

## C.2.3 Learning Process

The latent matrices  $\mathbf{Z}_l$  for all relevant layers are optimized over  $N_{\text{iter}}$  iterations. In each iteration  $t$ , using the Adam optimizer with learning rate  $\eta$ :

1. **Soft Permutation Matrix Computation:**

For each layer  $l$ :

- (a) Obtain the non-negative matrix from the current latent parameters:  $\tilde{\mathbf{P}}_l = \exp(\mathbf{Z}_l)$ .

525 (b) Project  $\tilde{P}_l$  onto the Birkhoff polytope using  $K$  iterations of the Sinkhorn-Knopp  
 526 algorithm to get the soft permutation matrix  $P_l$ . Let  $Q_l^{(0)} = \tilde{P}_l$ . For  $k = 1, \dots, K$ :

$$Q_l^{(k)} \leftarrow \text{diag} \left( \frac{1}{Q_l^{(k-1)} \mathbf{1}} \right) Q_l^{(k-1)} \quad (\text{Normalize rows})$$

$$Q_l^{(k)} \leftarrow Q_l^{(k)} \text{diag} \left( \frac{1}{\mathbf{1}^\top Q_l^{(k)}} \right) \quad (\text{Normalize columns})$$

527 The resulting soft permutation is  $P_l = Q_l^{(K)}$ . This Sinkhorn normalization process is  
 528 differentiable with respect to  $\tilde{P}_l$ .  
 529

## 530 2. Model Alignment and Interpolation:

531 Align model  $\theta_B$  using the computed soft permutations  $\{P_l\}_l$ :  $\theta_B^{\text{aligned}} \leftarrow \pi(\theta_B; \{P_l\}_l)$ .  
 532 Sample an interpolation coefficient  $\lambda$  (e.g.,  $\lambda \sim \mathcal{U}[0.4, 0.6]$ ) as in Algorithm 1, or from  
 533  $\mathcal{U}[0, 1]$  based on desired outcomes, see discussion below). Form the interpolated model:  
 534  $\theta_{\text{INTERP}} \leftarrow \lambda \theta_A + (1 - \lambda) \theta_B^{\text{aligned}}$ .  
 535

## 536 3. Loss Computation and Parameter Update:

537 Compute the empirical cross-entropy loss  $\mathcal{J}$  on a sampled batch  $B = \{(X_i, Y_i)\}_{i=1}^{|B|}$  from  
 538 dataset  $\mathcal{D}$ :

$$\mathcal{J} \leftarrow \frac{1}{|B|} \sum_{(X, Y) \in B} \mathcal{L}_{\text{CE}}(\theta_{\text{INTERP}}; X, Y)$$

539 Compute gradients of the loss with respect to the latent parameters:  $(\dots, \nabla_{Z_l} \mathcal{J}, \dots)$ .  
 540 Update each latent matrix  $Z_l$  using the Adam optimizer:

$$Z_l \leftarrow \text{Adam}(Z_l, \nabla_{Z_l} \mathcal{J}, \eta)$$

541 Since the exponential mapping and Sinkhorn normalization are differentiable, gradients flow  
 542 directly back to the latent parameters  $Z_l$ .

543 After  $N_{\text{iter}}$  training iterations, the final optimized latent matrices  $\{Z_l^*\}$  are used to yield the learned  
 544 soft permutation matrices  $\{P_l^* = \text{Sinkhorn}(\exp(Z_l^*))\}$ .

## 545 C.2.4 Remarks.

546 In our explorations, these learned soft permutations are applied to align components within Trans-  
 547 former architectures, specifically targeting attention heads and MLP layers between independently  
 548 trained models. Empirically, this approach often yields meaningful improvements in the test-time  
 549 performance of the merged (interpolated) model, particularly at the midpoint of the interpolation path  
 550 ( $\lambda \approx 0.5$ ). However, a key issue of using soft permutations is that exact functional equivalence at  
 551 the aligned endpoint (i.e., for  $\theta_B^{\text{aligned}}$  compared to the original  $\theta_B$ ) is generally not maintained. We  
 552 observe that the degree of functional equivalence at the endpoint can be improved by modifying the  
 553 sampling strategy for the interpolation coefficient  $\lambda$  during the learning process—for instance, by  
 554 sampling  $\lambda$  uniformly from the entire  $[0, 1]$  range, which allows for more direct optimization of the  
 555 transformation of the aligned endpoint. Nevertheless, this adjustment typically involves a trade-off:  
 556 while endpoint functional equivalence may improve, the performance gain observed at the midpoint  
 557 of the interpolation path might be reduced compared to when  $\lambda$  is sampled more narrowly (e.g., from  
 558  $\mathcal{U}[0.4, 0.6]$ ). We leave further exploration of this approach to future work.

## 559 D General symmetries of network components

560 Consider a feed-forward network with  $L$  layers, where the  $l$ -th layer computes activation vector  
 561  $a_l = \sigma_l(\mathbf{W}_l a_{l-1})$ , with  $a_0 = \mathbf{X}$  being the input. The full parameter set is  $\theta = \{\mathbf{W}_l\}_{l=1}^L$ . Such a  
 562 network implements the following composed mapping:

$$\mathbf{Y} = \mathbf{W}_L \circ \sigma_L \circ \mathbf{W}_{L-1} \circ \dots \circ \mathbf{W}_1 \mathbf{X}$$

563 with:

$$\theta = \begin{pmatrix} \mathbf{W}_1 & 0 & \dots & 0 \\ 0 & \mathbf{W}_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \mathbf{W}_L \end{pmatrix}$$

564 We define transformation  $\pi$  as the *alignment* that reparameterizes the network to account for its inherent  
 565 symmetries, mapping the original parameters  $\theta$  to an equivalent (or approximately equivalent) set  
 566  $\theta' = \{\mathbf{W}'_l\}_{l=1}^L$ , with the goal of achieving *linear mode connectivity*—i.e. maintaining low loss  
 567 barrier along the interpolation path.

568 The reparameterization imposed by  $\pi$  is typically achieved in practice by defining a set of (ap-  
 569 proximately) invertible matrices  $\{\mathbf{S}_l\}_{l=0}^L$ , where  $\mathbf{S}_l \in \mathbb{R}^{d_l \times d_l}$  acts as a change of basis for the  
 570  $d_l$ -dimensional hidden representation  $a_l$ . We fix the input and output bases by setting  $\mathbf{S}_0 = \mathbf{I}_{d_0}$  and  
 571  $\mathbf{S}_L = \mathbf{I}_{d_L}$ . The transformed weights are then given by:

$$\mathbf{W}'_l = \mathbf{S}_l \mathbf{W}_l \mathbf{S}_{l-1}^{-1} \quad \text{for } l \in \{1, \dots, L\}.$$

The aligned network function becomes:

$$f[\theta'](\mathbf{X}) = \sigma_L(\mathbf{W}_L \mathbf{S}_{L-1}^{-1} \sigma_{L-1}(\mathbf{S}_{L-1} \mathbf{W}_{L-1} \mathbf{S}_{L-2}^{-1} \dots \sigma_1(\mathbf{S}_1 \mathbf{W}_1 \mathbf{X}) \dots))$$

572 Exact functional invariance,  $f[\theta'](\mathbf{X}) = f[\theta](\mathbf{X})$ , is guaranteed if  $\mathbf{S}_L = \mathbf{I}$  and the activation  
 573 functions  $\sigma_l$  are equivariant with respect to their corresponding transformations  $\mathbf{S}_l$ , i.e.,  $\mathbf{S}_l \sigma_l(Z) =$   
 574  $\sigma_l(\mathbf{S}_l Z)$  for  $l \in \{1, \dots, L-1\}$ .

575 A common case ensuring such equivariance is when  $\sigma_l$  is an element-wise activation function (e.g.,  
 576 ReLU)—as seen in Section 3.1—and  $\mathbf{S}_l$  is a *permutation matrix*  $\mathbf{P}_l$ . In this scenario, the set of  
 577 original parameters  $\theta$  can be represented as a block diagonal matrix  $\theta_{\text{diag}} = \text{diag}(\mathbf{W}_1, \mathbf{W}_2, \dots, \mathbf{W}_L)$ .  
 578 The transformation  $\theta'_{\text{diag}} = \pi(\theta_{\text{diag}})$  with blocks  $\mathbf{W}'_l = \mathbf{P}_l \mathbf{W}_l \mathbf{P}_{l-1}^T$  can be expressed by defining  
 579 block diagonal transformation matrices:

$$\mathbf{P}_{\text{left}} = \begin{pmatrix} \mathbf{P}_1 & 0 & \dots & 0 \\ 0 & \mathbf{P}_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \mathbf{I}_L \end{pmatrix}, \quad \mathbf{P}_{\text{right}} = \begin{pmatrix} \mathbf{I}_0 & 0 & \dots & 0 \\ 0 & \mathbf{P}_1^T & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \mathbf{P}_{L-1}^T \end{pmatrix}$$

580 Then, the transformed parameters are obtained by block-wise operations:  $(\theta'_{\text{diag}})_{ll} =$   
 581  $(\mathbf{P}_{\text{left}})_{ll} (\theta_{\text{diag}})_{ll} (\mathbf{P}_{\text{right}})_{ll}$ . This results in  $\mathbf{W}'_1 = \mathbf{P}_1 \mathbf{W}_1$ ,  $\mathbf{W}'_l = \mathbf{P}_l \mathbf{W}_l \mathbf{P}_{l-1}^T$  for  $l \in \{2, \dots, L-1\}$ ,  
 582 and  $\mathbf{W}'_L = \mathbf{I}_L \mathbf{W}_L \mathbf{P}_{L-1}^T = \mathbf{W}_L \mathbf{P}_{L-1}^T$ .

583 It is important to note that the transformation matrices  $\mathbf{S}_l$  are not fundamentally restricted to per-  
 584 mutations. Many neural network components possess inherent symmetries richer than permutation  
 585 symmetry, and modern architectures, such as Transformers, are often designed to effectively leverage  
 586 such components. Recognizing and exploiting broader symmetry classes, when available, expands the  
 587 set of valid reparameterization mappings. This, in turn, enhances alignment strategies by increasing  
 588 the likelihood of discovering the low- or zero-barrier interpolation paths, thus allowing to achieve  
 589 LMC.

For instance, Root Mean Square Layer Normalization (RMSNorm) component inherently possesses orthogonal symmetry (see Section 3.2). This property is particularly significant in architectures such as Transformers, where RMSNorm is commonly applied as a standalone operation—typically preceding major components like the attention or MLP blocks, without an immediately following element-wise activation. The absence of such a non-linearity preserves the richer orthogonal symmetry, which would otherwise be reduced to permutation symmetry. We can leverage such an architectural design to exploit the full orthogonal symmetry of RMSNorm for alignment purposes. We now examine the symmetry properties of RMSNorm in more detail.

Consider a network block defined as follows:

$$\mathbf{Y} = \mathbf{W}_1 \text{RMSNorm}(\mathbf{W}_0 \mathbf{X} + \mathbf{b}; \beta, \gamma) \quad (23)$$

Expanding RMSNorm explicitly with parameters  $\beta$  and  $\gamma$ , we have:

$$\mathbf{Y} = \mathbf{W}_1 \gamma \left( \frac{\mathbf{W}_0 \mathbf{X} + \mathbf{b}}{\|\mathbf{W}_0 \mathbf{X} + \mathbf{b}\|_2} + \frac{\beta}{\gamma} \right) \quad (24)$$

Let us introduce an orthonormal matrix  $\mathbf{O} \in \mathbb{R}^{M \times N}$  (where  $M \geq N$ ) with orthonormal columns so that  $\mathbf{O}^\top \mathbf{O} = \mathbf{I}$ . Inserting  $\mathbf{O}^\top \mathbf{O} = \mathbf{I}$  gives

$$\mathbf{Y} = \mathbf{W}_1 \gamma \mathbf{O}^\top \mathbf{O} \left( \frac{\mathbf{W}_0 \mathbf{X} + \mathbf{b}}{\|\mathbf{W}_0 \mathbf{X} + \mathbf{b}\|_2} + \frac{\beta}{\gamma} \right) \quad (25)$$

Since  $\mathbf{O}$  preserves the  $L_2$ -norm ( $\|\mathbf{Z}\|_2 = \|\mathbf{O} \mathbf{Z}\|_2$  for any  $\mathbf{Z}$ ), this is equivalent to

$$\mathbf{Y} = \mathbf{W}_1 \gamma \mathbf{O}^\top \left( \frac{\mathbf{O} (\mathbf{W}_0 \mathbf{X} + \mathbf{b})}{\|\mathbf{O} (\mathbf{W}_0 \mathbf{X} + \mathbf{b})\|_2} + \frac{\mathbf{O} \beta}{\gamma} \right) \quad (26)$$

which can be expanded into:

$$\mathbf{Y} = \mathbf{W}_1 \gamma \mathbf{O}^\top \left( \frac{\mathbf{O} \mathbf{W}_0 \mathbf{X} + \mathbf{O} \mathbf{b}}{\|\mathbf{O} \mathbf{W}_0 \mathbf{X} + \mathbf{O} \mathbf{b}\|_2} + \frac{\mathbf{O} \beta}{\gamma} \right) \quad (27)$$

We can therefore rewrite the network as

$$\mathbf{Y} = \mathbf{W}'_1 \text{RMSNorm}(\mathbf{W}'_0 \mathbf{X} + \mathbf{b}'; \beta', \gamma') \quad (28)$$

with transformed parameters

$$\mathbf{W}'_1 = \mathbf{W}_1 \gamma \mathbf{O}^\top, \quad \mathbf{W}'_0 = \mathbf{O} \mathbf{W}_0, \quad \mathbf{b}' = \mathbf{O} \mathbf{b}, \quad (29)$$

and normalization constants

$$\gamma' = \mathbf{1}_M, \quad \beta' = \frac{\mathbf{O} \beta}{\gamma}. \quad (30)$$

This derivation for RMSNorm underscores a critical point: the nature of a component's inherent symmetries dictates the set of valid transformation matrices  $\mathbf{S}_l$  that can be used for reparameterization while preserving its functionality. Let  $\mathcal{S}_l$  denote the set of allowed transformation matrices for a given layer  $l$  of dimension  $d_l$ . If a layer exclusively admits permutation symmetry, then  $\mathcal{S}_l = \mathcal{P}_{d_l}$ , the finite set of  $d_l \times d_l$  permutation matrices. However, if a component, such as RMSNorm blocks in Transformers, exhibit orthogonal symmetry, the set of permissible transformations expands to  $\mathcal{S}_l = \mathcal{O}(d_l)$ , the orthogonal group. For components allowing even more general transformations, this could be  $\mathcal{S}_l = \mathcal{GL}(d_l, \mathbb{R})$ , the general linear group of invertible matrices. These sets of transformations are nested, with  $\mathcal{P}_{d_l} \subset \mathcal{O}(d_l) \subset \mathcal{GL}(d_l, \mathbb{R})$ , where  $\mathcal{P}_{d_l}$  is a finite group, while  $\mathcal{O}(d_l)$  and  $\mathcal{GL}(d_l, \mathbb{R})$  are continuous, significantly larger Lie groups. The overall alignment transformation  $\pi$  for the



entire network is constructed by selecting an appropriate  $S_l \in \mathcal{S}_l$  for each layer or component. Consequently, by identifying and utilizing the richest symmetry class available for each network component—for example, employing transformations from  $\mathcal{O}(d_l)$  for an RMSNorm layer rather than being restricted to the smaller set  $\mathcal{P}_{d_l}$  (which might be the case if its orthogonal symmetry were immediately broken by a subsequent element-wise activation function)—we drastically expand the search space of valid, function-preserving reparameterizations  $\pi(\theta)$ . This significantly larger search space offers more degrees of freedom in the alignment process, thereby substantially increasing the potential to discover configurations  $\theta'$  (aligned parameter vector) that lie on low-loss linear paths and thus achieve Linear Mode Connectivity between independently trained models.

## E Limitations

While the research introduces a unified framework for symmetry-aware model alignment in Transformers, it presents some limitations and areas for future exploration. The empirical results for language models were based on a smaller version of GPT-2 language models with reduced parameters due to resource constraints (Section 5.2), indicating the need for evaluation on larger, more contemporary language models. The current scope is focused on aligning pairs of models that have been pretrained on the same task, and further investigation could extend these methodologies to models trained on (partially) different tasks [Stoica et al., 2024] as well as explore scaling the alignment to enable the merging of multiple (more than two) Transformer models, similarly to what is shown in Singh and Jaggi [2020], Ainsworth et al. [2022], Crisostomi et al. [2024] for convolutional and residual networks. Additionally, while additional results (Appendix C) demonstrate that soft permutations can improve test-time performance of the merged model, more work is needed to fully refine soft relaxations of symmetry operations to improve the performance of aligned models. Finally, the study utilizes standard benchmarks such as CIFAR-10/100 for Vision Transformers and BookCorpus for GPT-2; exploring performance on a broader or more complex range of benchmarks could further validate the findings.