Symmetry-Aware Graph Metanetwork Autoencoders: Model Merging through Parameter Canonicalization

Odysseas Boufalis^{1*} **Jorge Carrasco-Pollo**^{1*} **Joshua Rosenthal**^{1*}
Eduardo Terrés-Caballero^{1*} **Alejandro García-Castellanos**^{2}

^{1}Informatics Institute, University of Amsterdam
^{2}Amsterdam Machine Learning Lab (AMLab), University of Amsterdam

Abstract

Neural network parameterizations exhibit inherent symmetries that yield multiple equivalent minima within the loss landscape. Scale Graph Metanetworks (ScaleGMNs) explicitly leverage these symmetries by proposing an architecture equivariant to both permutation and parameter scaling transformations. Previous work by Ainsworth et al. (2023) addressed permutation symmetries through a computationally intensive combinatorial assignment problem, demonstrating that leveraging permutation symmetries alone can map networks into a shared loss basin. In this work, we extend their approach by also incorporating scaling symmetries, presenting an autoencoder framework utilizing ScaleGMNs as invariant encoders. Experimental results demonstrate that our method aligns Implicit Neural Representations (INRs) and Convolutional Neural Networks (CNNs) under both permutation and scaling symmetries without explicitly solving the assignment problem. This approach ensures that similar networks naturally converge within the same basin, facilitating model merging, i.e., smooth linear interpolation while avoiding regions of high loss. The code is publicly available on our GitHub repository[†].

1 Introduction

Model merging has emerged as a powerful technique for combining the capabilities of independently trained neural networks, originally introduced to address Linear Mode Connectivity (LMC) barriers [4]. LMC barriers refer to the obstacles encountered when attempting to linearly interpolate between independently trained networks, which often results in suboptimal performance [26]. However, a fundamental challenge arises from the fact that neural networks exhibit extensive *parameter-space symmetries*: systematic permutations and scalings of parameters that leave the network's function unchanged [35]. These symmetries create multiple equivalent representations of the same function scattered across different regions of parameter space, making direct interpolation between network weights ineffective and often resulting in high loss barriers that hinder linear mode connectivity.

The standard solution is to *explicitly align* networks before merging by finding correspondences between their parameters. One approach uses numerical solvers: Git Re-Basin [1] formulates alignment as a layer-wise linear assignment problem solved via the Hungarian algorithm [13], but this scales poorly with network size due to combinatorial complexity and typically operates only on pairs of networks. An alternative approach employs *metanetworks*, i.e., neural networks that take other networks as input, to directly learn alignment mappings [5, 15, 11, 23]. These methods attempt to approximate optimal permutations of weights through learned transformations, bypassing combinatorial solvers.

[*]Authors ordered alphabetically. Equal contribution. Correspondence: `<a.garciacastellanos@uva.nl>`
[†]`https://github.com/odyboufalaki/Symmetry-Aware-Graph-Metanetwork-Autoencoders`

However, current approaches face significant limitations when handling the full spectrum of parameter symmetries. While some existing methods do address scaling symmetries [32, 25], they either operate only on specific architectures or employ heuristic approaches that sequentially handle different transformation types—first solving for one type of weight transformation (typically permutations) and then addressing scalings as a separate step. As we demonstrate in the Appendix (Section A), these sequential heuristics can be effective for simple cases like sign flip symmetries but may prove adversarial when dealing with general scaling symmetries, potentially degrading rather than improving alignment quality.

This limitation is particularly significant given recent evidence that scaling symmetries play a crucial role in understanding neural networks' weight space [35, 33, 16, 34, 3]. Indeed, it has been shown that acknowledging scaling symmetries substantially improves training dynamics during optimization [35]. However, there has been no systematic analysis of how general scaling symmetries interact with model merging quality, nor methods that jointly handle both permutation and scaling symmetries in a principled, architecture-agnostic manner—gaps that motivate our work.

These limitations motivate a fundamentally different approach: rather than explicitly computing alignments, we can *implicitly canonicalize* networks by learning to map them into a symmetry-invariant representation space. Such an approach requires an encoder that is equivariant to the relevant parameter symmetries. Scale Graph Metanetworks (ScaleGMNs) [9] provide exactly this property through their graph-based neural network representation that is provably equivariant to both permutations and scalings.

Building on this foundation, we propose a canonicalization autoencoder framework where a ScaleGMN encoder maps neural networks into symmetry-invariant latent codes, and an MLP decoder reconstructs canonical parameter representations from these codes. This approach automatically handles both permutation and scaling symmetries without combinatorial optimization, scales efficiently with network size, and generalizes across multiple networks simultaneously. The resulting canonical representations enable effective linear interpolation and provide new insights into the geometry of neural network parameter spaces.

In summary, our main contributions are:

- A novel symmetry-aware autoencoding framework that achieves implicit neural network canonicalization by jointly collapsing permutation and scaling symmetry orbits, eliminating the need for explicit alignment procedures.
- Comprehensive analysis on the impact of scaling symmetries on model interpolation quality.
- An extended Git Re-Basin algorithm that incorporates sign-flip symmetries, establishing a stronger baseline for symmetry-aware neural network alignment.
- Extensive experimental validation across implicit neural representations and CNNs, demonstrating significant improvements in linear mode connectivity over existing explicit alignment methods.

2 Related Work

Weight-space alignment. Model merging seeks to combine multiple trained models into one. In an ideal case, if models reside within the same loss basin, they possess linear mode connectivity and can be merged through a simple linear interpolation of their weights while maintaining low loss [4, 19]. However, this ideal condition is rarely met in practice, mainly because of the inherent symmetries of neural networks, such as permutation symmetry, which states that reordering the neurons within a hidden layer (and its connected weights) does not alter the overall function of the network [7].

Because of this symmetry, models trained with different random initializations tend to converge to functionally equivalent but geometrically distinct solutions, preventing a naive interpolation from succeeding [1, 28]. The key insight of modern merging techniques is to exploit this very property. Ainsworth et al. [1] do this by finding an optimal permutation P_i to align the neurons of one model with another by solving the *assignment problem*, thus bringing the models into a shared basin. Once aligned, the simple linear interpolation can be successfully applied to the weights of a given layer i:

$$W_i^* = \gamma W_i^A + (1-\gamma) P_i W_i^B P_{i-1}^\top$$

with $\gamma \in [0, 1]$, where P_i permutes the output dimension of layer i and P_{i-1} permutes its input dimension.

However, solving the assignment problem is NP-hard, and therefore usually requires using heuristic iterative optimization algorithms such as Git Re-Basin [1]. On the other hand, an alternative paradigm uses a dedicated neural network for amortizing this task [5, 15, 11]. Among these, DEEP-ALIGN [23] trains a permutation-equivariant architecture to predict the layer-wise permutation matrices in a single forward pass. By learning the alignment task itself, DEEP-ALIGN directly outputs the optimal transformation, bypassing the need for separate, iterative solvers.

As an alternative to explicit alignment, Lim et al. [18] show that reducing parameter symmetries in the network architecture itself can render permutation-based alignment for model merging unnecessary.

Similarly, our method yields canonical network representations without explicitly solving combinatorial alignment problems, relying instead on symmetry-aware encoder backbones combined with an appropriate functional loss. Beyond permutations, neural networks also exhibit scaling symmetries related to activation functions [6]. Our work seeks to account for both symmetries.

Weight-space networks and latent representations. Metanetworks [36, 38, 22, 29, 37] treat each data point as a distinct neural network, with symmetries extensively studied in these architectures. Graph Metanetworks (GMNs) [17, 12] represent neural networks as graphs, using permutation-equivariant graph neural networks (GNNs) for processing. ScaleGMNs [9] extend GMNs by incorporating scaling symmetries for MLPs and CNNs, demonstrating enhanced performance on tasks such as INR classification, network editing, and generalization prediction.

Another application of metanetworks is embedding model weights into meaningful latent representations. Luigi et al. [20] pioneered this approach using a simple MLP to map vectorized weights of Neural Fields to latent vectors. The authors showed that these latent vectors encode both the semantics and properties of the original network, including performance characteristics. Building on this work, Zhou et al. [37] presented INR2ARRAY, an autoencoder that maps model weights into a latent point cloud and reconstructs multiple weights representing local patches in the input domain of the original Neural Field.

Our method also constructs an autoencoder using an invariant encoder, but differs significantly from INR2ARRAY in two key aspects:

- **Encoder architecture:** While Zhou et al. [37] employ Neural Functional Transformers (NFTs) as their invariant encoder, we propose using Graph Metanetworks such as ScaleGMNs. Although NFTs offer high scalability, they require weight vectorization during the attention mechanism and do not actively leverage the computational graph structure as ScaleGMNs do. Additionally, NFTs are only equivariant with respect to permutations, whereas our approach acknowledges the full symmetry group of the network.
- **Latent representation and decoding:** The latent point cloud representation and decoding of multiple subnetwork weights can provide an expressive latent code for INRs [2]. However, this configuration limits their applications. We propose using a single invariant latent vector and decoding a single canonical weight configuration. This approach enables us to encode networks beyond INRs, such as CNN classifiers, and utilize the canonicalized weights for model merging applications.

3 Preliminaries

3.1 Implicit Neural Representations

Implicit Neural Representations (INRs), also known as Neural Fields, encode signals such as images, audio, or 3D shapes directly within neural network parameters, mapping continuous input coordinates to corresponding output values learned from discrete samples [20]. While standard MLP-based INRs are resolution-independent, they often struggle to capture high-frequency details due to spectral bias, leading to overly smooth reconstructions of signals with fine structures. The sinusoidal representation network (SIREN) architecture [27] addresses this limitation by replacing conventional activations with periodic sine functions, enabling faithful representation of both low and high frequency components and improving convergence in tasks like neural rendering and physical simulation. Since INRs store the entirety of the signal information in their weights, architectures processing these representations must also account for inherent parameter-space symmetries that can give rise to redundant minima [7].

3.2 Permutation and scaling symmetries

In this section, we review recent work that investigates the inherent symmetries in neural networks [6]. The key insight is that certain activation functions create hidden symmetries in the weight space, under which networks with different parameter values can exhibit functionally equivalent behavior.

Specifically, they prove that for many activation functions σ, the *intertwiner group* $I_{\sigma,d}$ (matrices A such that $\sigma(Ax) = B\sigma(x)$) consists of matrices of the form PQ. Here, P is a permutation matrix, and Q is a diagonal matrix $\text{diag}(q_1, \dots, q_d)$ where q_i belong to D_σ, a 1-dimensional group specific to σ.

For ReLU, Proposition 3.4 in [29] states that the intertwiner matrices $A = PQ$ are **generalised permutation matrices with positive entries**, i.e., $Q = \text{diag}(q_1, \dots, q_d)$ with $q_i > 0$. For SIRENs, which use the sine activation $\sigma(x) = \sin(\omega x)$, as well as for the $\tanh$ activation function, the same proposition specifies that A is a **signed permutation matrix**, meaning the entries q_i are restricted to ± 1. Moreover, for all the aforementioned activations, the transformation B coincides with A.

Applied layer-wise in a network, these symmetries take the form

$$W'_\ell = (Q_\ell P_\ell) W_\ell (P_{\ell-1}^\top Q_{\ell-1}^{-1}), \quad b'_\ell = (Q_\ell P_\ell) b_\ell \quad \Longrightarrow \quad u_{\theta'}(x) = u_\theta(x).$$

The $(Q_\ell P_\ell)$ factor reorders the neurons in layer ℓ and rescales them—by positive factors for ReLU, or by ± 1 for sine/$\tanh$. To preserve the network's output, W'_ℓ is transformed by $(Q_\ell P_\ell)$ on its output side, while $(P_{\ell-1}^\top Q_{\ell-1}^{-1})$ adjusts the input side to remain compatible with the preceding layer's transformation.

To address these symmetries, [9] introduce ScaleGMN, a specialized type of GMN that represents neural networks as computational graphs, with neurons as nodes and connections as edges (see Appendix B for details).

4 Neural Network Autoencoding for Canonicalization

We propose an autoencoder framework for canonicalizing neural networks in parameter space with respect to a predefined set of symmetries. The encoder maps input parameters into a symmetry-invariant latent vector, while a simple MLP decoder reconstructs a canonical representative of the network. The encoder's architectural invariance ensures a unique latent representation for symmetry-equivalent networks. This canonicalization technique aligns with recent work in Geometric Deep Learning such as [8], which learns a representative for each orbit by constructing a G-invariant function $f_\theta : X \to X$ such that $f_\theta(x) \in Gx$ for each $x \in X$. This approach contrasts with moving frame techniques [24], which instead learn the group action that maps elements to their orbit representative. As discussed in Section 2, the DEEP-ALIGN model [23] belongs to this latter category.

Our experiments employ two symmetry-aware encoders: Neural Graphs [12], which is permutation-invariant, and ScaleGMN [9], which is both permutation and scale-invariant. This choice lets us disentangle the effect of permutation invariance and measure the added value of scale invariance. The architectural differences between these encoders also highlight the robustness of our framework across diverse designs. Figure 1 illustrates the pipeline using ScaleGMN. More scalable permutation-invariant metanetworks such as UNFs [39] could extend this approach to larger architectures, including Transformers. Since ScaleGMN does not yet support such architectures, we focused on testing the autoencoder's generalizability across symmetries rather than model size. Nonetheless, scaling to larger networks is straightforward given an appropriate encoder.

We apply this pipeline to INRs and CNNs. The objective is for the decoded network to be functionally equivalent to the input while mapping outputs into a single basin. To enforce this, we use a functional training loss: for INRs, we regress the pixel activations of the image represented by the INR; for CNN classifiers, we minimize the KL divergence between probability distributions obtained from each network's logits via a temperature-scaled softmax. Notably, this loss formulation could be directly applied to canonicalizing other architectures with probabilistic outputs, such as Transformers, where the autoregressive objective relies on a softmax over the vocabulary. For CNNs we consider two

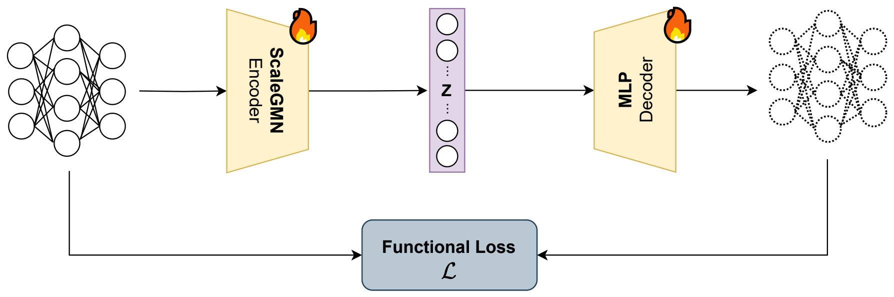


Figure 1: Autoencoder architecture for neural network canonicalization using a permutation and scaling-invariant ScaleGMN encoder, MLP decoder, and functional loss to preserve network equivalence.

activation variants—ReLU and *tanh*—to probe symmetry effects: ReLU induces positive-scaling symmetries across adjacent layers, whereas *tanh* only induces sign-flip symmetry. To assess the role of these symmetries, we also train an autoencoder whose encoder is a Neural Graphs network [12], which does not account for scaling.

5 Experiments and results

In this section, we empirically validate our autoencoder-based canonicalization framework. We evaluate its ability to establish high-performance linear paths between neural networks, a key indicator of successful model merging. All experiments are performed on a single H100 GPU.

5.1 Experimental setup

Tasks and Datasets. We evaluate our canonicalization approach across two complementary scenarios that test different aspects of network alignment:

(1) Aligning INRs: We test alignment between functionally *identical* but parametrically distinct Implicit Neural Representations (INRs). These networks are MLPs with sine activations trained to represent MNIST digits [14] introduced in [22]. To create functionally equivalent endpoints, we start from a trained INR, apply intertwiner group transformations (permutations and/or scaling), and add small perturbations to prevent identical latent mappings. We systematically vary the applied symmetries (using permutations only, scaling only, or both) to isolate the contribution of different symmetry types. The goal is to recover perfect linear mode connectivity with zero interpolation barrier. To enable a fair comparison, we generalize the Git Re-Basin algorithm to also handle scaling symmetries; the details of this generalization are provided in the Appendix (Section A). Also, as an additional ablation, we report interpolation results without resorting to adding noise in the Appendix (Section D.1).

(2) Aligning CNN Classifiers: We evaluate the ability to connect pairs of CNNs that are not necessarily functionally equivalent. We interpolate between pairs of CNNs with distinct CIFAR-10 test accuracies, selected from the SmallCNN Zoo dataset [31]. We report interpolation results for randomly sampled CNN pairs as well as averaged results over 20 pairs, which are drawn from the top 1,500 highest-performing models in the SmallCNN Zoo. The models are simple CNNs with a [16, 16, 16] hidden channel architecture, 3x3 convolutions, and an average pooling layer. The goal is to find a high-performance path that avoids the typical accuracy drop.

Compared Methods. We benchmark our proposed method against several baselines:

- **Naive Interpolation:** A standard baseline involving linear interpolation in the original parameter space.
- **Linear assignment** (w/ Git Re-Basin [1]): An optimization-based method that explicitly solves the layer-wise assignment problem to align permutation symmetries. As previously

mentioned, for the INR experiments, we use our generalized version (Appendix, Section A) that also handles sign-flip symmetries.

- **Neural Graphs Autoencoder (Ours):** Our autoencoder framework equipped with a permutation-only equivariant encoder [12] to ablate the effect of handling scaling symmetries.
- **ScaleGMN Autoencoder (Ours):** Our proposed autoencoder using the ScaleGMN encoder, which is equivariant to both permutation and scaling symmetries.

Evaluation Metrics. For both tasks, we evaluate the quality of the linear interpolation path between two networks, θ_A and θ_B, after alignment. For INRs (Figure 3), we measure the reconstruction loss at intervals along the path, where a lower barrier indicates better alignment. For CNNs (Figures 4 and 5), we measure both the CIFAR-10 test accuracy and the cross-entropy loss along the path, where successful linear mode connectivity is indicated by a monotonic, high-accuracy curve and a correspondingly low loss barrier.

5.2 Results

Autoencoder Performance. We report on the reconstruction capacity of our autoencoder framework using distinct metrics for each task. For the INR setting (Table 1), reconstruction quality is quantified by the mean squared error (MSE) between the pixel activations of the original and decoded networks. For the CNN setting (Table 2), we assess functional preservation via two metrics: the L1 error, representing the absolute difference in CIFAR-10 test accuracy, and Kendall's Tau [10], a rank correlation coefficient that quantifies how well the performance ordering of networks is maintained after reconstruction.

Table 1: Reconstruction error of autoencoder variants for the Implicit Neural Representation (INR) task.

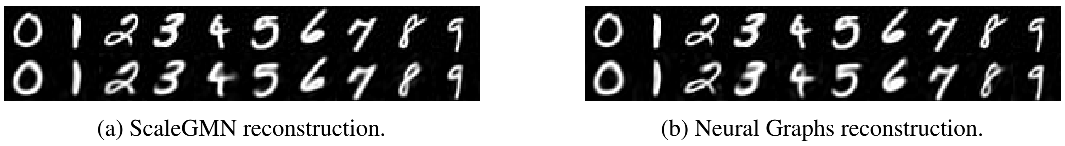
Model	**Reconstruction Error (↓)**	
	Train	**Test**
ScaleGMN	0.0096	**0.0106**
Neural Graphs	**0.0068**	0.0135

Table 2: L1-Error and Kendall's Tau of autoencoder variants on the test set for the CNN canonicalization.

Model	**L1-Error (↓)**	**Kendall's Tau (↑)**
ReLU Variants		
ScaleGMN-ReLU	**0.0111**	**0.9100**
Neural Graphs-ReLU	0.0142	0.8914
Tanh Variants		
ScaleGMN-tanh	**0.0090**	**0.9160**
Neural Graphs-tanh	0.0113	0.8905

Model selection mirrors the evaluation metrics: for INRs we choose by validation MSE; for CNNs by validation L1 error, aiming to preserve the original accuracy as closely as possible. Hyperparameter tuning and other experimental details for both autoencoders are provided in Appendix C.

In Figure 2, we visualize the output of our autoencoder across the two experimental autoencoder setups. We see that in each case, the autoencoder is able to successfully recover the input INR, producing mappings with high visual fidelity.



(a) ScaleGMN reconstruction.

(b) Neural Graphs reconstruction.

Figure 2: Comparison between the ground truth (top row) and reconstruction obtained with the different types of INR autoencoders (bottom row), for a set of distinct MNIST digits.

INR Interpolation. As shown in Figure 3, naive interpolation fails with a significant loss barrier, confirming the endpoints lie in separate basins. Both Git Re-Basin and our autoencoder approach succeed in establishing near-linear mode connectivity across all three transformation types (permutation, scaling, and their combination), with the autoencoder exhibiting more stable performance, as indicated by the smaller standard deviation (shaded region).

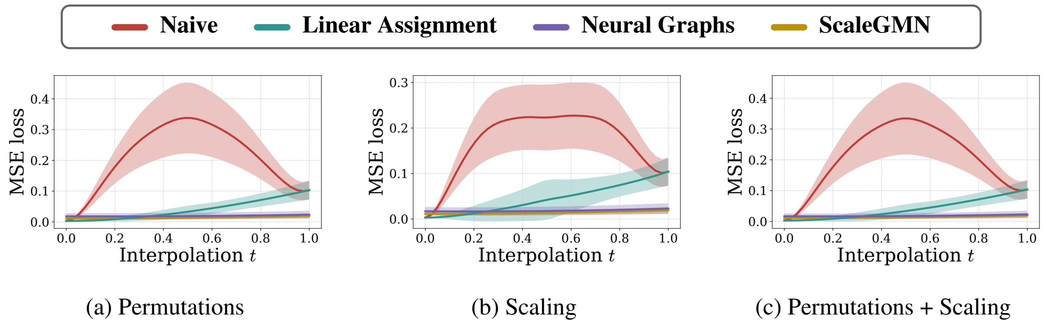


Figure 3: Interpolation experiments comparing different neural network alignment methods across various perturbation scenarios.

Alignment via Git Re-Basin shows a higher loss at the endpoints due to the perturbations introduced by the added noise. In contrast, both autoencoder variants are robust to this noise, with the ScaleGMN encoder performing slightly better, maintaining a lower loss profile. The ScaleGMN-based autoencoder performs marginally better than the Neural Graphs version, demonstrating the slight benefit of its scaling symmetry-aware architecture in this controlled setting. Similar qualitative behavior is observed across the three orbit-construction settings (permutation only, scaling only, and both), supporting the applicability of both our method and the Git Re-Basin algorithm.

Furthermore, when we interpolate directly in the ScaleGMN latent space for identical INR pairs, we observe identical performance to weight-space interpolation, confirming that our autoencoder learns robust representations (detailed results in Appendix Section D.2).

CNN Interpolation. The results of interpolating functionally distinct CNNs are presented in Figures 4 and 5, which show average interpolation curves computed over 20 pairs of distinct CNN models. Additional results for individual CNN pairs are provided in Appendix (Section D.3). While Git Re-Basin substantially mitigates the catastrophic failure observed with naive interpolation, it still faces a significant performance barrier, with accuracy dropping sharply at the midpoint. This aligns with the findings of Ainsworth et al. [1], which indicate that permutation matching alone is often insufficient for connecting networks of limited width, such as those in the SmallCNN Zoo dataset. Furthermore, Navon et al. [23] report that the weight-matching strategy employed by Git Re-Basin is not optimal on CNN datasets.

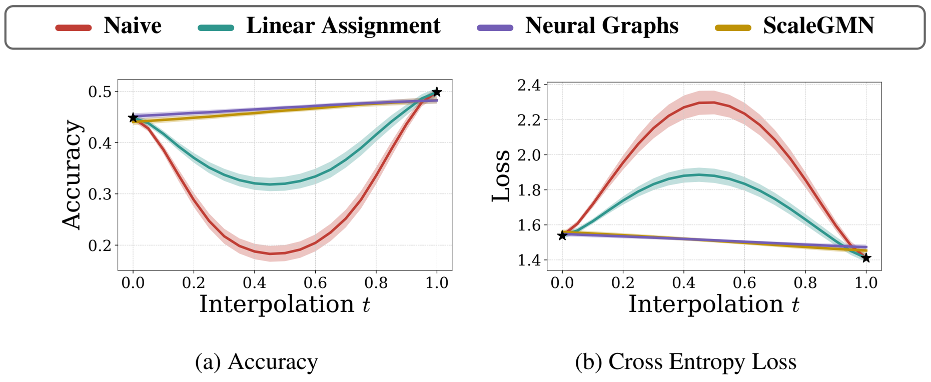
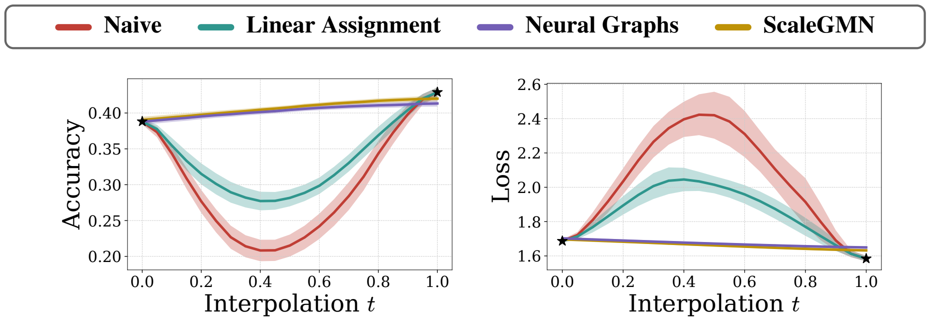


Figure 4: Average interpolation curves (and standard deviation) over 20 pairs of distinct CNN models with ReLU activation.

In contrast, our autoencoder's reconstructed space enables a nearly monotonic, straight-line interpolation path between lower- and higher-performing models, revealing a comprehensive geometric representation that captures permutations and scaling symmetries. Furthermore, the comparable performance of the permutation-invariant Neural Graphs variant and the scale-aware ScaleGMN variant suggests that scaling symmetries hold less importance for model merging within the context of the datasets used.



(a) Accuracy (b) Cross Entropy Loss

Figure 5: Average interpolation curves (and standard deviation) over 20 pairs of distinct CNN models with Tanh activation.

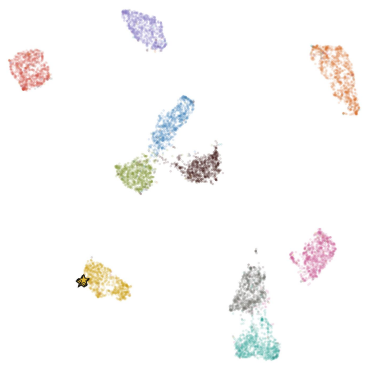
6 Discussion

6.1 Importance of scaling symmetries

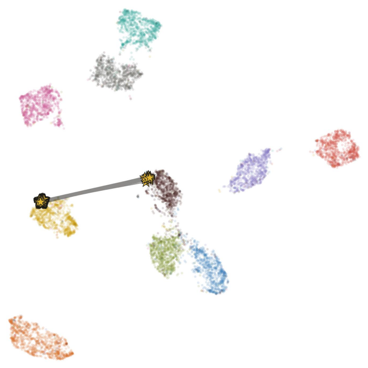
We begin by visualizing the latent representations learned by both the ScaleGMN and Neural Graphs encoders for the INR experiments (see Figure 6). Using UMAP [21] to reduce the dimensionality of latent representations across the entire INR dataset, we observe that data points naturally cluster by class (i.e., the digit they represent), despite the autoencoder being trained solely for reconstruction with functional equivalence rather than classification.

While the interpolation experiments presented in Section 5.2 demonstrate that model merging can be achieved in expectation without explicitly leveraging scaling symmetries, we further investigate the importance of addressing these symmetries within our autoencoder framework.

To assess the role of scaling symmetries, we plot the latent representation of a reference data point alongside a subset of its orbit generated with respect to the scaling group. As shown in Figure 6, the ScaleGMN encoder successfully collapses all orbit members to a single point in the latent space, whereas the Neural Graphs encoder fails to achieve this collapse. This difference has important implications: since the ScaleGMN latent space aligns all scale-equivalent networks to the same representation, their decoded versions naturally correspond to the same canonical representative in parameter space. In contrast, the Neural Graphs encoder's inability to collapse scale-equivalent networks in the latent space prevents it from producing canonicalized representations, making it impossible to recover scale-equivalent canonical networks from its outputs.



(a) Scale-GMN



(b) Neural Graphs

Figure 6: Latent representations learned by both encoder variants. The stars represent weight-transformed, functionally equivalent versions of the same INR.

Therefore, while scale invariance may be less critical for model merging tasks specifically, it becomes essential for other latent space applications. Following approaches in [20, 37], one could utilize these latent codes as network proxies for downstream tasks such as performance prediction, network classification, and INR-based generative modeling. For all such applications, having a latent representation where each point represents an entire family of functionally equivalent networks—as achieved by our ScaleGMN encoder—would be highly beneficial.

6.2 Algorithmic complexity

The inference time complexity of the autoencoder scales linearly with the total number of parameters, P, due to its main components: the ScaleGMN encoder and MLP decoder. All model operations—initialization, message computation, aggregation, node and edge updates, and readout—scale with $\mathcal{O}(P)$. The encoder is dominated by message passing, with complexity $\mathcal{O}(L_{\mathrm{gnn}}P)$, where L_{gnn} is the number of GNN layers. Both graph initialization and readout, as well as the MLP decoder, scale linearly with P. Linear assignment using the Hungarian algorithm has complexity $\mathcal{O}(n^3)$ per iteration for an $n \times n$ cost matrix. In the worst case, $n \approx \sqrt{P}$, giving a cost of $\mathcal{O}(P^{3/2})$ per iteration. With T iterations, this becomes $\mathcal{O}(T \cdot P^{3/2})$, though in architectures with deep networks and fixed-size weight matrices (e.g., CNNs with small kernels), the cost is reduced to $\mathcal{O}(T \cdot P)$. Comparing both methods, Git Re-Basin is an iterative method that requires multiple assignments until convergence and constitutes an approximation [26] to the original assignment problem (see more in the Appendix, Section A), whereas ScaleGMN and Neural Graph autoencoders require model training, which implies sufficient training iterations and data. In turn, they present a complexity of $\mathcal{O}(P)$ during inference, while Git Re-Basin presents $\mathcal{O}(T \cdot P)$ and $\mathcal{O}(T \cdot P^{3/2})$ in its best and worst cases, respectively.

7 Conclusion

We address the challenging problem of model merging, which is often hindered by parameter-space symmetries. Our approach introduces a symmetry-aware autoencoder framework that leverages ScaleGMNs to respect the natural permutation and scaling symmetries inherent in neural networks. Rather than explicitly solving a combinatorial assignment problem, the method learns to map functionally equivalent networks to a single canonical representation in weight space.

At inference time, our framework canonicalizes unseen networks in a single forward pass, eliminating the need for iterative optimization. Experiments demonstrate superior linear mode connectivity compared to both naive interpolation and explicit alignment methods. Using a Neural Graphs encoder as a comparative backbone, we further highlight the importance of incorporating scaling symmetries alongside permutation symmetries on INRs and small CNNs. In addition, the learned latent space exhibits well-structured properties, suggesting potential for downstream applications beyond model merging.

This study primarily evaluates the framework on smaller-scale MLPs and CNNs, providing a controlled setting in which to establish its effectiveness. A limitation of the approach is the requirement to train the autoencoder on a representative dataset of networks—an upfront cost that enables efficient canonicalization at inference time. Together, these aspects form a strong foundation for demonstrating feasibility, while leaving open opportunities to extend the framework to larger and more complex architectures, such as Transformers and Large Language Models [30]. We view this as an exciting direction for future research, with the potential to enable practical, symmetry-aware canonicalization at scale.

Acknowledgements

We wish to thank Erik J. Bekkers and Efstratios Gavves, as well as the University of Amsterdam, for providing access to the computational resources for this project. Alejandro García Castellanos is funded by the Hybrid Intelligence Center, a 10-year programme funded through the research programme Gravitation which is (partly) financed by the Dutch Research Council (NWO).

References

[1] Samuel K. Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries, 2023. URL `https://arxiv.org/abs/2209.04836`.

[2] Matthias Bauer, Emilien Dupont, Andy Brock, Dan Rosenbaum, Jonathan Richard Schwarz, and Hyunjik Kim. Spatial functa: Scaling functa to imagenet classification and generation. *arXiv preprint arXiv:2302.03130*, 2023.

[3] Alejandro García-Castellanos, Giovanni Luca Marchetti, Danica Kragic, and Martina Scolamiero. Relative representations: Topological and geometric perspectives. *arXiv preprint arXiv:2409.10967*, 2024.

[4] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 8803–8812, Red Hook, NY, USA, 2018. Curran Associates Inc.

[5] Daniel Gibbons, Cheng-Chew Lim, and Peng Shi. Deep learning for bipartite assignment problems. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 2318–2325, 2019. doi: 10.1109/SMC.2019.8914228.

[6] Charles Godfrey, Davis Brown, Tegan Emerson, and Henry Kvinge. On the symmetries of deep learning models and their internal representations, 2023. URL `https://arxiv.org/abs/2205.14258`.

[7] Robert Hecht-Nielsen. On the algebraic structure of feedforward network weight spaces. In *Advanced Neural Computers*, pages 129–135. Elsevier, 1990.

[8] Sékou-Oumar Kaba, Arnab Kumar Mondal, Yan Zhang, Yoshua Bengio, and Siamak Ravanbakhsh. Equivariance with learned canonicalization functions. In *International Conference on Machine Learning*, pages 15546–15566. PMLR, 2023.

[9] Ioannis Kalogeropoulos, Giorgos Bouritsas, and Yannis Panagakis. Scale equivariant graph metanetworks, 2024. URL `https://arxiv.org/abs/2406.10685`.

[10] Maurice Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–89, 1938.

[11] Minseok Kim, Hoon Lee, Hongju Lee, and Inkyu Lee. Deep learning based resource assignment for wireless networks. *IEEE Communications Letters*, 25(12):3888–3892, 2021. doi: 10.1109/LCOMM.2021.3116233.

[12] Miltiadis Kofinas, Boris Knyazev, Yan Zhang, Yunlu Chen, Gertjan J. Burghouts, Efstratios Gavves, Cees G. M. Snoek, and David W. Zhang. Graph neural networks for learning equivariant representations of neural networks. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=oO6FsMyDBt`.

[13] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. doi: https://doi.org/10.1002/nav.3800020109. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109`.

[14] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL `http://yann.lecun.com/exdb/mnist/`.

[15] Mengyuan Lee, Yuanhao Xiong, Guanding Yu, and Geoffrey Ye Li. Deep neural networks for linear sum assignment problems. *IEEE Wireless Communications Letters*, 7(6):962–965, 2018. doi: 10.1109/LWC.2018.2843359.

[16] Daniel Lengyel, Janith Petangoda, Isak Falk, Kate Highnam, Michalis Lazarou, Arinbjörn Kolbeinsson, Marc Peter Deisenroth, and Nicholas R Jennings. Genni: Visualising the geometry of equivalences for neural network identifiability. *arXiv preprint arXiv:2011.07407*, 2020.

[17] Derek Lim, Haggai Maron, Marc T. Law, Jonathan Lorraine, and James Lucas. Graph metanetworks for processing diverse neural architectures, 2023.

[18] Derek Lim, Theo Putterman, Robin Walters, Haggai Maron, and Stefanie Jegelka. The empirical impact of neural parameter symmetries, or lack thereof. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL `https://openreview.net/forum?id=pCVxYw6FKg`.

[19] Zhanran Lin, Puheng Li, and Lei Wu. Exploring neural network landscapes: Star-shaped and geodesic connectivity, 2024. URL `https://arxiv.org/abs/2404.06391`.

[20] Luca De Luigi, Adriano Cardace, Riccardo Spezialetti, Pierluigi Zama Ramirez, Samuele Salti, and Luigi Di Stefano. Deep learning on implicit neural representations of shapes, 2023. URL `https://arxiv.org/abs/2302.05438`.

[21] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2020. URL `https://arxiv.org/abs/1802.03426`.

[22] Aviv Navon, Aviv Shamsian, Idan Achituve, Ethan Fetaya, Gal Chechik, and Haggai Maron. Equivariant architectures for learning in deep weight spaces, 2023.

[23] Aviv Navon, Aviv Shamsian, Ethan Fetaya, Gal Chechik, Nadav Dym, and Haggai Maron. Equivariant deep weight space alignment. *arXiv preprint arXiv:2310.13397*, 2023.

[24] Peter J Olver. Joint invariant signatures. *Foundations of computational mathematics*, 1(1):3–68, 2001.

[25] Fabrizio Pittorino, Antonio Ferraro, Gabriele Perugini, Christoph Feinauer, Carlo Baldassi, and Riccardo Zecchina. Deep networks on toroids: removing symmetries reveals the structure of flat regions in the landscape geometry. In *International Conference on Machine Learning*, pages 17759–17781. PMLR, 2022.

[26] Wei Ruan, Tianze Yang, Yifan Zhou, Tianming Liu, and Jin Lu. From task-specific models to unified systems: A review of model merging approaches. *arXiv preprint arXiv:2503.08998*, 2025.

[27] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

[28] George Stoica, Daniel Bolya, Jakob Brandt Bjorner, Pratik Ramesh, Taylor Hearn, and Judy Hoffman. Zipit! merging models from different tasks without training. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=LEYUkvdUhq`.

[29] Hoang Tran, Thieu Vo, Tho Huu, Tan Nguyen, et al. Monomial matrix group equivariant neural functional networks. *Advances in Neural Information Processing Systems*, 37:48628–48665, 2024.

[30] Viet-Hoang Tran, Thieu N Vo, An Nguyen The, Tho Tran Huu, Minh-Khoi Nguyen-Nhat, Thanh Tran, Duy-Tung Pham, and Tan Minh Nguyen. Equivariant neural functional networks for transformers. *arXiv preprint arXiv:2410.04209*, 2024.

[31] Thomas Unterthiner, Daniel Keysers, Sylvain Gelly, Olivier Bousquet, and Ilya Tolstikhin. Predicting neural network accuracy from weights, 2021. URL `https://arxiv.org/abs/2002.11448`.

[32] Binchi Zhang, Zaiyi Zheng, Zhengzhang Chen, and Jundong Li. Beyond the permutation symmetry of transformers: The role of rotation for model fusion. In *Forty-second International Conference on Machine Learning*, 2025. URL `https://openreview.net/forum?id=wBJIO15pBV`.

[33] Bo Zhao, Iordan Ganev, Robin Walters, Rose Yu, and Nima Dehmamy. Symmetries, flat minima, and the conserved quantities of gradient flow. *arXiv preprint arXiv:2210.17216*, 2022.

[34] Bo Zhao, Nima Dehmamy, Robin Walters, and Rose Yu. Understanding mode connectivity via parameter space symmetry. In *UniReps: the First Workshop on Unifying Representations in Neural Models*, 2023. URL `https://openreview.net/forum?id=aP2a5i1iUf`.

[35] Bo Zhao, Robin Walters, and Rose Yu. Symmetry in neural network parameter spaces. *arXiv preprint arXiv:2506.13018*, 2025.

[36] Allan Zhou, Kaien Yang, Kaylee Burns, Adriano Cardace, Yiding Jiang, Samuel Sokota, J Zico Kolter, and Chelsea Finn. Permutation equivariant neural functionals. *Advances in neural information processing systems*, 36:24966–24992, 2023.

[37] Allan Zhou, Kaien Yang, Yiding Jiang, Kaylee Burns, Winnie Xu, Samuel Sokota, J. Zico Kolter, and Chelsea Finn. Neural functional transformers, 2023. URL `https://arxiv.org/abs/2305.13546`.

[38] Allan Zhou, Chelsea Finn, and James Harrison. Universal neural functionals. *Advances in neural information processing systems*, 37:104754–104775, 2024.

[39] Allan Zhou, Chelsea Finn, and James Harrison. Universal neural functionals, 2024. URL `https://arxiv.org/abs/2402.05232`.

A Generalized Git Re-Basin algorithm

As the authors in [1] state, the sum of a bilinear assignment problem is an NP-hard problem. Similarly, we approximate the linear assignment problem (LAP) using their same formulation, where we also account for weight matching and a transformation consisting of permutation and scale $\boldsymbol{T} = \boldsymbol{P}\boldsymbol{Q}$. Equation (1) constitutes the starting objective for approximating the LAP, where $\langle\cdot\rangle_F$ denotes the Frobenius inner product. Equation (4) then follows from the definition of the Frobenius inner product and properties of the trace.

$$\arg\max_{\boldsymbol{T}_\ell} \Big(\langle \boldsymbol{W}_\ell^{(\mathrm{A})}, \boldsymbol{T}_\ell \boldsymbol{W}_\ell^{(\mathrm{B})} \boldsymbol{T}_{\ell-1}^\top\rangle_F + \langle \boldsymbol{W}_{\ell+1}^{(\mathrm{A})}, \boldsymbol{T}_{\ell+1} \boldsymbol{W}_{\ell+1}^{(\mathrm{B})} \boldsymbol{T}_\ell^\top\rangle_F + \langle \boldsymbol{b}_\ell^{(\mathrm{A})}, \boldsymbol{T}_\ell \boldsymbol{b}_\ell^{(\mathrm{B})}\rangle_F\Big) \tag{1}$$

$$= \arg\max_{\boldsymbol{T}_\ell} \Big(\mathrm{Tr}\Big(\boldsymbol{T}_\ell \boldsymbol{W}_\ell^{(\mathrm{B})} \boldsymbol{T}_{\ell-1}^\top (\boldsymbol{W}_\ell^{(\mathrm{A})})^\top\Big) + \mathrm{Tr}\Big(\boldsymbol{T}_\ell (\boldsymbol{W}_{\ell+1}^{(\mathrm{B})})^\top \boldsymbol{T}_{\ell+1}^\top \boldsymbol{W}_{\ell+1}^{(\mathrm{A})}\Big) + \mathrm{Tr}\Big(\boldsymbol{T}_\ell \boldsymbol{b}_\ell^{(\mathrm{B})} (\boldsymbol{b}_\ell^{(\mathrm{A})})^\top\Big)\Big) \tag{2}$$

$$= \arg\max_{\boldsymbol{T}_\ell} \mathrm{Tr}\Big(\boldsymbol{T}_\ell \Big(\boldsymbol{W}_\ell^{(\mathrm{B})} \boldsymbol{T}_{\ell-1}^\top (\boldsymbol{W}_\ell^{(\mathrm{A})})^\top + (\boldsymbol{W}_{\ell+1}^{(\mathrm{B})})^\top \boldsymbol{T}_{\ell+1}^\top \boldsymbol{W}_{\ell+1}^{(\mathrm{A})} + \boldsymbol{b}_\ell^{(\mathrm{B})} (\boldsymbol{b}_\ell^{(\mathrm{A})})^\top\Big)\Big) \tag{3}$$

$$= \arg\max_{\boldsymbol{T}_\ell} \langle \boldsymbol{T}_\ell, \underbrace{\Big(\boldsymbol{W}_\ell^{(\mathrm{A})} \boldsymbol{T}_{\ell-1} (\boldsymbol{W}_\ell^{(\mathrm{B})})^\top + (\boldsymbol{W}_{\ell+1}^{(\mathrm{A})})^\top \boldsymbol{T}_{\ell+1} \boldsymbol{W}_{\ell+1}^{(\mathrm{B})} + \boldsymbol{b}_\ell^{(\mathrm{A})} (\boldsymbol{b}_\ell^{(\mathrm{B})})^\top\Big)}_{:= \boldsymbol{C}_\ell}\rangle_F \tag{4}$$

Defining $\boldsymbol{P}_\ell = (p_{ij}^{(\ell)})_{1\le i,j\le N}$ and $\boldsymbol{Q}_\ell = \mathrm{diag}(q_1^{(\ell)}, \ldots, q_N^{(\ell)})$ and rewriting the objective function by explicitly writing the transformation as $\boldsymbol{T} = \boldsymbol{P}\boldsymbol{Q}$ we obtain

$$\langle \boldsymbol{P}_\ell \boldsymbol{Q}_\ell, \boldsymbol{C}_\ell\rangle_F = \sum_{i=1}^N \sum_{k=1}^N (p_{ik}^{(\ell)} q_k^{(\ell)})(\boldsymbol{C}_\ell)_{ik} = \sum_{i=1}^N q_{\pi(i)}^{(\ell)} (\boldsymbol{C}_\ell)_{i,\pi(i)}. \tag{5}$$

We now demonstrate that when the network's activation functions are either sine or $\tanh$, this optimization problem admits an efficient solution within a single algorithm iteration. The key insight is to decompose the problem into two sequential steps: first solve an approximate Linear Assignment Problem (LAP) to obtain $\boldsymbol{P}_\ell$, then determine $\boldsymbol{Q}_\ell$ as an explicit function of both $\boldsymbol{C}_\ell$ and the previously computed $\boldsymbol{P}_\ell$.

Proposition 1. *Suppose the activation functions of the network are either* $\sin$ *or* $\tanh$*. Then, for a fixed cost matrix* $\boldsymbol{C}_\ell$*, the optimal transformation* $\boldsymbol{T}_\ell = \boldsymbol{P}_\ell^* \boldsymbol{Q}_\ell^*$ *that maximizes the objective in Equation* (5) *is given by:*

- $\boldsymbol{Q}_\ell^* = \mathrm{diag}({q^*}_1^{(\ell)}, \ldots, {q^*}_N^{(\ell)})$*, where* ${q^*}_i^{(\ell)} = \mathrm{sign}((\boldsymbol{C}_\ell)_{i,\pi^*(i)})$*, and*
- $\boldsymbol{P}_\ell^* = \arg\max_{\boldsymbol{P}_\ell} \sum_{i=1}^N |(\boldsymbol{C}_\ell)_{i,\pi(i)}|$.

Proof. For an arbitrary transformation $\boldsymbol{P}_\ell \boldsymbol{Q}_\ell$,

$$\langle \boldsymbol{P}_\ell^* \boldsymbol{Q}_\ell^*, \boldsymbol{C}_\ell\rangle_F = \sum_{i=1}^N \mathrm{sign}((\boldsymbol{C}_\ell)_{i,\pi^*(i)})(\boldsymbol{C}_\ell)_{i,\pi^*(i)} \tag{6}$$

$$= \sum_{i=1}^N |(\boldsymbol{C}_\ell)_{i,\pi^*(i)}| \tag{7}$$

$$\ge \sum_{i=1}^N |(\boldsymbol{C}_\ell)_{i,\pi(i)}| \qquad \text{(permutation assignment } \pi^*(\cdot) \text{ is optimal)} \tag{8}$$

$$\ge \sum_{ß=1}^N q_{\pi(i)}^{(\ell)} (\boldsymbol{C}_\ell)_{i,\pi(i)} \qquad (q_{\pi(i)}^{(\ell)} \in \{+1,-1\} \text{ for all } \ell, i) \tag{9}$$

$$= \langle \boldsymbol{P}_\ell \boldsymbol{Q}_\ell, \boldsymbol{C}_\ell\rangle_F\,. \tag{10}$$

□

Essentially, Proposition 1 shows that it suffices to perform weight matching considering only permutations on the absolute value of the cost matrix and later defining $\boldsymbol{Q}_\ell^*$ such that all terms in the sum of the inner product $\langle \boldsymbol{P}_\ell^* \boldsymbol{Q}_\ell^*, \boldsymbol{C}_\ell \rangle_F$ are positive, thus contribute to maximizing the objective. This allows for an efficient weight matching approximation as opposed to considering the cartesian product of the groups of permutations and scalings. It is important to note that $\boldsymbol{P}_\ell^* \boldsymbol{Q}_\ell^*$ are optimal for a fixed cost matrix $\boldsymbol{C}_\ell$, meaning that within a given iteration of the algorithm we can obtain the optimal transformation, but this may not be optimal with respect to the true objective. Since $\boldsymbol{C}_\ell$ is a function of $\boldsymbol{P}_\ell \boldsymbol{Q}_\ell$, the obtained solution is an approximation, which we empirically show to have considerable standard deviation but behaves adequately in expectation (monotonically increasing and slightly convex LAP interpolation, Figure 3).

An easy counterexample to see how this method does not converge to the global optimum is to consider two INRs, where one was generated by just permuting but not flipping signs, this is, $\boldsymbol{Q}_\ell = \boldsymbol{I}$. Now, $\boldsymbol{Q}_\ell^*$ will not necessarily be $\boldsymbol{I}$, since $\boldsymbol{C}_\ell$ can have negative entries, meaning the method will unnecessarily flip the signs of the INR and thus not match both into the same basin. In this sense, there is no preference for setting $\boldsymbol{Q}_\ell = \boldsymbol{I}$, nor is the case for $\boldsymbol{P}_\ell = \boldsymbol{I}$.

Performing this process of optimization by fixing the cost matrices, $\boldsymbol{C}_\ell$, intra-iteration and by updating the transformations of one layer at a time and at random, we arrive at Algorithm 1.

Algorithm 1 PERMUTATION+SCALINGCOORDINATEDESCENT

```
Require: Model weights $\boldsymbol{\Theta}_A = \{\boldsymbol{W}_1^{(A)}, \dots, \boldsymbol{W}_L^{(A)}\}$ and $\boldsymbol{\Theta}_B = \{\boldsymbol{W}_1^{(B)}, \dots, \boldsymbol{W}_L^{(B)}\}$.
Ensure: A set of layer-wise transformations $\{\boldsymbol{T}_1, \dots, \boldsymbol{T}_{L-1}\}$, where each $\boldsymbol{T}_\ell = \boldsymbol{Q}_\ell \boldsymbol{P}_\ell$ is a
    composition of a permutation $\boldsymbol{P}_\ell$ and a diagonal scaling $\boldsymbol{Q}_\ell$, chosen so as to approximately
    maximize $\mathrm{vec}(\boldsymbol{\Theta}_A) \cdot \mathrm{vec}(T(\boldsymbol{\Theta}_B))$.
1: Initialize $\boldsymbol{T}_\ell \leftarrow \boldsymbol{I}$   for all $\ell = 1, \dots, L-1$.
2: repeat
3:     for $\ell \in$ RandomPermutation$(1, \dots, L-1)$ do
4:
           $$\boldsymbol{T}_\ell \leftarrow \textsc{SolveLAP}\Big(\boldsymbol{W}_\ell^{(A)} \boldsymbol{T}_{\ell-1} \big(\boldsymbol{W}_\ell^{(B)}\big)^\top + \big(\boldsymbol{W}_{\ell+1}^{(A)}\big)^\top \boldsymbol{T}_{\ell+1} \boldsymbol{W}_{\ell+1}^{(B)} + \boldsymbol{b}_\ell^{(A} (\boldsymbol{b}_\ell^{(B)})^T\Big)$$
5:     end for
6: until convergence
```

A critical analysis of parameter matching algorithms for model merging reveals fundamental trade-offs between the generality of the transformation and the optimality of the optimization strategy. The authors in [32] propose a matching methodology that is constrained by its sequential optimization strategy, where rotation and rescaling symmetries are addressed in two separate, consecutive steps. This decoupled approach is fundamentally a greedy, path-dependent algorithm; by first committing to an optimal rotation matrix R, it constrains the subsequent search for a rescaling factor a. Consequently, it does not guarantee convergence to a global optimum, as a different initial rotation might have enabled a superior overall alignment. The authors of [32] acknowledge this deficiency, framing their method as a practical approximation that trades theoretical optimality for computational tractability.

In contrast, our proposed approach addresses the related problem of matching models under transformations composed of permutations and sign-flips. A crucial distinction lies in the optimization strategy: whereas the former method is sequential, our approach intertwines the optimization of permutations and scaling within each iteration. For a fixed cost matrix at a given step, it jointly computes the optimal permutation and sign-flips. However, this method's form of scaling is restricted to discrete values of $+1$ and -1, a notable limitation compared to the arbitrary, continuous scaling admitted by the former. The comparison highlights a central challenge in the field: one approach handles a more general scaling problem with a suboptimal sequential strategy, while the other employs a more principled intertwined strategy for a more restricted problem, yet both ultimately yield approximations. This underscores the need for a more general and powerful approach to model merging that can accommodate arbitrary symmetries with formal guarantees of optimality, an effort the authors of [32] note is non-trivial for their joint optimization problem.

B Graph Metanetworks

Graph Metanetworks (GMNs) [12] process FFNNs as graphs using conventional GNNs, leveraging permutation symmetries. The computational graph $G = (\mathcal{V}, \mathcal{E})$ uses $i \in \mathcal{V}$ for neurons and $(i,j) \in \mathcal{E}$ for edges from j to i. Vertex features $\mathbf{x}_V \in \mathbb{R}^{|\mathcal{V}| \times d_v}$ (biases) and edge features $\mathbf{x}_E \in \mathbb{R}^{|\mathcal{E}| \times d_e}$ (weights) are inputs. A T-iteration GMN is defined as:

$$\mathbf{h}_V^0(i) = \mathrm{INIT}_V(\mathbf{x}_V(i)), \qquad \mathbf{h}_E^0(i,j) = \mathrm{INIT}_E(\mathbf{x}_E(i,j)) \tag{Init}$$

$$\mathbf{m}_V^t(i) = \bigoplus_{j \in N(i)} \mathrm{MSG}_V^t(\mathbf{h}_V^{t-1}(i), \mathbf{h}_V^{t-1}(j), \mathbf{h}_E^{t-1}(i,j)) \tag{Msg}$$

$$\mathbf{h}_V^t(i) = \mathrm{UPD}_V^t(\mathbf{h}_V^{t-1}(i), \mathbf{m}_V^t(i)), \qquad \mathbf{h}_E^t(i,j) = \mathrm{UPD}_E^t(\mathbf{h}_V^{t-1}(i), \mathbf{h}_V^{t-1}(j), \mathbf{h}_E^{t-1}(i,j)) \tag{Upd}$$

$$\mathbf{h}_G = \mathrm{READ}(\{\mathbf{h}_V^T(i)\}_{i \in V}) \tag{Readout}$$

where READ is optional and is usually permutation-invariant (e.g. DeepSets).

Scale-Equivariant Message Passing: In order to respect scaling symmetries, Kalogeropoulos et al. [9] replace MSG and UPD of the standard Graph Metanetwork framework with *scale-equivariant* blocks. Define:

$$\mathrm{ScaleEq}(x_1, \dots, x_n) = \big[\Gamma_1 x_1, \dots, \Gamma_n x_n\big] \odot \rho\big(\mathrm{canon}(x_1), \dots, \mathrm{canon}(x_n)\big), \tag{11}$$

$$\mathrm{ReScaleEq}(y, e) = (\Gamma_y\, y) \odot (\Gamma_e\, e), \tag{12}$$

where the canonicalization function $\mathrm{canon}(x)$ is chosen based on the underlying symmetry. For **sign symmetry**, $\mathrm{canon}(x) = \mathrm{MLP}(x) + \mathrm{MLP}(-x)$ is used whereas for **positive scalings**, L2 normalization $\mathrm{canon}(x) = x/\|x\|$ is used. The $\mathrm{canon}(x)$ function removes scale, thus making ρ invariant, and Γ_i are learnable linear maps which are then combined with the scale invariant component to achieve scale equivariance. Then

$$\mathrm{MSG}_{\mathrm{SE}}(\mathbf{x}, \mathbf{y}, \mathbf{e}) = \mathrm{ScaleEq}\big[\mathbf{x}, \mathrm{ReScaleEq}(\mathbf{y}, \mathbf{e})\big], \quad \mathrm{UPD}_{\mathrm{SE}}(\mathbf{x}, \mathbf{m}) = \mathrm{ScaleEq}\big[\mathbf{x}, \mathbf{m}\big].$$

For any diagonal $D_\ell \succ 0$, $\mathrm{MSG}_{\mathrm{SE}}(D_\ell x,\, D_{\ell-1} y,\, D_\ell\, D_{\ell-1}^{-1} e) = D_\ell\, \mathrm{MSG}_{\mathrm{SE}}(x, y, e)$, and similarly for $\mathrm{UPD}_{\mathrm{SE}}$, thus satisfying equivariance for the scaling subgroup with respect to the central node x. Combining these with the permutation-equivariant GMN yields a model equivariant to the full group of scalings and permutations.

C Experimental details

C.1 Datasets and splits

Our experiments utilize the following dataset partitions. For the Implicit Neural Representation (INR) models, the train/validation/test split was 55,000/5,000/10,000 samples. For the Convolutional Neural Network (CNN) models, the splits varied by activation function: 5,976/1,465/7,433 for the Tanh variant and 6,024/1,535/7,567 for the ReLU variant.

The CNN models' functional loss—defined as the KL divergence between the output distributions of the original and reconstructed networks—was computed using images from the CIFAR-10 dataset. To ensure computational efficiency during training, we used a fixed, random subset of 10,000 CIFAR-10 images. For evaluation tasks such as generating interpolation lines, a distinct, held-out set of 10,000 images was used.

C.2 Hyperparameters

To ensure optimal performance, we conducted a comprehensive hyperparameter sweep for our primary models. The search spaces for the key hyperparameters for our Implicit Neural Representation (INR) and Convolutional Neural Network (CNN) autoencoder architectures are detailed in Table 3 and Table 4, respectively. The final hyperparameter values chosen for our experiments, which yielded the best performance on a smaller portion of the validation set, are indicated in bold. Further details on the final graph neural network architectures and optimizer settings for both the INR and CNN experiments are provided in Table 5 and Table 6.

Table 3: Hyperparameters explored for INR experiments. The decoder hidden dimensions are computed from a base hidden size of 128.

Hyperparameter	**Search Space**
Decoder hidden layers	[], [256], **[256, 512]**, [256, 512, 1024]
Learning rate	0.01, **0.001**, 0.0001

Table 4: Hyperparameters explored for CNN experiments. Decoder hidden dimensions are computed from the base hidden size (128 or 256).

Hyperparameter	**Search Space**
Temperature	**0.5**, 1.0, 1.5
Learning rate	**0.001**, 0.0005, **0.0001**
Encoder Backbone hidden dimension	128, **256**
Decoder hidden layers (base=128)	[256, 512], [256, 512, 1024]
Decoder hidden layers (base=256)	**[512, 1024]**, [512, 1024, 2048]

Table 5: Final model and optimizer parameters for the INR experiments.

Hyperparameter	**ScaleGMN**	**NeuralGraphs**
Epochs	300	300
Optimizer	AdamW	AdamW
Learning Rate	1×10^{-3}	1×10^{-3}
Learning Rate Warmup Steps	1K	1K
Weight Decay	1×10^{-2}	5×10^{-4}
GNN Layers	4	4
Hidden Dimension	128	128
Readout Method	Full graph	Last-layer pooling
Decoder Hidden Layers	[256, 512]	[256, 512]
Decoder Activation	SiLU	SiLU

Table 6: Final model and optimizer parameters for the CNN experiments.

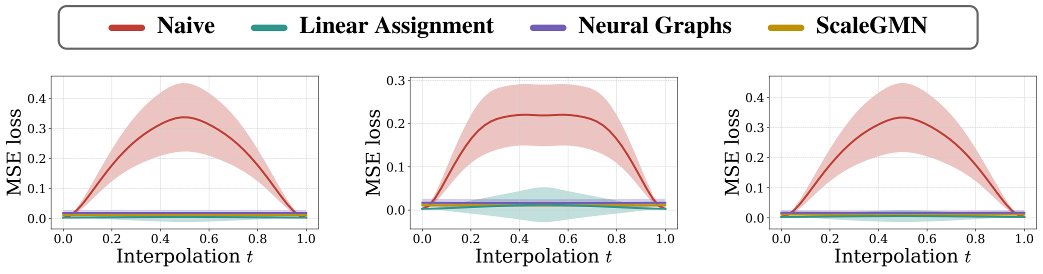
Hyperparameter	**ScaleGMN (ReLU)**	**ScaleGMN (Tanh)**	**NeuralGraphs**
Epochs	300	300	300
Optimizer	AdamW	AdamW	AdamW
Learning Rate	1×10^{-3}	1×10^{-3}	1×10^{-4}
Learning Rate Warmup Steps	1K	1K	1K
Weight Decay	1×10^{-2}	1×10^{-2}	5×10^{-4}
GNN Layers	4	4	4
Hidden Dimension	256	256	256
Readout Method	Full graph	Full graph	Last-layer pooling
Decoder Hidden Layers	[512, 1024]	[512, 1024]	[512, 1024]
Decoder Activation	SiLU	SiLU	SiLU

D Additional results

This section presents additional results that support the claims made in this study. We also include a complete set of experiments to provide a comprehensive view of the proposed method.

D.1 INR interpolation without noise

The rationale behind introducing a noise perturbation is that the very design of a ScaleGMN encoder ensures that functionally equivalent input networks differing only in a group action being applied will be mapped to the same point. For completeness, we provide the results of performing the interpolation experiment without noise for the INR setting in Figure 7. The observed behavior of the ScaleGMN autoencoder aligns with theoretical expectations, as all three transformations are mapped to an identical point in the latent space, yielding a flat interpolation. This invariance to permutations is similarly an anticipated characteristic of Neural Graphs. Conversely, upon the introduction of scaling, there is no guarantee that both models will be encoded to the same latent point, a phenomenon illustrated in Figure 6. Nonetheless, it is empirically observed that the reconstructed models coincide in practice (Figures 7b and 7c). This ability is attributed to the learning process guided by the functional loss of the autoencoder architecture. Despite the model weights being sign-swapped, the underlying functions they represent remain equivalent, and the autoencoder has consequently learned to assign a common canonical mapping to both.



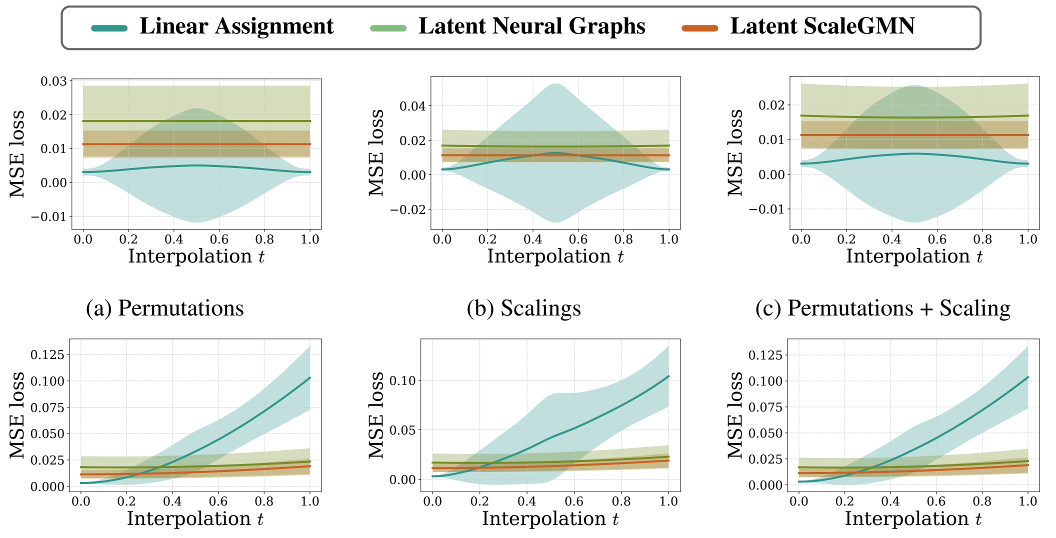
(a) Permutations (b) Scaling (c) Permutations + Scaling

Figure 7: INR interpolation experiments without noise added.

D.2 INR interpolation in latent space

In Section 5, we focused on weight space interpolation, as typically done in the model merging literature. However, latent space interpolation is also common practice, as demonstrated by Luigi et al. [20]. Within the autoencoder framework, interpolation in weight space presents a notable computational advantage over interpolation in latent space. The latter method requires generating a sequence of N latent points, each needing a separate forward pass through the decoder. Consequently, if the decoder's computational complexity were super-linear with respect to the number of input network parameters, this process would be markedly less efficient than weight-space interpolation, which exhibits linear complexity. However, as we use an MLP decoder, its decoding cost is also linear, thereby neutralizing this specific efficiency gain.

We therefore investigate interpolation directly in the latent space produced by our autoencoders. As expected, Figure 8 shows that latent space interpolation on functionally identical INR pairs performs on par with our canonicalized weight space interpolation (Figure 3). This demonstrates that our learned latent representations are robust to perturbations in the encoded weights, which, as discussed in Section 6.1, will prove valuable for downstream tasks beyond model merging.



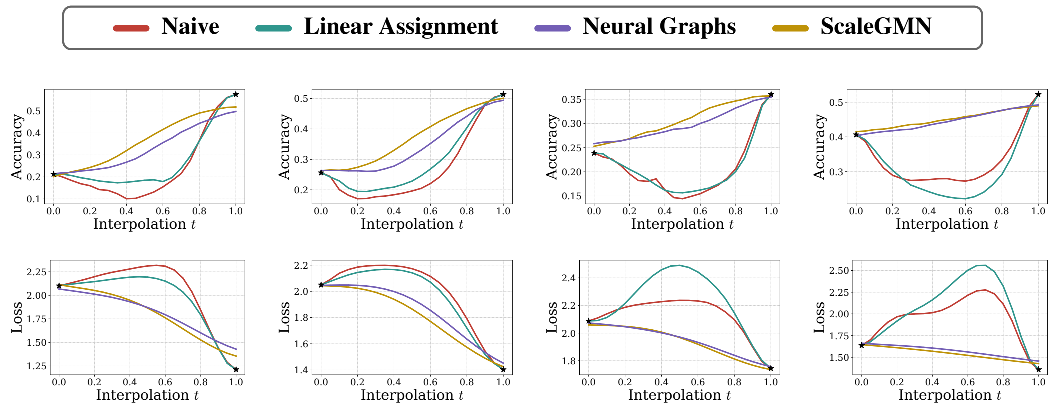
(a) Permutations (b) Scalings (c) Permutations + Scaling

(d) Permutations (Noise) (e) Scalings (Noise) (f) Permutations + Scaling (Noise)

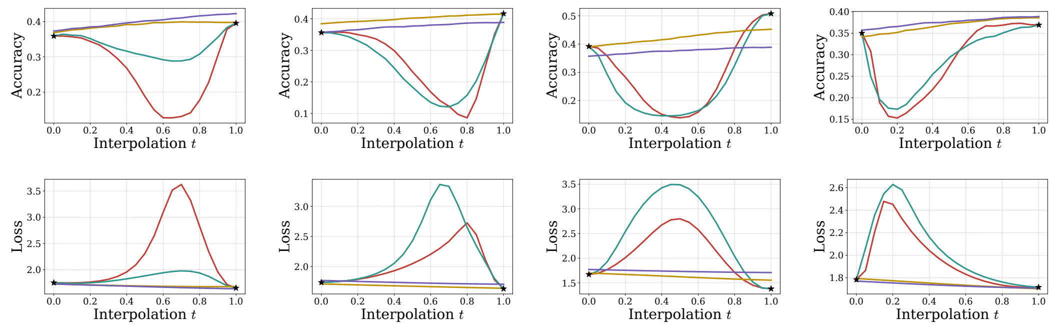
Figure 8: INR latent interpolation experiments.

D.3 CNN interpolation additional results

The interpolation results presented previously in Figures 4 and 5 were derived from Convolutional Neural Networks (CNNs) selected for their high accuracy, as these represent the most performant models and thus the most interesting for model merging in practical terms. However, given the compact architecture, a significant portion of independently trained models from the SmallCNN Zoo dataset achieve suboptimal performance, with accuracies on the CIFAR dataset falling as low as 20%. In Figure 9 we present interpolation curves for randomly chosen model pairs to provide a more realistic and comprehensive view of the interpolation behavior across a wider spectrum of model quality. In contrast to the Implicit Neural Representation (INR) experiments, the observed irregularities demonstrate that these CNNs constitute a more heterogeneous population of networks.



(a) Models with ReLU activation.



(b) Models with Tanh activation.

Figure 9: Interpolation curves for CNN models comparing different activation functions across 4 model pairs each. For each block, the top row displays accuracies and the bottom row displays losses.