HIERARCHICAL GRAPH METANETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

The emerging field of weight space learning has demonstrated the potential to enable metanetworks, i.e. models that process neural network (NN) data, to support several steps of the machine learning lifecycle, from model optimisation to evaluation and adaptation. However, this potential has yet to materialise beyond small-scale MLPs or CNNs: most current models, including message-passing based graph metanetworks (GMNs), operate exclusively at the finest level of granularity, processing every neuron and weight of the input model. Their cost therefore scales with the number of parameters. Yet NNs are *inherently organised hierarchically*, with neurons forming layers and layers forming blocks and modules. Additionally, several of their properties, such as Lipschitz upper bounds and generalisation measures, are compositions of layer- and block-level quantities. To incorporate this inductive bias, we introduce Hierarchical Graph Metanetworks (H-GMNs), which leverage the input architecture's hierarchy to coarsen and refine the graph using permutation-symmetry-preserving pooling and un-pooling layers. Further, we extend H-GMNs to Transformers by introducing a graph construction that preserves permutation symmetries and enables multi-level pooling. Our evaluation includes new ResNet-18 and ViT model zoos with up to 21M parameters per model, on tasks spanning model evaluation (an analytic Lipschitz bound proxy, generalisation and robustness prediction), testing (backdoor detection) and editing (pruning). Experimental results show that, compared to non-equivariant alternatives and flat GMNs, H-GMNs are markedly more sample-efficient, and their cost remains nearly constant as metanetwork depth grows, allowing deeper and more expressive processing where flat GMNs run out of memory, without compromising accuracy, which in certain cases considerably improves.

1 INTRODUCTION

Every training loop of a machine learning model leaves behind a collection of artefacts - weights, gradients, computational traces, optimisation trajectories - a considerable fraction of which are subsequently released on public model repositories. Notwithstanding their abundance, such artefacts have traditionally been regarded as terminal outputs of the training process and rarely as objects of analysis in their own right. It was not until recently that a growing body of work started to show that these objects, recently grouped under the term *neural artefacts*, are in fact information-rich (Schürholt et al., 2021; Huben et al., 2024). In particular, they have been used to evaluate and edit model behaviour, such as generalisation (Eilertsen et al., 2020; Unterthiner et al., 2020), sycophancy (Chen et al., 2024), hallucinations (Frasca et al., 2026) and fairness (Vig et al., 2020; Karvonen & Marks, 2025; Yang et al., 2025), in neural field processing (Erkoç et al., 2023), in model retrieval (Horwitz et al., 2025), comparison (Pal et al., 2025) and merging (Soro et al., 2026).

In parallel, the structure of these objects has itself become a subject of study. Neural network (NN) parameters admit symmetries (Godfrey et al., 2022), i.e. transformations that leave the function computed by the network unchanged, and they are organised according to a graph topology inherited from the underlying architecture. These observations have motivated the design of dedicated, structure-aware architectures for processing NN parameters, commonly referred to as *metanetworks* (alternatively, neural functionals or meta-models) (Navon et al., 2023; Zhou et al., 2023a; Lim et al., 2024; Kofinas et al., 2024; Kalogeropoulos et al., 2024; Vo et al., 2025; Tran et al., 2025; Ballerini et al., 2026), predominantly within the field of Weight Space Learning (WSL), in which NN parameters constitute the primary object of study. Among these, graph metanetworks (GMNs) represent an

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.

NN as a parameter graph, whose vertices and edges correspond to neurons and weights, and process it with a graph neural network (Lim et al., 2024; Kofinas et al., 2024; Kalogeropoulos et al., 2024).

Existing GMNs, however, operate at a single level of resolution, that of individual neurons and weights, and equivariant metanetworks have, to date, been applied predominantly to modest MLPs, CNNs and Transformers. Methods that do scale to larger models, notably those that tokenise weight matrices and process the resulting sequences with a generic Transformer (Schürholt et al., 2024; Soro et al., 2026), account for neither the graph structure nor the symmetries of the input network, and therefore dispense with the inductive biases to which structure-aware metanetworks owe their sample efficiency. Neural networks, moreover, are constructed modularly, with channels forming layers, layers forming blocks, and blocks forming complete architectures. Accordingly, several properties of interest are naturally characterised at these coarser levels of abstraction; Lipschitz constant upper bounds, for example, arise as compositions of layer- and block-level quantities. Although hierarchical structure of this kind has been exploited in graph representation learning (Ying et al., 2018; Wang et al., 2020), it has not, to the best of our knowledge, been leveraged in the design of metanetworks.

In this paper, we propose *Hierarchical Graph Metanetworks* (H-GMNs), which process the parameter graph (Lim et al., 2024; Kofinas et al., 2024) jointly with a sequence of coarsenings derived from the architecture of the input network. *Local pooling* layers consolidate each architectural component, such as a layer, an attention head or a block, into a single node, invariantly to the permutations acting within that component (Hecht-Nielsen, 1990) and equivariantly to permutations of interchangeable components, such as the heads of a self-attention layer. The framework is not restricted to permutations: it extends to other parameter symmetries, such as scalings (Godfrey et al., 2022; Kalogeropoulos et al., 2024), provided that all layers, including pooling and un-pooling, respect them. Since the coarsened graphs scale with the number of architectural components rather than with the parameter count, deeper and more expressive processing at the coarse levels, including graph transformers (Ma et al., 2023), becomes computationally affordable. For editing tasks, which require outputs at the resolution of individual neurons, we introduce equivariant *un-pooling* layers within a U-Net-style encoder–decoder (Ronneberger et al., 2015): un-pooling broadcasts coarse representations back to finer levels, where skip connections combine them with the fine-grained representations of the encoder. H-GMNs thereby bridge two existing paradigms and contain both as limiting cases: statistics-based metanetworks (Eilertsen et al., 2020), which summarise each layer through symmetric statistics before any message passing, and flat GMNs (Lim et al., 2024; Kofinas et al., 2024), which perform message passing exclusively on the parameter graph. Finally, to extend H-GMNs to Transformers, we propose a per-head graph construction for multi-head attention that preserves its permutation symmetries and enables pooling at the level of individual heads. Our contributions can be summarised as follows.

- **A multi-resolution view of metanetworks.** H-GMNs process the parameter graph together with a sequence of architecture-derived coarsenings, and contain statistics-based metanetworks and flat GMNs as limiting cases of a single family.
- **Symmetry-respecting pooling and un-pooling layers**, under which predictions are invariant and edits are equivariant to neuron permutations, yielding coarse graphs whose size scales with the number of architectural components rather than with the parameter count.
- **A per-head parameter-graph construction for multi-head attention**, whose node groups expose both within-head and head-relabelling symmetries to pooling.
- **Three new model zoos** (ResNet-18, ViT-S/16 and backdoored ResNet-18) with hyperparameter diversity, together with a task whose ground truth is known and compositional, namely a Lipschitz upper bound.
- **An empirical evaluation** spanning model evaluation, testing and editing, which shows that H-GMNs match or improve upon flat GMNs at lower computational cost, with gains most pronounced in low-data regimes and on the larger ViT zoo.

2 Related Work

Weight Space Learning & Metanetworks. Initial attempts to learn to process NN parameters focused on simple statistics (Eilertsen et al., 2020; Unterthiner et al., 2020; Martin et al., 2021) or on parameter flattening (Schürholt et al., 2021). Concurrently, a separate line of work focused

on designing intricate **Tensor-based linear layers** to account for symmetries stemming from the model architecture. Navon et al. (2023); Zhou et al. (2023a;b); Sverdlov et al. (2025) examined **permutation** equivariance for MLPs and CNN, Herrmann et al. (2024) extended this to RNNs and Zhou et al. (2024) to general architectures, deriving maximally expressive equivariant layers for RNNs and Transformers. Moving beyond permutations, Tran et al. (2024) and Vo et al. (2025) proposed **scale**-equivariant layers. Zhang et al. (2025) investigated transformer **rotation** symmetries, while more general **GL-equivariant** layers have been proposed for self-attention and LoRA weights (Tran et al., 2025; Putterman et al., 2025). Relevant works also include (Andreis et al., 2024), where regularisation is used for **approximate invariance** and (Shamsian et al., 2024) that propose weight-space **augmentations**. **Graph Metanetworks (GMNs):** Kofinas et al. (2024) and Lim et al. (2024) proposed an alternative viewpoint, where NNs are converted into a special class of graphs to be processed with Graph Neural Networks (GNNs) with symmetry breaking wherever necessary, focusing on MLPs, CNNs attention, normalization, and group-equivariant linear layers. Subsequently, Kalogeropoulos et al. (2024) adapted this paradigm to account for a variety of scaling symmetries. Applications of GMNs, beyond standard property prediction, include (Ballerini et al., 2026) for NeRFs and (Kalogeropoulos et al., 2025) for unsupervised model editing, while Knyazev et al. (2025) used a GMN to forecast future weights and accelerate training. GMNs are adaptable to new architectures, take into account the underlying NN graph topology and have demonstrated better experimental performance. Therefore, they are selected as a backbone for the present work.

Towards larger models. However, none of the above methods has been applied to networks beyond the tens-of-thousands-of-parameters regime. The sole exception is Knyazev et al. (2025), where the largest model used for meta-training was a GPT2-style Transformer ($\sim$1.6M parameters) and the largest model evaluated on was a Llama3-style model ($\sim$111M parameters), for which 5% edge sampling was necessary and inference took over 10 sec/prediction. Towards this direction, Schürholt et al. (2024) proposed an autoencoder, named *SANE*, to learn self-supervised representations of tokenised weight matrices, a work extended in (Falk et al., 2025b), and adapted for learnable model merging (Soro et al., 2026) and dataset-NN parameter representation alignment (Asefaw et al., 2026). Nevertheless, these methods lack appropriate inductive biases (graph-structure aware, equivariant) and typically need larger training datasets and tend to underperform (see section 7). Finally, probing methods sidestep parameter count, representing a model via its responses to learned probes, fed to the whole network (Kahana et al., 2025) or a single layer (Horwitz et al., 2025).

Hierarchical GNNs. Flat message-passing GNNs suffer from over-squashing and a limited effective receptive field (Alon & Yahav, 2021; Topping et al., 2022; Di Giovanni et al., 2023). A common remedy is to process the graph at multiple scales; hierarchical methods differ mainly in the origin of the hierarchy. **Learnable coarsening algorithms.** Dense soft-clustering, e.g. DiffPool (Ying et al., 2018) and MinCutPool (Bianchi et al., 2020), and sparse node selection, e.g. graph U-Nets (Gao & Ji, 2019), SAGPool (Lee et al., 2019), learn the coarsening end-to-end. **Non-learnable coarsening algorithms.** Another line computes the hierarchy from the graph topology with generic coarsening algorithms, e.g. Graclus (Defferrard et al., 2016), Louvain (Rampášek & Wolf, 2021; Zhong et al., 2023), BFS-based pooling (Cao et al., 2023) and recursive clustering (Vonessen et al., 2024; Finder et al., 2025). These are cheap and stable but agnostic to the data distribution. Scalability-oriented partitioning (Chiang et al., 2019; Liao et al., 2018) and virtual-node rewiring (Qian et al., 2024) are related but do not target hierarchical representation learning. **Hierarchies provided by the data.** When the data has an intrinsic multi-level organisation, it can be used directly: junction trees (Fey et al., 2020), protein residue-backbone-atom levels (Wang et al., 2023), or given subgraphs (Kim & Oh, 2024). Wang et al. (2020) formally characterises the symmetries of such nested structures. This is the strategy we also adopt, given the pre-defined modular manner, in which NNs are designed.

3 Preliminaries: Graph Metanetworks

Let $f_{C,\boldsymbol{\theta}}(\cdot)$ be an NN parametrised by its computational graph C and a vector of learnable parameters $\boldsymbol{\theta}$, with $f_{C,\boldsymbol{\theta}} \in \mathcal{F}$, where $\mathcal{F}$ is a function space. The bigger picture surrounding our approach is the design of *metanetworks*, learnable functions that process NNs, i.e. $\mathfrak{F} : \mathcal{F} \to \mathcal{F}'$, where $\mathcal{F}'$ can be a vector space or a function space itself, depending on the task.

Graph construction. To model $\mathfrak{F}$, we follow a common WSL paradigm that treats NN datapoints as *graphs* (Lim et al., 2024; Kofinas et al., 2024; Kalogeropoulos et al., 2024). Therefore, the

first step is to construct a graph faithfully representing each NN by a procedure $\mathfrak{G} : \mathcal{F} \to \mathcal{G}$, where $\mathcal{G}$ is a graph space, that receives an NN $f_{C,\boldsymbol{\theta}}$ and yields a *parameter graph* $\mathfrak{G}(f_{C,\boldsymbol{\theta}}) = G = (\mathcal{V}, \mathcal{E}, \mathbf{x}_V, \mathbf{x}_E)$, carefully designed such that $f_{C,\boldsymbol{\theta}}$ can be retrieved. Typically, $\mathfrak{G}$ traverses the computational graph C layer-by-layer, populating the set $\mathcal{V}$ with one node for each output channel of each layer and the set $\mathcal{E}$ with one edge for each pair of input-output channels that interact (i.e. when the latter is a function of the former). The exact wiring depends on the layer type: *channel-mixing* layers (e.g. linear, convolution, attention projections), connect input with output nodes by a complete bipartite graph, and *channel-wise* layers (e.g. BatchNorm, LayerNorm), connect them node-wise. Regarding the features, although $\mathfrak{G}$ differs between architectures, a general rule of thumb is that biases and weights become vertex and edge features, respectively. Additionally, the construction often allows $\mathbf{x}_V, \mathbf{x}_E$ to inherit the symmetries from the NN's parameters (e.g. permutations, scalings). Regardless of the input $f_{C,\boldsymbol{\theta}}$, the graph G exhibits a structural regularity: it is, by construction, a directed (edges start from input channel nodes and terminate at output ones), acyclic (no node points back to a node corresponding to a channel of a previous NN layer), multi-partite graph (nodes corresponding to channels of the same layer are never connected). Figure 1 depicts the Parameter Graph for a ResNet18 block. A detailed description of the graph construction can be found in section A.2.

Message Passing. The second step is to sequentially process the parameter graph with multiple message passing layers (Graph Metanetworks - GMNs). These closely follow the design principles of GNNs consisting of vertex- and edge- message and update functions, as well as positional encodings for symmetry breaking, all potentially modified to account for specific NN symmetries. Please refer to (Lim et al., 2024; Kalogeropoulos et al., 2024) for additional details. With a slight notation abuse, we will denote a GNN with message passing layers also with $\mathfrak{F} : \mathcal{G} \to \mathcal{G}$, with $\mathfrak{F}(\mathcal{V}, \mathcal{E}, \mathbf{h}_V, \mathbf{h}_E) = (\mathcal{V}, \mathcal{E}, \mathbf{h}'_V, \mathbf{h}'_E)$, where $(\mathcal{V}, \mathcal{E})$ the graph topology that remains unchanged and $\mathbf{h}_V, \mathbf{h}_E, \mathbf{h}'_V, \mathbf{h}'_E$, the input and output vertex and edge features. The last step is a **Readout** function $\mathfrak{R} : \mathcal{G} \to \mathcal{F}'$ that projects the output of the last metanetwork layer to the output space of interest. This will be a prediction head combined with either a global invariant aggregation for NN behaviour prediction, or with the inverse function $\mathfrak{G}^{-1}$ to project back to an output NN, for NN editing.

4 Hierarchical Graph Metanetworks

NNs are compositional: channels form layers, layers form blocks, and blocks form entire modules. To account for this, we propose a progressive coarsening scheme producing a sequence of graphs $(G_i)_{i=1}^M$, where M is the number of levels. Node and edge features of graphs higher in the hierarchy consolidate information from lower levels by learnable *graph pooling layers* and, in certain cases, can pass on information to lower levels by *graph unpooling*. Unlike many hierarchical models, which additionally learn how to coarsen the graph (Ying et al., 2018; Lee et al., 2019), our graph sequence is derived from the input's architecture: each step aggregates the nodes and edges of the same architectural component (e.g. a layer, attention head, or block). Besides giving the model direct access to component-level structure, coarsening yields graphs that scale with the NN depth, instead of width^2 $*$ depth, allowing for deeper processing using wider GNN layers or even Graph Transformers (Ma et al., 2023) and extending the receptive field to capture longer-range interactions.

Pooling Layer. Let $G_i = (\mathcal{V}_i, \mathcal{E}_i, \mathbf{h}_V^{(i)}, \mathbf{h}_E^{(i)})$ be a graph and $(\mathcal{V}_{i+1}, \mathcal{E}_{i+1})$ a coarsened topology. A *pooling* layer $\mathrm{Pool}_i : \mathcal{G} \to \mathcal{G}$, with $\mathrm{Pool}_i(\mathcal{V}_i, \mathcal{E}_i, \mathbf{h}_V^{(i)}, \mathbf{h}_E^{(i)}) = (\mathcal{V}_{i+1}, \mathcal{E}_{i+1}, \mathbf{h}_V^{(i+1)}, \mathbf{h}_E^{(i+1)})$ is specified by a surjective node-grouping map $g_i : \mathcal{V}_i \to \mathcal{V}_{i+1}$ and node and edge pooling functions $\rho_{V,i}, \rho_{E,i}$, each aggregating the embeddings of a group to a single embedding:

$$\mathbf{h}_V^{(i+1)}(u) = \rho_{V,i}\left(\left(\mathbf{h}_V^{(i)}(v)\right)_{g_i(v)=u}\right), \mathbf{h}_E^{(i+1)}(u,w) = \rho_{E,i}\left(\left(\mathbf{h}_E^{(i)}(s,t)\right)_{(g_i(s),g_i(t))=(u,w)}\right), \tag{1}$$

with $u \in \mathcal{V}_{i+1}, (u,w) \in \mathcal{E}_{i+1}$ and $\mathcal{E}_{i+1} = \{(u,w) : \exists s,t \in \mathcal{V}, \text{ with } (s,t) \in \mathcal{E}_i,\ g_i(s), g_i(t) = u, w\}$.

Each g_i is derived from the architecture and depends on the hierarchy levels it connects. A simple H-GMN allows communication only between consecutive layers (akin to hierarchical CNNs) and composes pooling with i-level GNNs $\mathfrak{F}_i : \mathcal{G} \to \mathcal{G}$ acting on G_i, $i = 1, \ldots, M$, where $G_1 = G$:

$$\mathfrak{F}_{\text{pred}}(f_{C,\boldsymbol{\theta}}) = \mathfrak{R} \circ \mathfrak{F}_M \circ \mathrm{Pool}_{M-1} \circ \cdots \circ \mathrm{Pool}_1 \circ \mathfrak{F}_1 \circ \mathfrak{G}(f_{C,\boldsymbol{\theta}}), \tag{2}$$

Each $\mathfrak{F}_i$ comprises $k_i \geq 0$ message-passing layers, with $k_i = 0$ corresponding to the identity map.

Resolution as a design axis. The hierarchy exposes a choice that existing metanetworks fix implicitly, namely the resolution at which processing takes place. H-GMNs are indexed by the number of levels M and the number of message-passing layers k_i applied at each level. With $k_i = 0$ at every level, fixed symmetric statistics as $\rho_{V,1}, \rho_{E,1}$, a final pooling that concatenates all layer embeddings into a single node, and an MLP readout, an H-GMN summarises every layer before any message passing and recovers StatNN (Unterthiner et al., 2020), up to the choice of statistics and predictor. With a single level ($M = 1$), it reduces to a flat GMN. Intermediate choices spend a few layers at neuron resolution, where interactions between individual weights are visible, and the remaining layers at coarser resolutions, where the graph is orders of magnitude smaller (see table 1).

Unpooling Layer. Consider given vertex and edge sets $\mathcal{V}_i, \mathcal{E}_i$, a coarsening map $g_i : \mathcal{V}_i \to \mathcal{V}_{i+1}$, and embeddings $\mathbf{h}_V^{(i+1)}, \mathbf{h}_E^{(i+1)}$ of G_{i+1}. The *unpooling* layer $\mathrm{UnPool}_i : \mathcal{G} \to \mathcal{G}$, with $\mathrm{UnPool}_i(\mathcal{V}_{+1}, \mathcal{E}_{i+1}, \mathbf{h}_V^{(i+1)}, \mathbf{h}_E^{(i+1)}) = (\mathcal{V}_i, \mathcal{E}_i, \mathbf{h}_V^{(i)}, \mathbf{h}_E^{(i)})$ copies the embedding of a node in G_{i+1} to every node and edge that maps to it based on g_i:

$$\mathbf{h}_V^{(i)}(v) = \mathbf{h}_V^{(i+1)}\left(g_i(v)\right),\ v \in \mathcal{V}_i, \qquad \mathbf{h}_E^{(i)}(s,t) = \mathbf{h}_E^{(i+1)}\left((g_i(s), g_i(t))\right), \qquad (s,t) \in \mathcal{E}_i. \tag{3}$$

Editing pipeline. Editing tasks require output at the same resolution as the input. Naively broadcasting an intermediate level representation back to the parameter graph would assign every neuron in a layer the same coarse embedding, discarding the fine-grained distinctions an edit might require. Instead, we employ a U-Net-style encoder-decoder (Ronneberger et al., 2015; Gao & Ji, 2019): the method in eq. (2) serves as an encoder, and a decoder climbs up the resolution levels, via unpooling, concatenating with the encoder's embeddings and further processing at each resolution, giving the final GNN stage access to both global structure and local embeddings.

In particular, the encoder is a H-GMN as in eq. (2) without readout, and $G_i = (\mathcal{V}_i, \mathcal{E}_i, \mathbf{h}^{(i)})$ is the GNN output at level i, where we use the shorthand $\mathbf{h}^{(i)} = (\mathbf{h}_V^{(i)}, \mathbf{h}_E^{(i)})$ for the encoder embeddings. The decoder returns to the finest level, unpooling at each step, combining with the encoder embeddings of that level through a skip connection and processing with a GNN layer $\tilde{\mathfrak{F}}_i$ on $\tilde{G}_i = (\mathcal{V}_i, \mathcal{E}_i, \tilde{\mathbf{h}}^{(i)})$, where $\tilde{\mathbf{h}}^{(i)} = (\tilde{\mathbf{h}}_V^{(i)}, \tilde{\mathbf{h}}_E^{(i)})$ are the decoder embeddings:

$$\begin{aligned} &\mathbf{h}^{(1)} = \mathfrak{F}_1(\mathfrak{G}(f_{C,\boldsymbol{\theta}})), \qquad \mathbf{h}^{(i)} = \mathfrak{F}_i\big(\mathrm{Pool}_{i-1}(G_{i-1})\big), \quad i = 2, \dots, M, \quad \tilde{\mathbf{h}}^{(M)} = \mathbf{h}^{(M)}, \\ &(\mathcal{V}_i, \mathcal{E}_i, \tilde{\mathbf{h}}_a^{(i)}) = \mathrm{UnPool}_i(\tilde{G}_{i+1}), \quad \tilde{\mathbf{h}}_b^{(i)} = \tilde{\mathbf{h}}_a^{(i)} \,\|\, \mathbf{h}^{(i)}, \quad \tilde{\mathbf{h}}^{(i)} = \tilde{\mathfrak{F}}_i\big(\mathcal{V}_i, \mathcal{E}_i, \tilde{\mathbf{h}}_b^{(i)}\big), \end{aligned} \tag{4}$$

where $i = 1, \dots, M-1$ and concatenation $\|$ happens across the channel dimension. The output of the pipeline is $\mathfrak{F}_{\mathrm{edit}}(f_{C,\boldsymbol{\theta}}) = \mathfrak{G}^{-1}\big(\psi(\tilde{\mathbf{h}}^{(1)})\big)$, where ψ is a task-specific node- and edge-wise head.

5 Scaling to Transformers: Multi-Head Self-Attention Graphs

Multi-head self-attention layers (MHSA) are understudied within the GMN paradigm. Lim et al. (2024) and Kofinas et al. (2024) treat attention projection as standard linear layer parameters, imperfectly modelling permutation symmetries. Knyazev et al. (2025) rectifies this and is closer to our construction, but introduces unnecessary auxiliary vertices and edges. Below, we provide a general graph construction method that seamlessly allows MHSA processing by H-GMN.

Self-attention. Consider an input containing n elements $\mathbf{x} \in \mathbb{R}^{d_{\text{in}} \times n}$ and a self-attention function $f_{C,\boldsymbol{\theta}}(\mathbf{x}) = (\mathbf{W}_v\mathbf{x})\sigma\big((\mathbf{W}_q\mathbf{x})^T(\mathbf{W}_k\mathbf{x})\big)$, where $\mathbf{W}_q, \mathbf{W}_k \in \mathbb{R}^{d_k \times d_{\text{in}}}$ project queries and keys into the same space to compute the scores (dot products), $\sigma(\cdot)$ a non-linear function converting scores to attention coefficients (e.g. softmax after scaling with $1/\sqrt{d}$) and $\mathbf{W}_v \in \mathbb{R}^{d_{\text{v}} \times d_{\text{in}}}$ the value matrix (biases are omitted for simplicity). To construct the parameter graph, we follow the standard linear layer strategy, i.e. vertex-channel/edge-weight mapping (similar to CNNs (Lim et al., 2024), the dimension n should not affect the graph, since it is data-dependent), albeit with crucial differences.

Let $\mathcal{V}$ be the input vertices corresponding to $\mathbf{x}$, i.e. $|\mathcal{V}| = d_{\text{in}}$. First, we construct a set of vertices $\mathcal{V}_v$, with $|\mathcal{V}_v| = d_v$ and connections $\mathcal{E}_v = \mathcal{V} \times \mathcal{V}_v$ with $\mathbf{x}_E(i,j) = [\mathbf{W}_v(j,i)]$ as with all linear layers. Then, we construct a set $\mathcal{V}_k$, with $|\mathcal{V}_k| = d_k$ and connections $\mathcal{E}_k = \mathcal{V} \times \mathcal{V}_k$ with corresponding edge features $\mathbf{x}_E(i,j) = [\mathbf{W}_q(j,i), \mathbf{W}_k(j,i)]$. This reflects that these weight matrices are synchronised and share permutations. Finally, we create connections $\mathcal{E}_{kv} = \mathcal{V}_k \times \mathcal{V}_v$ with constant features and amend the latter two edge sets with different edge types to discern them from linear layers.

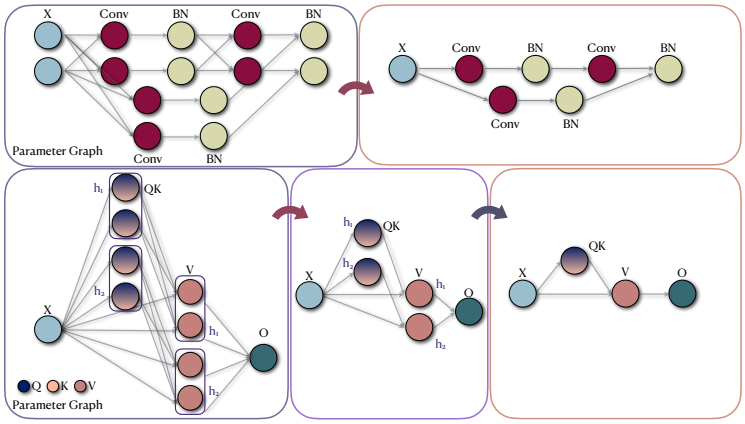


Figure 1: **Top**: Pooling on ResNet basic block. Red arrow indicates pooling across the width of each layer, resulting in the coarsened layer graph. **Bottom**: Pooling on an MHSA block. Pooling across the width leaves head vertex sets visible. Further pooling across heads leads to the layer graph level.

Multi-head. In the classical multi-head setting, all head outputs are further projected and mixed into a single output, i.e. $f_{C,\boldsymbol{\theta}}(\mathbf{x}) = \sum_h \mathbf{W}_o^h(\mathbf{W}_v^h\mathbf{x})\sigma\big((\mathbf{W}_q^h\mathbf{x})^T(\mathbf{W}_k^h\mathbf{x})\big)$, where $\mathbf{W}_o^h \in \mathbb{R}^{d_{\text{out}} \times d_v}$. Therefore, we repeat the creation of new vertices and edges for each head separately and additionally create a new vertex set $\mathcal{V}_o$, with $|\mathcal{V}_o| = d_{\text{out}}$, with connections $\mathcal{E}_o = \{(i,j)|i \in \mathcal{V}_v^h, j \in \mathcal{V}_o, \forall h\}$, with edge features $\mathbf{x}_E(i,j) = \mathbf{W}_o^h(j,i)$,where the superscript h has been used to extend the above notation (see fig. 1 for an illustration). In section A.7, we provide a proof and additional explanations on how parameter symmetries are inherited to the vertex/edge features for equivariant processing.

6 Analysis & Instantiations

Pooling and Unpooling Symmetries. Instantiating pooling layers in a symmetry-respecting manner is tightly related to how vertices are grouped at each level of hierarchy. To simplify, we focus on *permutation symmetries*. The most common case is when the grouping contains *all* vertices that are equivalent to each other, e.g. coarsening the parameter graph into a layer graph. To make the metanetwork invariant, the pooling function should be invariant to valid permutations, and those permutations are no longer applicable in the next levels of hierarchy (*collapsed symmetries*). Another case arises when the grouping contains *a subset* of equivalent vertices, e.g. when separately grouping the vertices corresponding to different attention heads. Again here, invariance to valid permutations is necessary, but the resulting vertices remain equivalent to the ones that did not participate in the grouping. In other words, certain permutations carry over in the next level of hierarchy, and subsequent layers may be equivariant to them (*surviving symmetries*). To achieve metanetwork equivariance when editing, the concatenation trick of eq. (4) is key, as the original symmetries are carried over to the output and remain unaffected when broadcasting invariants with unpooling.

In practice, we use classical permutation-invariant aggregators for all pooling functions and account for non-valid permutations (when pooling inputs that are not equivalent, e.g. pooling vertices of the layer graph) with positional encodings. For both vertex and edge pooling we found that a simple DeepSets architecture (Zaheer et al., 2017) with several symmetric statistics (e.g. mean, max, standard deviation as in PNA (Corso et al., 2020)) was sufficient for our experiments. Note that our approach can be seamlessly extended to continuous symmetries, such as scaling, by modifying the pooling functions to be invariant to those as well, e.g. see (Kalogeropoulos et al., 2024).

ResNet Hierarchies. The simplest case of an MLP or a CNN admits a two-level hierarchy, from parameters to layers. Additional interest is found in ResNets, where an extra level arises due to their block structure (Figure 1 illustrates a ResNet-18 basic block). Pool_1 groups neurons collapsing each layer into one vertex and each weight matrix into one edge; then Pool_2 groups layers of a residual block into one vertex and one edge. Since Pool_1 already collapses every symmetry of $\boldsymbol{\theta}$, the pipeline is already invariant at this point and subsequent functions can be chosen arbitrarily.

Transformer hierarchies. NNs that contain MHSA blocks add an extra hierarchy level, that of heads. Pool_1 groups neurons at each block and head, collapsing within-head symmetries, but retaining cross-head ones, as mentioned before. Thus, subsequent metanetwork layers are equivariant to head permutations, until Pool_2 that groups heads into a layer graph, as above (fig. 1, bottom).

Complexity. To compare the complexity of a flat GMN with H-GMN, consider an input MLP with (L, d) depth/width processed by flat GMN of (K, w) depth/width vs a 2-level H-GMN with (k_1, w_1, k_2, w_2) depth/width per layer and a pooling operation. Then, G will have $(|\mathcal{V}|, |\mathcal{E}|) = (Ld, Ld^2)$ vertices and edges. First, the flat GMN scales linearly with the number of edges, and therefore will have $O(Ld^2Kw)$ memory and $O(Ld^2Kw^2)$ time complexity. In contrast, the coarsened graph has L vertices and edges and the total memory complexity is $O(Ld^2(k_1+1)w_1+Lk_2w_2)$ and the time complexity is $O(Ld^2(k_1+1)w_1^2+Lk_2w_2^2)$. The quadratic dependence on d multiplies only k_1, whereas the second term is independent of d. Since large networks are usually wide, keeping k_1 small leads to significant savings compared to a flat K-layer GMN. Note also that communication across remote nodes is costly for the flat GMN, which will require $K \geq L$, contrary to H-GMN, where communication can happen at the second level and the number of layers can be reduced depending on the hierarchy; in the worst case, $k_2 \geq L$, which incurs a much smaller cost (empirical time and memory measurements in section A.1).

7 Experiments

Datasets. We evaluate H-GMN on model zoos of ResNet18 (He et al., 2016) and ViT (Dosovitskiy et al., 2021) models. Although both architectures appear in prior zoos (Schürholt et al., 2024; Falk et al., 2025a), these vary only the random seed and are not fully publicly available, so we construct our own by also varying the learning rate and weight decay. ResNet18 models are trained on CIFAR10 and ViT models on CIFAR100 (Krizhevsky, 2009). Construction details are in Section A.3, and graph statistics are in Table 1. All datasets will be publicly released.

Table 1: Graph size in different hierarchy levels. Number of checkpoints: ResNet18 8,000; MNIST ViT 15,689; CIFAR100 ViT 3,640.

	ResNet18 (∼11M)			MNIST ViT (∼20k)			CIFAR100 ViT (∼21M)		
Graph Type	$\lvert V\rvert$	$\lvert E\rvert$	Diameter	$\lvert V\rvert$	$\lvert E\rvert$	Diameter	$\lvert V\rvert$	$\lvert E\rvert$	Diameter
Parameter Graph	$10,125$	$1,405,184$	18	619	$16,896$	16	$46,951$	$19,817,472$	27
Head Graph	–	–	–	23	28	16	208	363	27
Layer Graph	43	50	18	19	20	16	88	123	27
ResBlock Graph	10	9	9	–	–	–	–	–	–

Lipschitz constant upper-bound proxy prediction. To explicitly evaluate the importance of hierarchy as an inductive bias, we designed an experiment around a task that fundamentally demands a hierarchical understanding. Specifically, we train a metanetwork to predict a known ground-truth function: a spectral-norm proxy of the network's Lipschitz constant upper bound, computed compositionally from layer- and block-level quantities eq. (17). We compare, across different training set sizes, the performance of plain GMN with H-GMN operating on different levels of hierarchy, keeping all models under the same parameter budget, obtained by using a total of 3 GNN layers in each case. As shown in fig. 2, 3-lvl-H-GMN operating on 3 hierarchy levels (parameter, layer, and ResNet block), **achieves comparable performance with the flat GMN using only one-fifth of the training data, with $2\times$ faster inference time**. 2-lvl-H-GMN (parameter, layer) performs consistently better than GMN with both models demonstrating improved performance as the training set size increases. **H-GMNs also converge faster** - see depictions next to each point (earliest epoch after which performance gains remain below a tolerance ϵ).

Generalisation prediction. A foundational task in WSL, generalisation prediction seeks to estimate a model's test accuracy directly from its parameters. Beyond the inductive bias of H-GMN , this experiment clearly depicts the improved scalability of our method, as we evaluate on both ResNet18 ($\sim 11M$) and ViT ($\sim 21M$) models. **ResNet18**: H-GMN matches GMN when the full training set is available, and both clearly outperform StatNN and SANE. The advantage of H-GMN emerges as training data becomes scarce: the τ gap over GMN grows from 0.006 at 50% of the training checkpoints to 0.106 at 10%. Notably, even when flat GMN suffices, collapsing the parameter graph

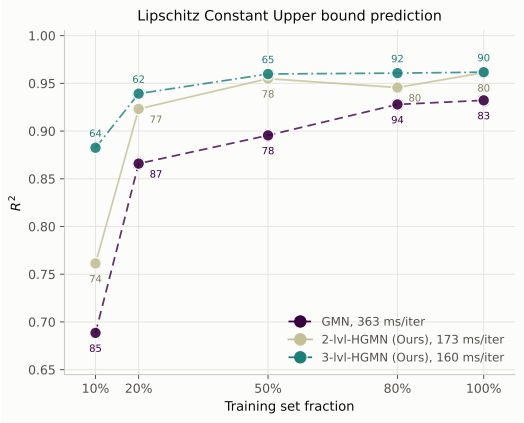


Figure 2: Coefficient of determination ($R^2 \uparrow$) on the test split for different training set sizes.

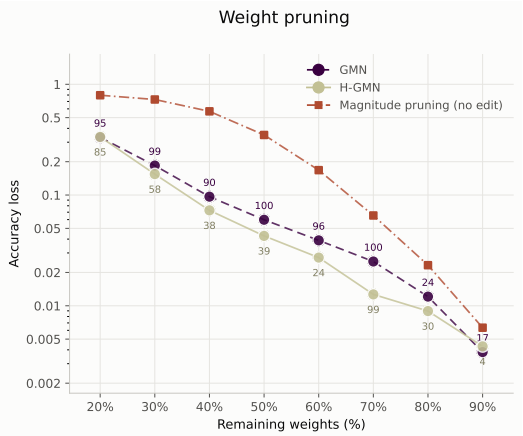


Figure 3: Average test accuracy drop (measured as $\frac{\text{acc}_{\boldsymbol{\theta}} - \text{acc}_{\tilde{\boldsymbol{\theta}}}}{\text{acc}_{\boldsymbol{\theta}}}$) across pruning ratios.

into an orders of magnitude smaller graph (see Table 1) *incurs no performance degradation*, contrary to what one might expect. This enables more memory- and time-efficient processing, allowing the method to scale to much larger graphs. The poor performance of SANE is attributed mainly to the $\sim$ 5 times smaller training dataset than the one used by Schürholt et al. (2024). **ViT: Our method achieves the best τ and R^2 on both zoos, and its margin grows with model size.** On MNIST ViT, introduced with Transformer-NFN (Tran et al., 2025), 3-lvl-H-GMN improves τ over that method by 0.013. Employing Transformer-NFN on the $\sim$ 21M-parameter CIFAR100 ViT becomes intractable. On this dataset **3-lvl-H-GMN significantly outperforms all other methods by a large margin**. Crucially, due to the size of the CIFAR100 ViT, increasing the number of GNN layers was prohibitive in terms of memory and runtime. In fact, a model with more than one GNN layer did not fit in a single 64 GB NVIDIA H100 GPU, even with automatic mixed precision and minimal batch size and hidden dimension. On the coarsened graphs, by contrast, using deeper and wider GNNs, or even Graph Transformers, introduces negligible computational overhead. Using a single GNN layer at the fine-grained resolution and reserving more expressive, but costlier, layers for the coarser ones is precisely what makes our method more scalable than existing symmetry-aware alternatives.

Robustness prediction. Beyond accuracy, a meaningful task for metanetworks is predicting robustness. We compute mean Corruption Error (mCE), following Hendrycks & Dietterich (2019), and treat mCE as the ground-truth labels to train our metanetwork. Implementation details can be found in Section A.4.3. As shown in Table 3, 3-lvl-H-GMN outperforms GMN on ResNet18 in the low-data regime and converges consistently faster than any other model, while on the CIFAR100 ViT zoo 3-lvl-H-GMN clearly outperforms all baselines. Unlike generalisation prediction, however, StatNN is highly competitive on ResNet18 when training data is scarce, achieving the best scores at 10% of the training set, suggesting that layer-level statistics capture much of the signal relevant to robustness. H-GMN overtakes it only on the full training set, where it achieves the best performance.

Table 2: Generalisation prediction. We report Kendall's rank correlation ($\tau \uparrow$), coefficient of determination ($R^2 \uparrow$), and convergence epoch. "–": architecturally inapplicable baseline.

Method	ResNet18-0.1			ResNet18-0.5			ResNet18			MNIST ViT			CIFAR100 ViT		
	τ	R^2	Epoch	τ	R^2	Epoch	τ	R^2	Epoch	τ	R^2	Epoch	τ	R^2	Epoch
StatNN	0.677	0.684	92	0.824	0.884	78	0.837	0.890	82	0.879	0.989	181	0.877	0.811	94
Transformer-NFN	–	–	–	–	–	–	–	–	–	0.906	0.987	**102**	–	–	–
GMN	0.744	0.840	93	0.886	**0.943**	94	0.900	**0.953**	97	0.867	0.995	120	0.804	0.673	**84**
SANE	0.584	0.477	-	0.614	0.566	–	0.633	0.604	–	–	–	–	–	–	–
2-lvl-H-GMN(Ours)	0.846	**0.927**	80	**0.892**	0.941	74	**0.902**	0.951	82	–	–	–	–	–	–
3-lvl-H-GMN(Ours)	**0.850**	0.906	**61**	0.881	0.927	**63**	0.893	0.945	**50**	**0.919**	**0.998**	178	**0.927**	**0.979**	86

Table 3: mCE prediction. We report Kendall's $\tau(\uparrow)$, $R^2(\uparrow)$, and convergence epoch

Method	ResNet18-0.1			ResNet18-0.5			ResNet18			CIFAR100 ViT		
	τ	R^2	Epoch	τ	R^2	Epoch	τ	R^2	Epoch	τ	R^2	Epoch
StatNN	**0.847**	**0.913**	77	**0.864**	0.927	92	0.864	0.934	82	0.899	0.959	86
GMN	0.808	0.898	87	0.858	**0.930**	90	0.867	0.934	87	0.873	0.918	**94**
SANE	0.586	0.487	–	0.614	0.583	–	0.632	0.617	–	–	–	–
2-lvl-H-GMN(Ours)	0.798	0.867	83	0.861	0.929	81	0.870	**0.942**	82	–	–	–
3-lvl-H-GMN(Ours)	0.831	0.903	**72**	0.860	0.922	**60**	**0.868**	0.938	**52**	**0.932**	**0.985**	96

Backdoor model detection. To mitigate backdoor vulnerabilities, recent literature increasingly analyses the footprint these attacks leave on NN parameters (Fields et al., 2021; Peng et al., 2025). We evaluate whether metanetworks can detect backdoored models. We train 200 ResNet18 models on CIFAR10 for each of three attacks, BadNets (Gu et al., 2017), Blended (Chen et al., 2017), and WaNet (Nguyen & Tran, 2021), and consider both a binary task (per attack) and a multi-class task. We train by subsampling the majority "clean" class and report AUROC, accuracy, F1, and recall, which penalises poisoned models misclassified as clean. 2-lvl-H-GMN (GRIT) instantiates $\mathfrak{F}_2$ as a GRIT (Ma et al., 2023) rather than a GMN as in 2-lvl-H-GMN . Full details in Section A.3.3. As shown in Table 4, **2-lvl-H-GMN (GRIT) is the best-performing method across all settings**, almost perfectly distinguishing poisoned from clean models in every case (confusion matrices in Figure 9). StatNN performs strongly in some settings, even outperforming GMN, presumably because its simple statistical features are less prone to overfitting when training data is scarce. The advantage of H-GMN over GMN suggests that detecting the traces left by backdoor attacks requires inductive biases that GMN does not easily capture, such as hierarchical or longer-range interactions.

Table 4: Backdoor model detection performance (averaged over 3 seeds) on ResNet18.

	BadNets				Blended				WaNet				Multiclass			
Method	AUROC	Acc	F1	Recall	AUROC	Acc	F1	Recall	AUROC	Acc	F1	Recall	AUROC	Acc	F1	Recall
StatNN	0.986	0.971	0.830	0.775	0.999	0.991	0.951	0.978	0.997	0.981	0.896	0.894	0.994	0.959	0.923	0.899
GMN	0.977	0.954	0.767	0.832	0.454	0.092	0.170	1.000	0.553	0.503	0.180	0.566	0.959	0.891	0.791	0.783
2-lvl-H-GMN (Ours)	0.896	0.913	0.598	0.664	**1.000**	0.995	0.974	0.970	0.970	0.959	0.775	0.753	0.999	0.983	0.950	0.948
2-lvl-H-GMN (GRIT) (Ours)	**1.000**	**0.999**	**0.996**	**0.992**	0.999	**0.999**	**0.981**	**1.000**	**1.000**	**0.999**	**0.992**	**0.995**	**0.999**	**0.984**	**0.956**	**0.955**

Pruning. Pruning is a leading approach to compressing pre-trained networks (Yu et al., 2018; Wang et al., 2019). We consider unstructured pruning of individual weights (Dong et al., 2017). Since mask selection is non-differentiable, we fix the mask with a magnitude (ℓ_1) heuristic and train H-GMN on the ResNet18 dataset to edit the unpruned parameters of $f_{C,\boldsymbol{\theta}}$ to compensate for the pruned ones, minimizing a statistical distance between the outputs of the original and the pruned, edited network on held-out images. Full details, including the training objective, are provided in Section A.6. As shown in Figure 3, H-GMN converges significantly faster than GMN while leading to smaller performance drops, particularly at intermediate pruning ratios, where the metanetwork can meaningfully compensate for the pruned weights. The exceptions are the extreme ratios, where there is little room for the metanetwork to intervene, and the two methods perform nearly identically. Notably, these gains in both performance and convergence stem solely from a lightweight module operating on the coarsened graph. Since the coarsened graph is orders of magnitude smaller than the Parameter Graph, these layers add negligible computational overhead (see section A.6).

8 Conclusion and Limitations

In this work, we propose H-GMN , a graph metanetwork framework that processes NNs at multiple resolution levels, mirroring their hierarchical structure. Our experiments on ResNet18 and ViT models show that hierarchy is a valuable inductive bias and that operating on coarsened graphs substantially improves scalability without performance compromises. Regarding limitations, H-GMNs scale efficiently well beyond the models typically studied in the GMN literature, but there is still room for improvement concerning billion-parameter models. Additionally, symmetries beyond permutations have not been tested, although theoretically applicable. Overall, with our work we aspire to draw attention to scalable, structure-aware weight space learning and to broaden the applicability of graph metanetworks to larger models.

AI USE STATEMENT

In this work, we used generative AI tools to create or modify scientific figures or images, help with implementations and create or edit software code. Additionally, to brainstorm certain methodological or experimental elements and edit specific phrasings to improve readability. We have reviewed all AI-assisted work. For example, all LLM-assisted code, including the implementation of our methods, was reviewed line by line, tested with unit tests and validated against reference implementations. We have verified the correctness of scientific figures, and reviewed and, where necessary, revised all LLM-edited text. We have not used generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, assist with translation, clean and reformat datasets, support qualitative and thematic data analysis, or interpret results. We take responsibility for the final content of this work, including text, claims or artefacts produced with the aid of generative AI.

ETHICS STATEMENT

This work uses no human subjects or private data; We trained all metanetworks on publicly available benchmarks or on datasets that we created and will be released to the public. In our experimental evaluation, we use metanetworks for automated model evaluation and testing; these results require human oversight and should not be taken as a guarantee for model behaviour. Regarding backdoor detection, our results support weight-based auditing of third-party open-source models. However, the following should be stated: First, the backdoored models in our study were created solely for evaluation using standard poisoning protocols. Second, a published detector may also guide adaptive attacks designed to evade it, and therefore should be used with care. Third, detection results should not be taken as a guarantee that a model is safe. We do not foresee other direct negative societal impacts beyond those general to weight space learning methods.

REPRODUCIBILITY STATEMENT

The code for H-GMN, the baselines, and all training and evaluation scripts will be open-sourced. The methodology and model instantiations are described in detail in sections 4 and 6, while graph construction, architectural details and hyperparameters are given in sections A.2 and A.4. The datasets that we created are described in Appendix A.3.

REFERENCES

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

Bruno Andreis, Soro Bedionita, Philip Torr, and Sung J Hwang. Set-based neural network encoding without weight tying. *Advances in Neural Information Processing Systems*, 37:76500–76528, 2024.

Aron Asefaw, Konstantinos Tzevelekakis, Damian Falk, Léo Meynent, and Damian Borth. Weightclip: Aligning datasets and models for weight space learning. *arXiv preprint arXiv:2607.03551*, 2026.

Francesco Ballerini, Pierluigi Zama Ramirez, Luigi Di Stefano, and Samuele Salti. Weight space representation learning on diverse nerf architectures. In *International Conference on Learning Representations*, volume 2026, pp. 122085–122108, 2026.

Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In *International conference on machine learning*, pp. 874–883. PMLR, 2020.

Yadi Cao, Menglei Chai, Minchen Li, and Chenfanfu Jiang. Efficient learning of mesh-based physical simulation with bi-stride multi-scale graph neural network. In *International conference on machine learning*, pp. 3541–3558. PMLR, 2023.

Wei Chen, Zhen Huang, Liang Xie, Binbin Lin, Houqiang Li, Le Lu, Xinmei Tian, Deng Cai, Yonggang Zhang, Wenxiao Wan, Xu Shen, and Jieping Ye. From yes-men to truth-tellers: addressing sycophancy in large language models with pinpoint tuning. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 257–266, 2019.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in neural information processing systems*, 33:13260–13271, 2020.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

Francesco Di Giovanni, Lorenzo Giusti, Federico Barbero, Giulia Luise, Pietro Lio, and Michael M Bronstein. On over-squashing in message passing neural networks: The impact of width, depth, and topology. In *International conference on machine learning*, pp. 7865–7885. PMLR, 2023.

Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *Advances in neural information processing systems*, 30, 2017.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

Gabriel Eilertsen, Daniel Jönsson, Timo Ropinski, Jonas Unger, and Anders Ynnerman. Classifying the classifier: dissecting the weight space of neural networks. In *European Conference on Artificial Intelligence (ECAI 2020)*, volume 325, pp. 1119–1126. IOS PRESS, 2020.

Ziya Erkoç, Fangchang Ma, Qi Shan, Matthias Nießner, and Angela Dai. Hyperdiffusion: Generating implicit neural fields with weight-space diffusion. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 14254–14264, 2023. doi: 10.1109/ICCV51070.2023.01315.

Damian Falk, Léo Meynent, Florence Pfammatter, Konstantin Schürholt, and Damian Borth. A model zoo of vision transformers. *arXiv preprint arXiv:2504.10231*, 2025a.

Damian Falk, Konstantin Schürholt, Konstantinos Tzevelekakis, Léo Meynent, and Damian Borth. Learning model representations using publicly available model hubs. *arXiv preprint arXiv:2510.02096*, 2025b.

Matthias Fey, Jan-Gin Yuen, and Frank Weichert. Hierarchical inter-message passing for learning on molecular graphs. In *ICML Graph Representation Learning and Beyond (GRL+) Workhop*, 2020.

Greg Fields, Mohammad Samragh, Mojan Javaheripi, Farinaz Koushanfar, and Tara Javidi. Trojan signatures in dnn weights. In *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pp. 12–20. IEEE, 2021.

Shahaf E. Finder, Ron Shapira Weber, Moshe Eliasof, Oren Freifeld, and Eran Treister. Improving the effective receptive field of message-passing neural networks. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=QJLGj57MfZ.

Fabrizio Frasca, Guy Bar-Shalom, Yftah Ziser, and Haggai Maron. Neural message-passing on attention graphs for hallucination detection. In *International Conference on Learning Representations*, volume 2026, pp. 449–475, 2026.

Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pp. 2083–2092. PMLR, 2019.

Charles Godfrey, Davis Brown, Tegan Emerson, and Henry Kvinge. On the symmetries of deep learning models and their internal representations. *Advances in Neural Information Processing Systems*, 35:11893–11905, 2022.

Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J Cree. Regularisation of neural networks by enforcing lipschitz continuity. *Machine Learning*, 110(2):393–416, 2021.

Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

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

Dan Hendrycks and Thomas G. Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Vincent Herrmann, Francesco Faccio, and Jürgen Schmidhuber. Learning useful representations of recurrent neural network weight matrices. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235, pp. 18205–18227. PMLR / OpenReview.net, 2024.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Eliahu Horwitz, Bar Cavia, Jonathan Kahana, and Yedid Hoshen. Learning on model weights using tree experts. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 20468–20478. IEEE, 2025.

Robert Huben, Hoagy Cunningham, Logan Smith, Aidan Ewart, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. In *International Conference on Learning Representations*, volume 2024, pp. 7827–7845, 2024.

Jonathan Kahana, Eliahu Horwitz, Imri Shuval, and Yedid Hoshen. Deep linear probe generators for weight space learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=XoYdD3m0mv.

Ioannis Kalogeropoulos, Giorgos Bouritsas, and Yannis Panagakis. Scale equivariant graph metanetworks. *Advances in Neural Information Processing Systems*, 37:106800–106840, 2024.

Ioannis Kalogeropoulos, Giorgos Bouritsas, and Yannis Panagakis. Metanetworks as regulatory operators: Learning to edit for requirement compliance. *arXiv preprint arXiv:2512.15469*, 2025.

Adam Karvonen and Samuel Marks. Robustly improving llm fairness in realistic settings via interpretability. *arXiv preprint arXiv:2506.10922*, 2025.

Dongkwan Kim and Alice Oh. Translating subgraphs to nodes makes simple gnns strong and efficient for subgraph representation learning. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24, 2024.

Boris Knyazev, Abhinav Moudgil, Guillaume Lajoie, Eugene Belilovsky, and Simon Lacoste-Julien. Accelerating training with neuron interaction and nowcasting networks. In *International Conference on Learning Representations*, volume 2025, pp. 29569–29588, 2025.

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, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. URL `https://api.semanticscholar.org/CorpusID:18268744`.

Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International conference on machine learning*, pp. 3734–3743. pmlr, 2019.

Yiming Li, Mengxi Ya, Yang Bai, Yong Jiang, and Shu-Tao Xia. Backdoorbox: A python toolbox for backdoor learning. *arXiv preprint arXiv:2302.01762*, 2023.

Renjie Liao, Marc Brockschmidt, Daniel Tarlow, Alexander L Gaunt, Raquel Urtasun, and Richard Zemel. Graph partition neural networks for semi-supervised classification. *arXiv preprint arXiv:1803.06272*, 2018.

Derek Lim, Haggai Maron, Marc T. Law, Jonathan Lorraine, and James Lucas. Graph metanetworks for processing diverse neural architectures. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=ijK5hyxs0n`.

J. Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991. doi: 10.1109/18.61115.

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

Liheng Ma, Chen Lin, Derek Lim, Adriana Romero-Soriano, Puneet K Dokania, Mark Coates, Philip Torr, and Ser-Nam Lim. Graph inductive biases in transformers without message passing. In *International Conference on Machine Learning*, pp. 23321–23337. PMLR, 2023.

Charles H Martin, Tongsu Peng, and Michael W Mahoney. Predicting trends in the quality of state-of-the-art neural networks without access to training or testing data. *Nature Communications*, 12 (1):4122, 2021.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

Aviv Navon, Aviv Shamsian, Idan Achituve, Ethan Fetaya, Gal Chechik, and Haggai Maron. Equivariant architectures for learning in deep weight spaces. In *International Conference on Machine Learning*, pp. 25790–25816. PMLR, 2023.

Tuan Anh Nguyen and Anh Tuan Tran. Wanet - imperceptible warping-based backdoor attack. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

Koyena Pal, David Bau, and Renée J. Miller. Model lakes. In *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, pp. 985–995. OpenProceedings.org, 2025. doi: 10.48786/EDBT.2025.81.

Jie Peng, Hongwei Yang, Jing Zhao, Hengji Dong, Hui He, Weizhe Zhang, and Haoyu He. Circumventing backdoor space via weight symmetry. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267. PMLR / OpenReview.net, 2025.

Theo Putterman, Derek Lim, Yoav Gelberg, Michael M Bronstein, Stefanie Jegelka, and Haggai Maron. Gl equivariant metanetworks for learning on low rank weight spaces. In *The Fourth Learning on Graphs Conference*, 2025.

Chendi Qian, Andrei Manolache, Christopher Morris, and Mathias Niepert. Probabilistic graph rewiring via virtual nodes. *Advances in Neural Information Processing Systems*, 37:28359–28392, 2024.

Ladislav Rampášek and Guy Wolf. Hierarchical graph neural nets can capture long-range interactions. In *2021 IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6. IEEE, 2021.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Konstantin Schürholt, Dimche Kostadinov, and Damian Borth. Self-supervised representation learning on neural network weights for model characteristic prediction. *Advances in Neural Information Processing Systems*, 34:16481–16493, 2021.

Konstantin Schürholt, Diyar Taskiran, Boris Knyazev, Xavier Giró-i Nieto, and Damian Borth. Model zoos: A dataset of diverse populations of neural network models. *Advances in Neural Information Processing Systems*, 35:38134–38148, 2022.

Konstantin Schürholt, Michael W. Mahoney, and Damian Borth. Towards scalable and versatile weight space learning. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235, pp. 43947–43966. PMLR / OpenReview.net, 2024.

Aviv Shamsian, Aviv Navon, David W. Zhang, Yan Zhang, Ethan Fetaya, Gal Chechik, and Haggai Maron. Improved generalization of weight space networks via augmentations. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235, pp. 44378–44393. PMLR / OpenReview.net, 2024.

Bedionita Soro, Aoxuan Zhang, Bruno Andreis, Jaehyeong Jo, Song Chong, and Sung Ju Hwang. Ls-merge: Merging language models in latent space. In *International Conference on Learning Representations*, volume 2026, pp. 147812–147833, 2026.

Andreas Peter Steiner, Alexander Kolesnikov, Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. How to train your vit? data, augmentation, and regularization in vision transformers. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL `https://openreview.net/forum?id=4nPswr1KcP`.

Yonatan Sverdlov, Ido Springer, and Nadav Dym. Revisiting multi-permutation equivariance through the lens of irreducible representations. In *International Conference on Learning Representations*, volume 2025, pp. 19220–19242, 2025.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

Viet-Hoang Tran, Thieu Vo, Tho Tran-Huu, An T Nguyen, and Tan Nguyen. Monomial matrix group equivariant neural functional networks. *Advances in Neural Information Processing Systems*, 37: 48628–48665, 2024.

Viet-Hoang Tran, Thieu Vo, An Nguyen, Tho-Huu Tran, Minh-Khoi Nguyen-Nhat, Thanh Tran, Duy-Tung Pham, and Tan Nguyen. Equivariant neural functional networks for transformers. In *International Conference on Learning Representations*, volume 2025, pp. 16270–16341, 2025.

Thomas Unterthiner, Daniel Keysers, Sylvain Gelly, Olivier Bousquet, and Ilya Tolstikhin. Predicting neural network accuracy from weights. *arXiv preprint arXiv:2002.11448*, 2020.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart M. Shieber. Investigating gender bias in language models using causal mediation analysis. In *Advances in Neural Information Processing Systems*, 2020.

Thieu Vo, Hoang V. Tran, Tho Tran Huu, An Nguyen The, Thanh Tran, Minh-Khoi Nguyen-Nhat, Duy-Tung Pham, and Tan Minh Nguyen. Equivariant polynomial functional networks. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 61689–61744. PMLR, 13–19 Jul 2025.

Carlos Vonessen, Florian Grötschla, and Roger Wattenhofer. Next level message-passing with hierarchical support graphs. *arXiv preprint arXiv:2406.15852*, 2024.

Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong Zhang. Eigendamage: Structured pruning in the kronecker-factored eigenbasis. In *International Conference on Machine Learning*, pp. 6566–6575. PMLR, 2019.

Limei Wang, Haoran Liu, Yi Liu, Jerry Kurtin, and Shuiwang Ji. Learning hierarchical protein representations via complete 3d graph networks. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

Renhao Wang, Marjan Albooyeh, and Siamak Ravanbakhsh. Equivariant networks for hierarchical structures. *Advances in Neural Information Processing Systems*, 33:13806–13817, 2020.

Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4820–4828, 2016.

Yi Yang, Hanyu Duan, Ahmed Abbasi, John P Lalor, and Kar Yan Tam. Bias a-head? analyzing bias in transformer-based language model attention heads. In *Proceedings of the 5th Workshop on Trustworthy NLP (TrustNLP 2025)*, pp. 276–290, 2025.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.

Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 9194–9203, 2018.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

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, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267. PMLR / OpenReview.net, 2025.

Zhiqiang Zhong, Cheng-Te Li, and Jun Pang. Hierarchical message-passing graph neural networks: Z. zhong et al. *Data Mining and Knowledge Discovery*, 37(1):381–408, 2023.

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, 2023a.

Allan Zhou, Kaien Yang, Yiding Jiang, Kaylee Burns, Winnie Xu, Samuel Sokota, J Zico Kolter, and Chelsea Finn. Neural functional transformers. *Advances in neural information processing systems*, 36:77485–77502, 2023b.

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

A Appendix

A.1 Time and Memory Scaling with Metanetwork Depth.

We empirically verify the complexity analysis of section 6. Using the notation of that section, for a flat GMN, both memory $O(Ld^2Kw)$ and time $O(Ld^2Kw^2)$ complexity scale linearly with the number of GMN's layers K. In contrast, for an H-GMN, which has $O(Ld^2(k_1+1)w_1 + Lk_2w_2)$ memory and the $O(Ld^2(k_1+1)w_1^2 + Lk_2w_2^2)$ time complexity, setting $k_1 = 1$, renders the d-dependent term fixed, and additional layers only contribute to the second term of each complexity, which should be negligible. Figures 4 and 5 report inference time, training step time and peak training memory against the total number of GNN layers.

Setup. All measurements are taken on a single NVIDIA H100 GPU (64 GB), one configuration at a time, each in a separate process. For every configuration we discard 10 warm-up steps and report the median over the next 50 steps; timings are GPU-synchronised CUDA-event measurements of the metanetwork alone (data loading and host-to-device transfer are excluded), and memory is the peak allocated GPU memory during the timed steps. A training step comprises the forward pass, the loss, the backward pass, and an optimizer update. Inference time is reported per network (B=1); training steps use the per-GPU training batch size of each zoo (B=4 for ResNet18, B=1 for ViT). ResNet18 experiments run in fp32 and ViT experiments in bf16 mixed precision, matching the training setup of section 7. Hidden sizes are configured to result in a same number of parameters model, similar to the Lipschitz constant experiment; ResNet18: GMN 64, 2-lvl-H-GMN 58, 3-lvl-H-GMN 51, each ~ 0.9M parameters; ViT: GMN 52, H-GMN 35, ~ 0.66M parameters. We then vary depth at a single level while keeping all widths fixed: the number of layers of the flat GMN, $K \in \{1, 2, 3, 4, 6, 8\}$, and the number of coarse-level layers of H-GMN , $k_2 \in \{1, 2, 4, 8, 16\}$ (for 3-lvl-H-GMN , the number of layers at the coarsest level), with a single Parameter Graph layer (k_1=1). Consequently, only the default-depth configurations are parameter-matched; parameters grow with depth. Once a configuration runs out of memory, deeper configurations of the same model are not run. We report memory only for training: during inference, the activations of each layer are released once the next layer has consumed them, so peak inference memory is nearly independent of depth (e.g. 3.5 GiB for the ResNet18 GMN from two to eight layers), whereas training must keep every layer's activations for the backward pass, which is what makes deep flat GMNs prohibitive.

ResNet18 zoo. As shown in fig. 4, GMN grows linearly with depth, as predicted, and runs out of memory beyond three layers. Both 2-lvl- and 3-lvl-H-GMN variants (whose minimum total number of layers is 2 and 3, respectively) remain nearly constant up to 17 layers. This confirms that their cost is dominated by the single Parameter Graph layer.

CIFAR100 ViT zoo. Here d is much larger, thus memory complexity $O(Ld^2Kw)$ makes a single GMN layer already expensive, and a second one exceeds GPU memory. As depicted in fig. 5, H-GMN again stays flat in all three metrics, since the coarse-level term is independent of d.

A.2 Graph Construction

The graph construction procedure is a map $\mathfrak{G} : \mathcal{F} \to \mathcal{G}$ that receives a network $f_{C,\boldsymbol{\theta}} \in \mathcal{F}$ and yields a graph $\mathfrak{G}(f_{C,\boldsymbol{\theta}}) = G = (\mathcal{V}, \mathcal{E}, \mathbf{x}_V, \mathbf{x}_E)$. Within the Graph Metanetworks framework, $\mathfrak{G}$ is designed so that the resulting nodes and edges inherit the same symmetries as the network's own parameters.

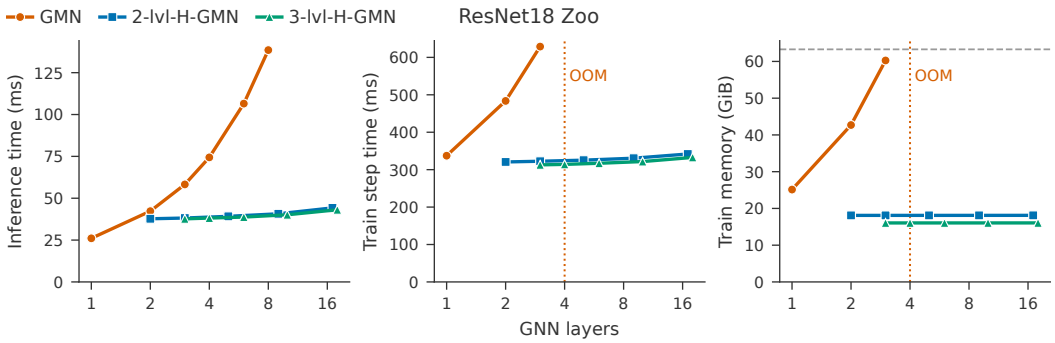


Figure 4: **ResNet18** Left: inference time per network (B=1). Middle and right: time and peak GPU memory of one training step (B=4). The x-axis counts all GNN layers. The dotted line marks where GMN runs out of memory, the dashed line is the GPU capacity (64 GB).

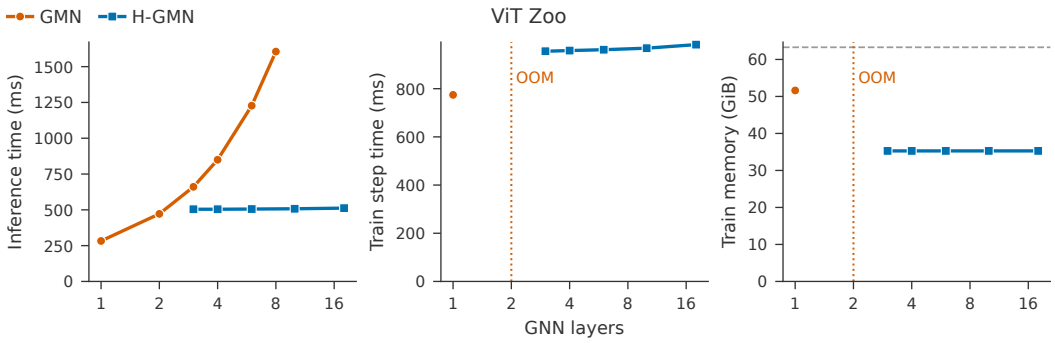


Figure 5: **CIFAR100 ViT** Left: inference time per network. Middle and right: time and peak GPU memory of one training step. B=1 throughout. The dotted line marks where GMN runs out of memory, the dashed line is the GPU capacity (64 GB).

Although many of the modules we are mapping to the Parameter Graph have been addressed by Lim et al. (2024), our proposed method deviates in most cases. To construct the parameter $\mathcal{G}$ of an input model $f_{C,\boldsymbol{\theta}}$, we typically traverse the computational graph C and progressively add new sets of nodes and edges with connectivity with the previous layers, dictated from each layer we encode. An abstract algorithm of this process is described in Algorithm 1. In the following sub-sections we analyze how ENCODE parses.

Algorithm 1 $\mathfrak{G}$: parameter-graph construction via layer-by-layer traversal of C

Require: NN $f_{C,\boldsymbol{\theta}}$ with computational graph C, topologically ordered into layers $\ell_1, \ldots, \ell_T$
Ensure: Parameter graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{x}_V, \mathbf{x}_E)$

$G \leftarrow (\emptyset, \emptyset)$
active[input] $\leftarrow$ one node per input channel of C
for $t = 1, \ldots, T$ **do** ▷ visit C in topological order
  $U \leftarrow$ active[$\cdot$] for the tensor(s) feeding ℓ_t
  $G, V_{\ell_t} \leftarrow \textsc{Encode}(\ell_t, U, G)$
  active[name(ℓ_t)] $\leftarrow V_{\ell_t}$
end for
return G

Channel-mixing layers. Let ℓ be a linear or convolutional layer with weight $\mathbf{W}$ and bias $\mathbf{b}$ (taken to be $\mathbf{0}$ when ℓ has none), and let $\mathcal{V}$, with $|\mathcal{V}| = d_{\text{in}}$, be its input vertices. Encode$(\ell, \mathcal{V}, G)$ adds one

vertex per output channel and connects every input channel to every output channel:

$$\Delta\mathcal{V} = \mathcal{V}_o,\ |\mathcal{V}_o| = d_{\text{out}}, \qquad \mathbf{x}_V(j) = [\mathbf{b}(j)], \tag{5}$$

$$\Delta\mathcal{E} = \mathcal{V} \times \mathcal{V}_o, \qquad \mathbf{x}_E(i,j) = \mathbf{W}(j,i,:) \in \mathbb{R}^{k_\ell}, \tag{6}$$

and returns $\mathcal{V}_o$. The complete bipartite graph $\mathcal{V} \times \mathcal{V}_o$ is exactly the wiring rule already stated for channel-mixing layers in section 3. The only thing that varies across modules is the per-pair feature width k_ℓ, the number of raw weight scalars ℓ allocates to a given (input, output) channel pair:

- $k_\ell = 1$ for a `Linear` layer whose input width equals $d_{\text{in}} = |\mathcal{V}|$, which is the ordinary scalar-weight case $\mathbf{x}_E(i,j) = [\mathbf{W}(j,i)]$;
- $k_\ell = \prod$ kernel_size for `Conv1d/2d/3d`, where the edge carries the whole spatial kernel between the two channels (the sequence/spatial dimension does not enter the graph, as in the main text);

Attention's query/key/value/output projections are channel-mixing layers too, but a plain bipartite call is not enough to expose per-head structure to later pooling; they are treated separately in sections A.2 and 5.

Channel-wise layers. Let ℓ be a normalization or pooling layer with input vertices $\mathcal{V}$, $|\mathcal{V}| = d_{\text{in}}$. Unlike channel-mixing layers, ℓ does not mix channels. It adds one new vertex per input channel, so $|\mathcal{V}_o| = d_{\text{out}} = d_{\text{in}}$, and connects each input vertex *one-to-one* to its output vertex instead of bipartitely. We index both sets by channel, so vertex $i \in \mathcal{V}$ is connected to vertex $j = i \in \mathcal{V}_o$:

$$\Delta\mathcal{V} = \mathcal{V}_o, \qquad \mathbf{x}_V(j) = [\beta(j)], \tag{7}$$

$$\Delta\mathcal{E} = \{(i,j) \in \mathcal{V} \times \mathcal{V}_o : j = i\}, \qquad \mathbf{x}_E(i,j) = [\gamma(j)], \tag{8}$$

and returns $\mathcal{V}_o$. This is the channel-wise wiring rule of section 3. The values of (γ, β) depend on the module:

- **BatchNorm / InstanceNorm at inference** (running statistics tracked): the four per-channel quantities are folded into two effective affine terms,

$$\gamma(j) = \frac{\gamma^{\text{raw}}(j)}{\sqrt{\sigma^2(j) + \epsilon}}, \qquad \beta(j) = \beta^{\text{raw}}(j) - \gamma^{\text{raw}}(j)\frac{\mu(j)}{\sqrt{\sigma^2(j) + \epsilon}}, \tag{9}$$

  where $(\gamma^{\text{raw}}, \beta^{\text{raw}})$ are the module's own affine parameters and (μ, σ^2) its running mean and variance. This assumes a frozen checkpoint evaluated at inference, which is the trained-model setting that weight space learning already works in.
- **LayerNorm, GroupNorm, InstanceNorm without running statistics**: there are no running statistics to fold, so $\gamma(j) = \gamma^{\text{raw}}(j)$ and $\beta(j) = \beta^{\text{raw}}(j)$.
- **Pooling** (max/average, plain or adaptive): ℓ has no learnable parameters, so $\mathbf{x}_V(j) = \mathbf{x}_E(i,j) = [0]$. Which pooling operator was applied is recorded only in the edge type (section A.2), not in $\mathbf{x}_E$. Because standard pooling layers have no learnable parameters, their vertices and edges are identical for every model in a zoo of fixed architecture. In such setups they can be discarded without affecting representation quality.

Skip connections. A skip connection adds two things to the graph. The first is an optional *parametric* shortcut branch, e.g. a strided 1×1 convolution followed by BatchNorm, as in ResNet's downsample path. The second is the *non-parametric* summation that recombines the shortcut with the main path. The shortcut branch needs no new rule: its channel-mixing and channel-wise layers are encoded exactly as in the previous two paragraphs. By the time C's traversal reaches the sum, both operands therefore already have vertex sets in G. What remains is the sum itself.

Let ℓ be an elementwise addition that merges the main branch's vertices $\mathcal{V}$ with the residual branch's vertices $\mathcal{V}'$, where $|\mathcal{V}| = |\mathcal{V}'|$ so the two can be summed. We index both sets by channel. ℓ adds no new vertices. It connects the residual branch to the main branch with a single set of unweighted, one-to-one edges:

$$\Delta\mathcal{V} = \emptyset, \tag{10}$$

$$\Delta\mathcal{E} = \{(i,j) \in \mathcal{V}' \times \mathcal{V} : j = i\}, \qquad \mathbf{x}_E(i,j) = [1], \tag{11}$$

and returns $\mathcal{V}$. Downstream layers continue from the main branch's vertices, and the residual branch's vertices are not touched again. When our model initializes the graph, vertices and edges that carry no parameters receive learnable embeddings instead of a fixed value. One embedding is shared by all vertices, and one by all edges, of the same layer.

Multi-head self-attention. A multi-head self-attention block does not fit the plain channel-mixing rule. Restricted to permutations, its symmetry group is $S_{\boldsymbol{\theta}}^{\mathrm{MHSA}} = (S_{d_k} \times S_{d_v})^H \rtimes S_H$. This group permutes each head's query/key coordinates jointly and its value/output coordinates jointly, which a single bipartite edge set cannot express. Each head is therefore given its own pair of vertex sets.

Consider a block with H heads, $f_{C,\boldsymbol{\theta}}(\mathbf{x}) = \sum_{h=1}^{H} \mathbf{W}_o^h(\mathbf{W}_v^h\mathbf{x})\,\sigma\big((\mathbf{W}_q^h\mathbf{x})^T(\mathbf{W}_k^h\mathbf{x})\big)$, with $\mathbf{W}_q^h, \mathbf{W}_k^h \in \mathbb{R}^{d_k \times d_{\text{in}}}$, $\mathbf{W}_v^h \in \mathbb{R}^{d_v \times d_{\text{in}}}$, $\mathbf{W}_o^h \in \mathbb{R}^{d_{\text{out}} \times d_v}$, and input vertices $\mathcal{V}$, $|\mathcal{V}| = d_{\text{in}}$ (in the notation of section 5, $\mathcal{V} = \mathbf{V}_l$). For each head $h \in [H]$, Encode adds two vertex sets, $\mathcal{V}_k^h$ with $|\mathcal{V}_k^h| = d_k$ and $\mathcal{V}_v^h$ with $|\mathcal{V}_v^h| = d_v$. It also adds one output set $\mathcal{V}_o$ with $|\mathcal{V}_o| = d_{\text{out}}$, shared by all heads. These sets are connected by four edge sets per head:

$$\mathcal{E}_k^h = \mathcal{V} \times \mathcal{V}_k^h, \qquad \mathbf{x}_E(i,j) = [\mathbf{W}_q^h(j,i), \mathbf{W}_k^h(j,i)] \in \mathbb{R}^2, \tag{12}$$

$$\mathcal{E}_v^h = \mathcal{V} \times \mathcal{V}_v^h, \qquad \mathbf{x}_E(i,j) = [\mathbf{W}_v^h(j,i)], \tag{13}$$

$$\mathcal{E}_{kv}^h = \mathcal{V}_k^h \times \mathcal{V}_v^h, \qquad \mathbf{x}_E(i,j) = [1/\sqrt{d_k}], \tag{14}$$

$$\mathcal{E}_o^h = \mathcal{V}_v^h \times \mathcal{V}_o, \qquad \mathbf{x}_E(i,j) = [\mathbf{W}_o^h(j,i)], \tag{15}$$

so that $\mathcal{E}_o = \bigcup_h \mathcal{E}_o^h$ as in the main text, and

$$\Delta\mathcal{V} = \mathcal{V}_o \cup \bigcup_{h=1}^{H} (\mathcal{V}_k^h \cup \mathcal{V}_v^h), \qquad \Delta\mathcal{E} = \bigcup_{h=1}^{H} (\mathcal{E}_k^h \cup \mathcal{E}_v^h \cup \mathcal{E}_{kv}^h \cup \mathcal{E}_o^h). \tag{16}$$

Encode returns $\mathcal{V}_o$ ($= \mathbf{V}_{l+1}$) as the block's output vertices. The per-head sets $\mathcal{V}_k^h, \mathcal{V}_v^h$ remain in G but are not passed on as input to later layers. As in the main text, biases are omitted for simplicity. $\mathcal{E}_k^h$ and $\mathcal{E}_{kv}^h$ get their own edge types, which distinguish them from linear layers; $\mathcal{E}_v^h$ and $\mathcal{E}_o^h$ use the linear-layer edge type.

$\mathcal{E}_k^h$ stores the query and key weights of a channel pair together as one 2-dimensional feature. Any permutation of $\mathcal{V}_k^h$ therefore reorders queries and keys together, which is exactly the action of S_{d_k}. $\mathcal{E}_{kv}^h$ is complete bipartite with a constant, parameter-free feature. It links $\mathcal{V}_k^h$ to $\mathcal{V}_v^h$, so relabeling the heads (S_H) moves both sets of a head together.

Positional encodings. A standard MPNN is equivariant to the full symmetric group $S_{|\mathcal{V}|}$ acting on vertex indices: with no notion of vertex identity beyond adjacency and features, it treats *any* relabeling of G as a symmetry. $S_{|\mathcal{V}|}$ is far larger than the group we actually want, $S_{\boldsymbol{\theta}} = \prod_g S_{|g|}$: it contains permutations that mix nodes across groups, which are not symmetries of $f_{C,\boldsymbol{\theta}}$ and must not be treated as such. We therefore equip G, at construction time, with symmetry-breaking positional encodings $\mathbf{p}_V, \mathbf{p}_E$ that are shared within a group but differ across groups, restricting the graph's effective symmetry to exactly $S_{\boldsymbol{\theta}}$. This mirrors standard practice in graph metanetworks (Lim et al., 2024; Kofinas et al., 2024; Kalogeropoulos et al., 2024).

Further, $\mathbf{p}_V$ typically also carries each vertex's type and, where applicable, its activation function; similarly, $\mathbf{p}_E$ carries each edge's type. Both remain constant within a group, so $S_{\boldsymbol{\theta}}$ is preserved.

A.3 Model Zoo construction

Weight space learning (WSL) is driven by the large number of trained models that are now publicly available: model hubs hold millions of checkpoints, and methods that treat these parameters as data could be used to audit, compare and edit models at scale. Benchmarking WSL methods has not kept pace. Most existing evaluations use populations of small MLPs or CNNs (Unterthiner et al., 2020; Schürholt et al., 2021), whose size and simplicity are far from the models people actually deploy. A few recent efforts have released populations of modern architectures, but each has a limitation that makes it a poor benchmark. The ResNet18 zoo of Schürholt et al. (2022) contains a checkpoint per training epoch, yet datapoint models differ only in the random seed. The training hyperparameters are held fixed, so the population covers a narrow range of model behaviour, and

predicting a property from the weights becomes a question of seed noise rather than of real functional differences. A population of Vision Transformers has been proposed by (Falk et al., 2025a), but only a small fraction of their models has been released, too few to serve as a benchmark. To address both problems, we build three new model zoos: a ResNet18 zoo and a ViT zoo, both trained under deliberately varied hyperparameters, and a zoo of ResNet18 backdoored models. Each checkpoint comes with a set of labels: clean accuracy, mean corruption error (mCE), and, for ResNet18, a Lipschitz constant upper bound. All datasets will be made publicly available. We describe each zoo below.

A.3.1 RESNET18 ZOO

Sampling strategy. Because every member of the zoo shares the ResNet18 architecture, structural hyperparameters (depth, width, activation, normalisation) cannot be varied without leaving the model family. All diversity in the population therefore has to come from the training procedure. Every model uses the standard CIFAR variant of ResNet18 ($\sim$11.2M parameters), is initialised with Kaiming-uniform initialisation (He et al., 2015), and is trained on CIFAR-10 for 50 epochs with SGD (momentum 0.9), batch size 128, a one-cycle learning-rate schedule and standard augmentation (random crop with 4-pixel padding and horizontal flips). For each model we independently sample the peak learning rate $\eta \sim \log\mathcal{U}[10^{-3}, 10^{-1}]$, the weight decay $\lambda \sim \log\mathcal{U}[10^{-6}, 10^{-2}]$, and the random seed, which controls initialisation and data ordering. Sampling on a logarithmic scale spreads the models evenly across two orders of magnitude in η and four in λ. This range runs from strongly regularised to nearly unregularised training, and so produces populations that differ in accuracy, robustness and weight norm.

Dataset size. The zoo contains $1{,}600$ ResNet18 models. We save a checkpoint every 5 epochs, giving $16{,}000$ checkpoints in total. Our experiments use the checkpoints at epochs $\{10, 20, 30, 40, 50\}$, i.e. $8{,}000$ labelled samples. Models are split into train, validation and test sets at the level of the training run ($1{,}120/240/240$ models), so all checkpoints of a model fall in the same split and no training trajectory leaks across splits.

A.3.2 VIT ZOO

Vision Transformers now dominate public model repositories, and in practice they are rarely trained from scratch: most checkpoints come from fine-tuning a small number of widely shared pretrained backbones. A realistic WSL benchmark for transformers should therefore capture both sources of variation, namely *which* pretrained model a checkpoint starts from and *how* it is fine-tuned. Falk et al. (2025a) built a ViT zoo in this spirit, but only a small subset of its models has been released, which is too few to train and evaluate WSL methods. We therefore construct a new, larger ViT zoo, closely following their procedure.

Sampling strategy. As with ResNet18, the architecture is fixed: all models are ViT-S/16 ($\sim$21.7M parameters), so diversity must come from the training pipeline. Following Falk et al. (2025a), we vary it along two axes. *(i) Pretrained backbone.* Besides the two ViT-S/16 backbones of Falk et al. (2025a), pretrained from scratch with a supervised and a contrastive objective respectively, we add three public `timm` checkpoints with different pretraining data (AugReg on ImageNet-1k, ImageNet-21k, and ImageNet-21k followed by ImageNet-1k fine-tuning (Steiner et al., 2022)). This gives five backbones in total. *(ii) Fine-tuning hyperparameters.* Using the fine-tuning protocol of Falk et al. (2025a), each backbone is fully fine-tuned on CIFAR-100 (upsampled to 224×224, with random-crop and horizontal-flip augmentation) for 50 epochs with a freshly initialised linear classification head. The optimiser is AdamW (Loshchilov & Hutter, 2019) or SGD (Nesterov momentum 0.9), with a constant learning rate $\eta \in \{10^{-4}, 3 \cdot 10^{-4}, 10^{-3}, 3 \cdot 10^{-3}\}$, weight decay $\lambda \in \{0, 10^{-5}, 10^{-4}, 10^{-3}\}$ and batch size $B \in \{64, 128, 256\}$. For each backbone we sample fine-tuning runs as follows: the random seed deterministically selects a configuration (η, λ, B) from the resulting $4 \times 4 \times 3 = 48$-point grid, and every seed is fine-tuned with both optimisers. Across seeds, every backbone covers the full grid, and each (backbone, configuration, optimiser) combination is trained with one or two seeds, which control the initialisation of the new head and the data ordering.

Dataset size. The zoo contains 728 models: 160 for each `timm` backbone and 124 for each backbone of Falk et al. (2025a). We keep checkpoints at epochs $\{10, 20, 30, 40, 50\}$, giving $3{,}640$ check-

points. As for ResNet18, models are split into train, validation and test sets at the model level ($70/15/15\%$), so all checkpoints of a fine-tuning run fall in the same split.

A.3.3 BACKDOOR ATTACKS

Attacks. We consider three data-poisoning attacks with qualitatively different triggers. In all of them, a fraction ρ of the training images is stamped with a trigger and relabelled to a target class y_t, and the model is trained as usual on the poisoned set. *BadNets* (Gu et al., 2017) stamps an opaque white patch in the image corner. *Blended* (Chen et al., 2017) blends the same patch into the image with weight 0.2, which makes it less visible. *WaNet* (Nguyen & Tran, 2021) replaces the additive patch with an imperceptible, smooth warping of the whole image, Our trigger implementations follow BackdoorBox (Li et al., 2023), a widely used open-source toolbox for backdoor learning, with its default CIFAR configurations.

Construction. Every backdoored model is the *twin* of a clean model in the corresponding zoo: it shares that model's architecture, training hyperparameters, random seed and checkpoint schedule, and differs only in the poisoned training data. For each model we sample the poisoning rate $\rho \sim \mathcal{U}[0.05, 0.5]$, a target class y_t uniformly at random, and a trigger seed, which fixes the WaNet warping field. For ResNet18 we train 200 backdoored twins per attack on CIFAR-10, using a 3×3 patch for BadNets and Blended. Each backdoored model inherits its twin's train/validation/test assignment, so a clean model and its backdoored counterpart never fall in different splits and a detector cannot exploit shared hyperparameters across splits.

A.4 EXPERIMENTAL DETAILS

In this section, we provide additional experimental details regarding our experiments.

Architecture. Every H-GMN applies a single message-passing layer on the Parameter Graph ($k_1 = 1$), since this is the only stage whose cost scales with the parameter count of the input network. Vertex and edge pooling are implemented as a DeepSets aggregator (Zaheer et al., 2017) over mean, max and standard deviation statistics (Corso et al., 2020). On the coarsened graphs, the networks $\mathfrak{F}_i$, $i \geq 2$, are instantiated either as message-passing GNNs or as GRIT (Ma et al., 2023). In the GRIT case, Relative Random Walk Probabilities (RRWP) are used to enlarge the receptive field of each layer. The number of coarse-level layers is treated as a hyperparameter. Positional encodings and the learnable embeddings of parameter-free vertices and edges follow section A.2. The 2-lvl-H-GMN variant pools the Parameter Graph into the Layer Graph; for ResNet18 we additionally consider a 3-lvl-H-GMN variant that further pools into the ResBlock Graph, and for ViTs the Head Graph is used as an intermediate level.

A.4.1 LIPSCHITZ CONSTANT UPPER BOUND DISTRIBUTION.

Unlike most experiments in the literature, where the ground-truth function is completely unknown, we propose a new experiment to evaluate the importance of hierarchy as an inductive bias. For ResNet-18 models, we bound the global network using the compositional product of its layers' spectral norms (Szegedy et al., 2014). For residual blocks, following Gouk et al. (2021), we bound the block by the sum of the spectral norms of the main computational path and the skip connection. The 4D convolution kernels are reshaped to 2D matrices to compute their spectral norm, aligning with Miyato et al. (2018). Since this reshaping does not guarantee to upper-bound the operator norm of a convolution, $f^*(\theta)$ should be read as a compositional proxy rather than a certified bound; what matters for our purposes is that it is an exact, known function of θ with an explicit hierarchical structure. As a result, the ground-truth label for each input θ can be derived as:

$$f^*(\theta) = \sigma_{\max}(\tilde{\mathbf{W}}_{conv1}) \cdot \prod_{b=1}^{B} \left(s_b + \prod_{\ell=1}^{L_B} \sigma_{\max}(\tilde{\mathbf{W}}_{b,\ell}) \right) \cdot \sigma_{\max}(\mathbf{W}_{fc}) \tag{17}$$

where B is the total number of residual blocks, L_B the number of convolution layers within each block, and s_b is the Lipschitz constant of the skip connection, defined as $s_b = 1$ for parameter-less identity shortcuts, and $s_b = \sigma_{\max}(\tilde{\mathbf{W}}_b^{skip})$ for downsampling projection shortcuts. The notation

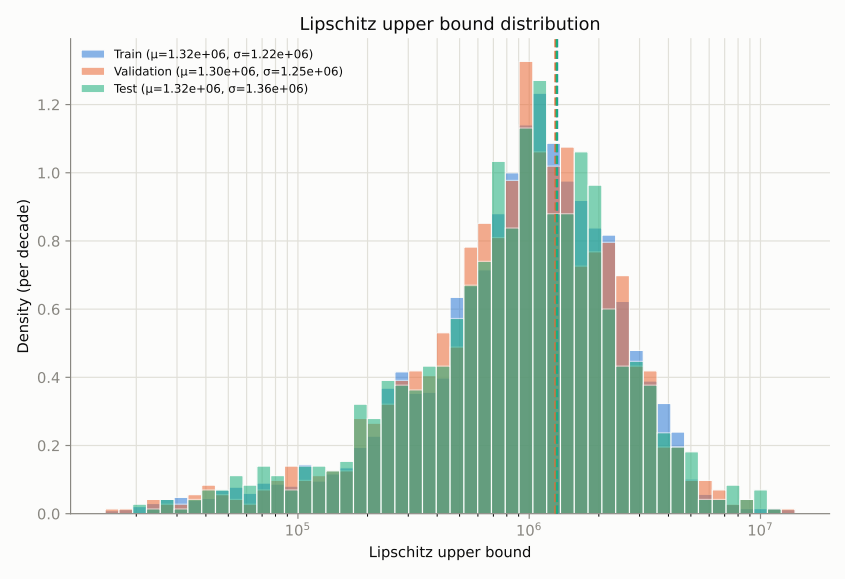


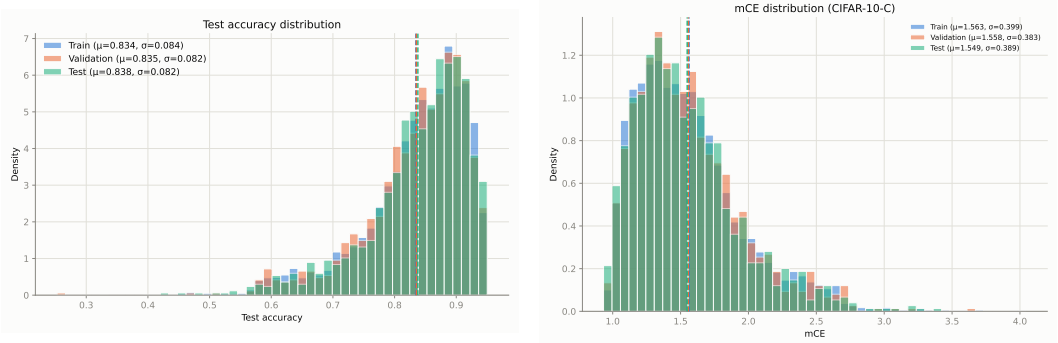
Figure 6: Distribution of the Lipschitz constant upper-bound proxy across the ResNet18 zoo.

$\tilde{\mathbf{W}}$ indicates the Conv-BatchNorm folded weights, defined as $\tilde{\mathbf{W}}_{b,\ell} = \frac{\gamma_{b,\ell}}{\sqrt{\sigma^2_{b,\ell}+\epsilon}}\mathbf{W}_{b,\ell}$. We plot the distribution of those values in fig. 6.

We plot the distribution of the Lipschitz constant upper bound in Figure 6. Evidently, our zoo consists of ResNet18 models of significantly varying Lipschitz constant upper bounds, with $\sigma = 1.22e + 06$. **Setup**: In this experiment we aim to evaluate the hierarchy of neural networks as an inductive bias. Thus we perform our experiments on a very controlled setup. We fix the number of parameters and the number of GNN layers and change only the resolution on which each GNN layer is applied. Flat GMN applies 3 GNN layers on the Parameter Graph level. 2-lvl-H-GMN uses 1 GNN layer in the fine-grained resolution, applies pooling of symmetric statistics (Corso et al., 2020) and then 2 GNN layers on the coarsened graph. Finally, 3-lvl-H-GMN spans 3 different hierarchy levels with one GNN layer in each. The second pooling need not be a permutation invariant aggregation, since, in ResNet models, Pool_1 has already collapsed all parameter symmetries. We simply concatenate the node/edge features of the elements that reside in the same residual block. Downsampling blocks contain two more layers than identity blocks (the 1×1 convolution and BatchNorm of the projection shortcut), so their concatenated features are longer. We therefore use a separate projection for each block type, mapping both to a common dimension. For datasets with heterogeneous architectures, where such mismatches are more frequent, zero-padding to a fixed number of slots, or an aggregation over the block's layers combined with positional encodings that identify each layer, would scale better.

A.4.2 Generalization prediction

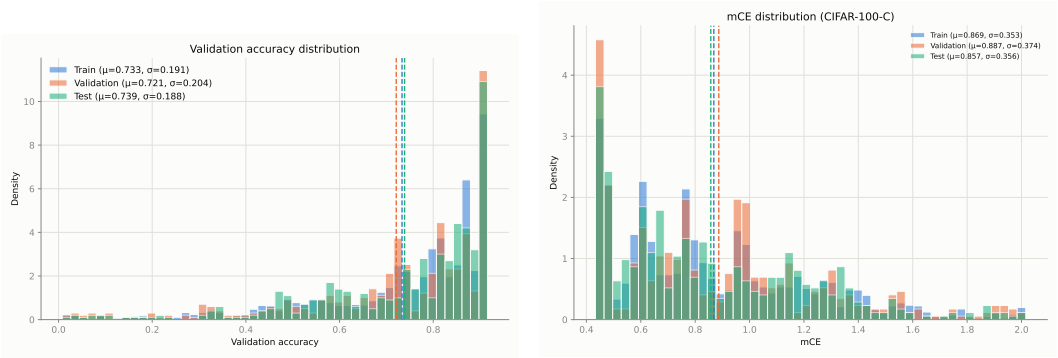
ResNet18 Zoo We plot the distribution of test accuracy in Figure 7a. Evidently, our zoo consists of ResNet18 models of significant variety. **Hyperparameters**: For H-GMN we use only one GNN layer at the Parameter Graph resolution. In the coarsened graph, typically denoted as 2-lvl-H-GMN , we experiment with both Graph Neural Networks and GRIT (Ma et al., 2023) for the instantiation of $\mathfrak{F}_2$. In both cases, we treat the number of layers as a hyperparameter and search the number of layers $k_2 \in \{1, 2, 3\}$. In the GRIT case, we also use Relative Random Walk Probabilities (RRWP) to swiftly increase the receptive field of each layer (Ma et al., 2023). Additionally, we set batch size $= 4$, hidden dimension for node/edge features in $\{32, 64\}$ for $\mathfrak{F}_1$ and in $\{64, 128\}$ for $\mathfrak{F}_2$. We also search learning rate in $\{0.001, 0.0005, 0.0001\}$ and weight decay in $\{0.0, 0.001\}$. After converging in a specific configuration, we then run the same experiment for all training size options. For the GMN baseline, we search $k_1 \in \{2, 3, 4\}$ and the hidden size in $\{32, 64, 128\}$ for the combinations that are able to fit in a GPU. The rest of the hyperparameters are treated the same way as in H-GMN



(a) Test accuracy

(b) mCE

Figure 7: Distributions of (a) test accuracy and (b) mean corruption error (mCE) across the ResNet18 zoo.



(a) Validation accuracy

(b) mCE

Figure 8: Distributions of (a) validation accuracy and (b) mean corruption error (mCE) across the CIFAR100 ViT Zoo.

. For SANE (Schürholt et al., 2024), to avoid costly re-pre-training for each training subset, we reuse the embeddings obtained from pre-training on the full training dataset and train only the downstream head on the corresponding subset. Despite this advantage, SANE underperforms all other methods.

CIFAR100 ViT Zoo In this Zoo, we use as target values for the generalisation prediction task, the accuracy on the validation split (Falk et al., 2025a). The distribution across the ViT Zoo is shown in Figure 8a. **Hyperparameters**: Again, for H-GMN we use only one GNN layer at the Parameter Graph resolution. In the coarsened graph, we experiment with GRIT (Ma et al., 2023) for the instantiation of $\mathfrak{F}_2$. We search the number of layers $k_2 \in \{1, 2, 4, 6\}$ and use Relative Random Walk Probabilities. Additionally, we set batch size $= 1$ and use distributed data parallelism based on torch.distributed. We also search hidden dimension for node/edge features in $\{32, 64\}$ for $\mathfrak{F}_1$ and in $\{64, 128\}$ for $\mathfrak{F}_2$. We also search learning rate in $\{0.001, 0.0005, 0.0001\}$ and weight decay in $\{0.0, 0.001\}$. After converging in a specific configuration, we then run the same experiment for all training size options. For the GMN baseline, we keep $k_1 \in \{1\}$ and the hidden size in $\{16, 32\}$. The rest of the hyperparameters are treated the same way as in H-GMN . The same procedure was followed for the MNIST ViT dataset (Tran et al., 2025). We evaluate the Transformer-NFN baseline (Tran et al., 2025) only on this dataset, since applying it to the $\sim$21M-parameter CIFAR100 ViT is intractable: its cost scales cubically with the embedding dimension of the input network and quadratically with the channel width of the metanetwork.

A.4.3 Mean Corruption Error Prediction

Predicting a checkpoint's robustness to distribution shift is a natural task for graph metanetworks. Following Hendrycks & Dietterich (2019), we measure robustness by the mean Corruption Error (mCE), which normalizes the error on each corruption type by that of a fixed reference model before averaging, and use it as the regression target for the metanetwork. For each model zoo, we compute mCE over the $|C| = 19$ corruption types of CIFAR-10-C or CIFAR-100-C (the 15 standard corruptions plus four additional ones introduced to test generalization beyond them) and five severity levels as shown in eq. (18):

$$CE_c^f = \frac{\sum_{s=1}^{5} E_{s,c}^f}{\sum_{s=1}^{5} E_{s,c}^{\text{ref}}}, \qquad \text{mCE}^f = \frac{1}{|C|} \sum_{c=1}^{|C|} CE_c^f. \tag{18}$$

The reference model is a single ResNet18 trained from scratch on CIFAR-10/CIFAR100. Since it is shared across all models in the zoo, the normalization only rescales each corruption type's contribution by a constant and does not affect the relative ordering of models on any individual corruption.

ResNet18 Zoo For the mean corruption error, we use the CIFAR-10-C dataset (Hendrycks & Dietterich, 2019). The distribution of mCE is shown in Figure 7b. **Hyperparameters.** For both H-GMN variants and GMN, we use the same hyperparameter search as for the ResNet18 zoo (section A.4.2). For SANE (Schürholt et al., 2024), we reuse the embeddings from pre-training on the full dataset and train the downstream head on the corresponding training subset.

CIFAR100 ViT Zoo For the CIFAR100 ViT zoo, we measure robustness as the mCE on CIFAR-100-C (Hendrycks & Dietterich, 2019), whose distribution is shown in Figure 8b. As for ResNet18, the hyperparameter search for both H-GMN and GMN follows section A.4.2.

A.5 Backdoor Detection

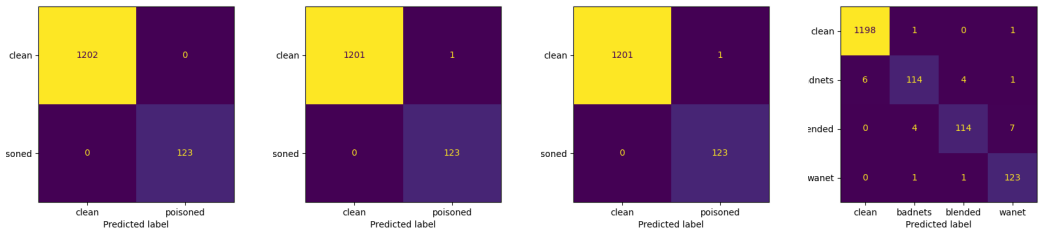
We use the backdoored ResNet18 zoo described in Section A.3.3, in which every poisoned model shares its train/validation/test split with its clean counterpart. We consider four classification tasks: three binary tasks (clean vs. BadNets, clean vs. Blended, and clean vs. WaNet) and a multi-class task that distinguishes clean models from each of the three attacks.

Since clean models far outnumber poisoned ones, we apply random subsampling to obtain balanced classes in the training split. Validation and test splits are kept intact. We compare four methods: StatNN (Unterthiner et al., 2020), the flat GMN baseline, and two variants of 2-lvl-H-GMN. Both variants apply a single GMN layer on the Parameter Graph ($k_1 = 1$) before pooling to the Layer Graph, on which $\mathfrak{F}_2$ is either a GMN or GRIT with RRWP (Ma et al., 2023) (2-lvl-H-GMN -GRIT). As before, the hyperparameter search closely follows section A.4.2 for the ResNet18 zoo.

Results. We select the checkpoint with the best validation F1 score. As shown in Table 4, GMN-GRIT is the strongest detector overall, almost perfectly separating poisoned from clean models for every attack, including the multi-class setting (see the confusion matrices in Figure 9). The flat GMN detects BadNets but collapses on Blended and WaNet, whose triggers are less conspicuous. StatNN remains a strong baseline throughout, suggesting that simple layer statistics already carry much of the backdoor signal. 2-lvl-H-GMN matches or outperforms StatNN on Blended and the multi-class task but falls behind on BadNets and WaNet. Instantiating $\mathfrak{F}_2$ with GRIT (Ma et al., 2023) closes this gap, hinting that long-range interactions between layers are important for this task.

A.6 Pruning

Towards complying with compute and energy requirements, numerous techniques have been developed to compress pre-trained networks (Hinton et al., 2015; Han et al., 2015; Wu et al., 2016), among which network pruning stands out as a leading approach (Yu et al., 2018; Wang et al., 2019). We consider unstructured pruning of individual weights (Dong et al., 2017) and train metanetworks to perform it via editing. However, to circumvent non-differentiability issues of mask selection, we



(a) Badnets VS clean (b) Blended VS clean (c) Wanet VS clean (d) Multiclass

Figure 9: Confusion matrices of 2-lvl-H-GMN (GRIT) predictions on each backdoor detection task.

derive the pruning mask via a simple heuristic, namely the ℓ_1 norm of the weights. The metanetwork's task is then to edit the unpruned parameters of the input network $f_{C,\boldsymbol{\theta}}$ to compensate for the pruned ones. H-GMN is thus instantiated as described in eq. (4), with the following training objective:

$$\min_{\tilde{\boldsymbol{\theta}}} \left(\mathop{\mathbb{E}}_{(\mathbf{x},y)\sim p_{\mathrm{d}}} \mathrm{d}\Big(f_{C,\boldsymbol{\theta}}(\mathbf{x}), f_{C,\tilde{\boldsymbol{\theta}}}(\mathbf{x}) \Big) \right), \tag{19}$$

where $(\mathbf{x}, y) \sim p_{\mathrm{d}}$ denotes labeled images not seen by $f_{C,\boldsymbol{\theta}}$ during training, and d is a statistical distance, $\tilde{\boldsymbol{\theta}} = \mathfrak{F}_{\mathrm{edit}}(f_{C,\boldsymbol{\theta}})$ as described in Section 4. In our experiments, we used Jensen–Shannon divergence (Lin, 1991). At inference time, $\mathfrak{F}_{\mathrm{edit}}$ requires access only to the network parameters $\boldsymbol{\theta}$. Since the pruning mask is computed by the same heuristic, H-GMN and GMN face exactly the same task at each pruning ratio: editing the same set of unpruned parameters so that the edited model remains functionally close to the original.

For the pruning experiment, we utilize the editing pipeline described in Section 4. We apply 1 GNN layer at the Parameter Graph resolution and PNA-style (Corso et al., 2020) pooling to obtain the coarsened graph. The $\mathfrak{F}_2$ model is then instantiated as GRIT (Ma et al., 2023) coupled with Relative Random Walk Probabilities, to achieve maximum receptive field. Finally, we employ the UnPooling operation of Equation (4) and one final GNN layer on the fine-grained resolution for the $\tilde{\mathfrak{F}}_1$ network. Both $\mathfrak{F}_1$ and $\tilde{\mathfrak{F}}_1$ are instantiated as single-direction GMNs in contrast to editing approaches followed by Kalogeropoulos et al. (2024); Kofinas et al. (2024).

To obtain the final weights in $\mathfrak{F}_{\mathrm{edit}}(f_{C,\boldsymbol{\theta}}) = \mathfrak{G}^{-1}\big(\psi(\tilde{\mathbf{h}}^{(1)})\big)$, we implement ψ as a residual update: the metanetwork output is scaled by learnable per-layer scalar multipliers and added to the input $\mathbf{h}^{(0)} = \mathfrak{G}(f_{C,\boldsymbol{\theta}})$. The metanetwork thus predicts a small correction to the input parameters instead of the new parameters themselves, following prior work (Zhou et al., 2023a; Kofinas et al., 2024; Kalogeropoulos et al., 2024). Moreover, to keep only the remaining parameters, ψ additionally applies the pruning mask to the updated parameters, so that pruned weights remain zero and only the unpruned ones are edited.

Hyperparameters: For H-GMN we only use 2-lvl-H-GMN and search for the number of layers in the coarsened level $k_2 \in \{1, 2, 3\}$. For the GMN baseline, we search $k_1 \in \{2, 3, 4\}$. For both of them, we search learning rate in $\{0.001, 0.0005, 0.0001\}$ and set fixed batch size of 4. After converging in a specific configuration, we then run the same experiment for all pruning intensity options. In our experiments, the best GMN configuration used two GNN layers ($k_1 = 2$). Even in this setup, where H-GMN and GMN have the same number of layers on the Parameter Graph, the additional cost of Pool_1, $\mathfrak{F}_2$, and UnPool_1 is negligible: inference takes 60 ms per network for H-GMN versus 58 ms for GMN.

A.7 Permutation equivariance of GMNs on MHSA parameter graphs

We show that the graph construction of section 5 maps the permutation symmetries of an MHSA block to relabellings of the vertices and edges of its parameter graph, and that GMNs are equivariant to these relabellings, provided that positional encodings and learnable embeddings are shared across heads. As in section 5, biases are omitted (see Remark A.4).

Parameter graph. Consider an MHSA block $f_{C,\boldsymbol{\theta}}$ with H heads and, for each head $h \in [H]$, parameters $\mathbf{W}_q^h, \mathbf{W}_k^h \in \mathbb{R}^{d_k \times d_{\text{in}}}$, $\mathbf{W}_v^h \in \mathbb{R}^{d_v \times d_{\text{in}}}$ and $\mathbf{W}_o^h \in \mathbb{R}^{d_{\text{out}} \times d_v}$, all collected in $\boldsymbol{\theta}$. Its parameter graph $\mathfrak{G}(f_{C,\boldsymbol{\theta}})$ has vertex set $\mathcal{V}_P = \mathcal{V} \cup \mathcal{V}_o \cup \bigcup_h (\mathcal{V}_k^h \cup \mathcal{V}_v^h)$ (a disjoint union), edge set $\mathcal{E}_P = \bigcup_h (\mathcal{E}_k^h \cup \mathcal{E}_v^h \cup \mathcal{E}_{kv}^h \cup \mathcal{E}_o^h)$, vertex features $\mathbf{x}_V$, positional encodings $\mathbf{p}_V, \mathbf{p}_E$ and edge features

$$\mathbf{x}_E(i,j) = \begin{cases} [\mathbf{W}_q^h(j,i), \mathbf{W}_k^h(j,i)], & (i,j) \in \mathcal{E}_k^h = \mathcal{V} \times \mathcal{V}_k^h, \\ [\mathbf{W}_v^h(j,i)], & (i,j) \in \mathcal{E}_v^h = \mathcal{V} \times \mathcal{V}_v^h, \\ [1/\sqrt{d_k}], & (i,j) \in \mathcal{E}_{kv}^h = \mathcal{V}_k^h \times \mathcal{V}_v^h, \\ [\mathbf{W}_o^h(j,i)], & (i,j) \in \mathcal{E}_o^h = \mathcal{V}_v^h \times \mathcal{V}_o, \end{cases} \qquad h \in [H]. \tag{20}$$

Here, (i,j) denotes an edge from i to j and, as in section 5, each vertex is identified with its index within its vertex set (e.g. $j \in \mathcal{V}_k^h$ indexes a row of $\mathbf{W}_q^h$). Only the features of $\mathcal{E}_k^h$, $\mathcal{E}_v^h$ and $\mathcal{E}_o^h$ depend on $\boldsymbol{\theta}$. All other attributes are $\boldsymbol{\theta}$-*independent*: the vertex features $\mathbf{x}_V$, since all vertices are parameter-free and receive learnable embeddings (section A.2), the features of $\mathcal{E}_{kv}^h$, and the positional encodings $\mathbf{p}_V, \mathbf{p}_E$, which include vertex and edge types.

Permutation Symmetries. Tran et al. (2025) studied the symmetries of Multi-Head Self-Attention modules and derived equivariant linear layers on their maximal symmetry group. Although self-attention parameters exhibit general linear symmetries, in this study, we focus on a subset thereof: permutation symmetries.

Every permutation $g = ((\pi_k^h, \pi_v^h)_{h=1}^H, \tau) \in S_{\boldsymbol{\theta}}^{\text{MHSA}} = (S_{d_k} \times S_{d_v})^H \rtimes S_H$, where S_n the symmetric group over n elements and $\rtimes$ the semi-direct product, transforms $\boldsymbol{\theta}$ into $\boldsymbol{\theta}' = g \cdot \boldsymbol{\theta}$, such that the MHSA function remains unaltered $f_{C,\boldsymbol{\theta}} = f_{C,g\cdot\boldsymbol{\theta}}$.[1] The element g induces the map $\pi_g : \mathcal{V}_P \to \mathcal{V}_P$,

$$\pi_g(i) = \begin{cases} i, & i \in \mathcal{V} \cup \mathcal{V}_o, \\ \text{the vertex of } \mathcal{V}_k^{\tau(h)} \text{ with index } \pi_k^h(i), & i \in \mathcal{V}_k^h, \\ \text{the vertex of } \mathcal{V}_v^{\tau(h)} \text{ with index } \pi_v^h(i), & i \in \mathcal{V}_v^h, \end{cases} \tag{21}$$

which fixes the input and output vertices, permutes the vertices within each head and relabels the heads by τ; it is a bijection since τ, π_k^h and π_v^h are bijections over their respective (non-overlapping) vertex sets.

To see why the function does not change, it is helpful to convert this into matrix form. The group acts on $\boldsymbol{\theta}$ as follows: for all $h \in 1, \ldots, H$:

$$\mathbf{W}_q'^h = \mathbf{P}_k^h \mathbf{W}_q^{\tau(h)}, \quad \mathbf{W}_k'^h = \mathbf{P}_k^h \mathbf{W}_k^{\tau(h)}, \quad \mathbf{W}_v'^h = \mathbf{P}_v^h \mathbf{W}_v^{\tau(h)}, \quad \mathbf{W}_o'^h = \mathbf{W}_o^{\tau(h)} (\mathbf{P}_v^h)^\top, \tag{22}$$

where $\mathbf{P}_k^h, \mathbf{P}_v^h$ are permutation matrices corresponding to, π_k^h, π_v^h (a permutation π maps to a matrix $\mathbf{P}$ with $\mathbf{P}(a, \pi(a)) = 1$ and all other entries zero and thus $(\mathbf{P}\mathbf{W})(a,i) = \mathbf{W}(\pi(a), i)$ and $(\mathbf{W}\mathbf{P}^\top)(j,a) = \mathbf{W}(j, \pi(a))$). Now, recall that $(\mathbf{P}_v^h)^\top \mathbf{P}_v^h = (\mathbf{P}_k^h)^\top \mathbf{P}_k^h = \mathbf{I}$. Thus head h of $f_{C,\boldsymbol{\theta}'}$ coincides with head $\tau(h)$ of $f_{C,\boldsymbol{\theta}}$. Additionally, $f_{C,\boldsymbol{\theta}'}$ is a summation over heads and since τ is a bijection, then $f_{C,\boldsymbol{\theta}'} = f_{C,\boldsymbol{\theta}}$.

Lemma A.1. *Let $g \in S_{\boldsymbol{\theta}}^{\text{MHSA}}$ and $\boldsymbol{\theta}' = g \cdot \boldsymbol{\theta}$. Then: (i)* Edge set: $(i,j) \mapsto (\pi_g(i), \pi_g(j))$ *maps $\mathcal{E}_k^h$, $\mathcal{E}_v^h$, $\mathcal{E}_{kv}^h$ and $\mathcal{E}_o^h$ onto $\mathcal{E}_k^{\tau(h)}$, $\mathcal{E}_v^{\tau(h)}$, $\mathcal{E}_{kv}^{\tau(h)}$ and $\mathcal{E}_o^{\tau(h)}$, respectively, for all $h \in [H]$; hence, $(i,j) \in \mathcal{E}_P$ if and only if $(\pi_g(i), \pi_g(j)) \in \mathcal{E}_P$. (ii)* Edge features: $\mathbf{x}'_E(i,j) = \mathbf{x}_E(\pi_g(i), \pi_g(j))$ *for all $(i,j) \in \mathcal{E}_k^h \cup \mathcal{E}_v^h \cup \mathcal{E}_o^h$ and $h \in [H]$.*

Proof. Write $\pi = \pi_g$. By (21), π is the identity on $\mathcal{V} \cup \mathcal{V}_o$ and restricts to bijections $\mathcal{V}_k^h \to \mathcal{V}_k^{\tau(h)}$ and $\mathcal{V}_v^h \to \mathcal{V}_v^{\tau(h)}$. (i) Hence, π maps $\mathcal{E}_k^h = \mathcal{V} \times \mathcal{V}_k^h$ onto $\mathcal{V} \times \mathcal{V}_k^{\tau(h)} = \mathcal{E}_k^{\tau(h)}$, and likewise for the other three edge sets. Since τ is a bijection, $\mathcal{E}_P$ is mapped onto $\mathcal{E}_P$; as π is injective, $(\pi(i), \pi(j)) \in \mathcal{E}_P$ implies $(i,j) \in \mathcal{E}_P$. (ii) By (20)–(21), for (i,j) in $\mathcal{E}_k^h$, $\mathcal{E}_v^h$ and $\mathcal{E}_o^h$, respectively,

$$\begin{aligned} \mathbf{x}_E(\pi(i), \pi(j)) &= [\mathbf{W}_q^{\tau(h)}(\pi_k^h(j), i), \mathbf{W}_k^{\tau(h)}(\pi_k^h(j), i)] && \text{for } (i,j) \in \mathcal{E}_k^h, \\ &= [\mathbf{W}_q'^h(j,i), \mathbf{W}_k'^h(j,i)] = \mathbf{x}'_E(i,j), \\ \mathbf{x}_E(\pi(i), \pi(j)) &= [\mathbf{W}_v^{\tau(h)}(\pi_v^h(j), i)] = [\mathbf{W}_v'^h(j,i)] = \mathbf{x}'_E(i,j), && \text{for } (i,j) \in \mathcal{E}_v^h, \\ \mathbf{x}_E(\pi(i), \pi(j)) &= [\mathbf{W}_o^{\tau(h)}(j, \pi_v^h(i))] = [\mathbf{W}_o'^h(j,i)] = \mathbf{x}'_E(i,j), && \text{for } (i,j) \in \mathcal{E}_o^h. \end{aligned}$$

[1] Throughout this section, a prime marks the counterpart of a quantity, when transforming $\boldsymbol{\theta}$ to $\boldsymbol{\theta}'$; e.g. $\mathbf{x}'_V$ and $\mathbf{x}'_E$ are the vertex and edge features of $\mathfrak{G}(f_{C,\boldsymbol{\theta}'})$.

□

Assumption A.2. For all $g \in S_{\boldsymbol{\theta}}^{\mathrm{MHSA}}$, the $\boldsymbol{\theta}$-independent attributes are invariant under π_g, i.e.

$$\begin{aligned} \mathbf{x}_V(\pi_g(i)) = \mathbf{x}_V(i), \quad \mathbf{p}_V(\pi_g(i)) = \mathbf{p}_V(i), &\qquad i \in \mathcal{V}_P, \\ \mathbf{p}_E(\pi_g(i), \pi_g(j)) = \mathbf{p}_E(i,j), &\qquad (i,j) \in \mathcal{E}_P, \\ \mathbf{x}_E(\pi_g(i), \pi_g(j)) = \mathbf{x}_E(i,j), &\qquad (i,j) \in \mathcal{E}_{kv}^h,\ h \in [H]. \end{aligned}$$

This holds whenever these attributes do not depend on the head index h or on the position of a vertex within $\mathcal{V}_k^h$ or $\mathcal{V}_v^h$. This is the case for the type-based positional encodings and the layer-wise embeddings of section A.2, provided they are shared across heads, and it is automatic for the constant features of $\mathcal{E}_{kv}^h$.

By Lemma A.1 and Assumption A.2, $\mathfrak{G}(f_{C,\boldsymbol{\theta}'})$ is thus $\mathfrak{G}(f_{C,\boldsymbol{\theta}})$ relabelled by π_g: π_g preserves the common edge set $\mathcal{E}_P$, and every attribute a of $\mathfrak{G}(f_{C,\boldsymbol{\theta}})$ (i.e. $\mathbf{x}_V$, $\mathbf{x}_E$, $\mathbf{p}_V$ or $\mathbf{p}_E$) and its counterpart a' satisfy

$$a'(i) = a(\pi_g(i))\ \ \forall i \in \mathcal{V}_P, \qquad a'(i,j) = a(\pi_g(i), \pi_g(j))\ \ \forall (i,j) \in \mathcal{E}_P. \tag{23}$$

Indeed, Lemma A.1(ii) covers $\mathbf{x}_E$ on $\mathcal{E}_k^h$, $\mathcal{E}_v^h$ and $\mathcal{E}_o^h$; all other attributes are $\boldsymbol{\theta}$-independent, so that $a' = a$ and (23) is Assumption A.2.

GMN. We consider GMNs with forward and backward messages, edge updates and positional encodings (Lim et al., 2024; Kofinas et al., 2024; Kalogeropoulos et al., 2024). With $\mathcal{N}_F(i) = \{j : (j,i) \in \mathcal{E}_P\}$, $\mathcal{N}_B(i) = \{j : (i,j) \in \mathcal{E}_P\}$ and $t = 1, \dots, T$, for $i \in \mathcal{V}_P$ and $(i,j) \in \mathcal{E}_P$,

$$\begin{aligned} \mathbf{h}_V^0(i) &= \mathit{INIT}_V\big(\mathbf{x}_V(i), \mathbf{p}_V(i)\big), \qquad \mathbf{h}_E^0(i,j) = \mathit{INIT}_E\big(\mathbf{x}_E(i,j), \mathbf{p}_E(i,j)\big), \\ \mathbf{m}_{V\text{-}F}^t(i) &= \textstyle\bigoplus_{j \in \mathcal{N}_F(i)} \mathit{MSG}_F^t\big(\mathbf{h}_V^{t-1}(i), \mathbf{h}_V^{t-1}(j), \mathbf{h}_E^{t-1}(j,i), \mathbf{p}_V(i), \mathbf{p}_V(j), \mathbf{p}_E(j,i)\big), \\ \mathbf{m}_{V\text{-}B}^t(i) &= \textstyle\bigoplus_{j \in \mathcal{N}_B(i)} \mathit{MSG}_B^t\big(\mathbf{h}_V^{t-1}(i), \mathbf{h}_V^{t-1}(j), \mathbf{h}_E^{t-1}(i,j), \mathbf{p}_V(i), \mathbf{p}_V(j), \mathbf{p}_E(i,j)\big), \\ \mathbf{h}_V^t(i) &= \mathit{UPD}_V^t\big(\mathbf{h}_V^{t-1}(i), \mathbf{m}_{V\text{-}F}^t(i), \mathbf{m}_{V\text{-}B}^t(i), \mathbf{p}_V(i)\big), \\ \mathbf{h}_E^t(i,j) &= \mathit{UPD}_E^t\big(\mathbf{h}_V^{t-1}(i), \mathbf{h}_V^{t-1}(j), \mathbf{h}_E^{t-1}(i,j), \mathbf{p}_V(i), \mathbf{p}_V(j), \mathbf{p}_E(i,j)\big), \end{aligned} \tag{24}$$

where $\bigoplus$ is a permutation-invariant aggregator, i.e. a function of the multiset of its arguments (e.g. sum, mean, max or PNA), and INIT, MSG and UPD are arbitrary functions (e.g. MLPs) shared across vertices and edges. Single-direction GMNs are obtained by dropping $\mathbf{m}_{V\text{-}B}^t$, and edge types enter through $\mathbf{p}_E$. For invariant tasks, a permutation-invariant readout $\mathit{READ}\big(\{\{\mathbf{h}_V^T(i) : i \in \mathcal{V}_P\}\}\big)$ follows.

Proposition A.3. *Under Assumption A.2, let $g \in S_{\boldsymbol{\theta}}^{\mathrm{MHSA}}$, $\boldsymbol{\theta}' = g \cdot \boldsymbol{\theta}$, and let $(\mathbf{h}_V^t, \mathbf{h}_E^t)$ and $(\mathbf{h}_V'^t, \mathbf{h}_E'^t)$ be the representations computed by the same GMN at layer t on $\mathfrak{G}(f_{C,\boldsymbol{\theta}})$ and $\mathfrak{G}(f_{C,\boldsymbol{\theta}'})$, respectively. Then, for all $t \in \{0, \dots, T\}$,*

$$\mathbf{h}_V'^t(i) = \mathbf{h}_V^t(\pi_g(i))\ \ \forall i \in \mathcal{V}_P, \qquad \mathbf{h}_E'^t(i,j) = \mathbf{h}_E^t(\pi_g(i), \pi_g(j))\ \ \forall (i,j) \in \mathcal{E}_P. \tag{25}$$

By (23)*, transforming the network by g relabels its parameter graph by π_g, and* (25) *states that the representations computed by the GMN are relabelled in the same way. Hence, the GMN is $S_{\boldsymbol{\theta}}^{\mathrm{MHSA}}$-equivariant, with g acting on parameters by $\boldsymbol{\theta} \mapsto g \cdot \boldsymbol{\theta}$ and on vertex and edge representations by relabelling with π_g; followed by a permutation-invariant readout, it is $S_{\boldsymbol{\theta}}^{\mathrm{MHSA}}$-invariant.*

Proof. Fix g and write $\pi = \pi_g$. By Lemma A.1(i), $j \in \mathcal{N}_F(i) \iff (j,i) \in \mathcal{E}_P \iff (\pi(j), \pi(i)) \in \mathcal{E}_P \iff \pi(j) \in \mathcal{N}_F(\pi(i))$, so, as π is a bijection of $\mathcal{V}_P$, it restricts to a bijection $\mathcal{N}_F(i) \to \mathcal{N}_F(\pi(i))$ and, likewise, $\mathcal{N}_B(i) \to \mathcal{N}_B(\pi(i))$, for every $i \in \mathcal{V}_P$. By (23), the positional encodings, which are the same for $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$, satisfy $\mathbf{p}_V(i) = \mathbf{p}_V(\pi(i))$ and $\mathbf{p}_E(i,j) = \mathbf{p}_E(\pi(i), \pi(j))$, i.e. the same identities as (25). As they enter (24) in the same way as $\mathbf{h}_V^{t-1}$ and $\mathbf{h}_E^{t-1}$, we omit them from the arguments below. We proceed by induction on t.

Base case ($t = 0$). By (23), $\mathbf{x}'_V(i) = \mathbf{x}_V(\pi(i))$ and $\mathbf{x}'_E(i,j) = \mathbf{x}_E(\pi(i), \pi(j))$. Hence,

$$\begin{aligned} \mathbf{h}_V'^0(i) &= \mathit{INIT}_V\big(\mathbf{x}'_V(i)\big) = \mathit{INIT}_V\big(\mathbf{x}_V(\pi(i))\big) = \mathbf{h}_V^0(\pi(i)), \\ \mathbf{h}_E'^0(i,j) &= \mathit{INIT}_E\big(\mathbf{x}'_E(i,j)\big) = \mathit{INIT}_E\big(\mathbf{x}_E(\pi(i), \pi(j))\big) = \mathbf{h}_E^0(\pi(i), \pi(j)). \end{aligned}$$

Induction step. Suppose (25) holds for $t-1$. For the forward messages,

$$\begin{aligned}\mathbf{m}'^{t}_{V\text{-}F}(i) &= \bigoplus\nolimits_{j\in\mathcal{N}_F(i)} MSG^t_F\big(\mathbf{h}'^{t-1}_V(i), \mathbf{h}'^{t-1}_V(j), \mathbf{h}'^{t-1}_E(j,i)\big)\\ &= \bigoplus\nolimits_{j\in\mathcal{N}_F(i)} MSG^t_F\big(\mathbf{h}^{t-1}_V(\pi(i)), \mathbf{h}^{t-1}_V(\pi(j)), \mathbf{h}^{t-1}_E(\pi(j),\pi(i))\big)\\ &= \bigoplus\nolimits_{j\in\mathcal{N}_F(\pi(i))} MSG^t_F\big(\mathbf{h}^{t-1}_V(\pi(i)), \mathbf{h}^{t-1}_V(j), \mathbf{h}^{t-1}_E(j,\pi(i))\big) = \mathbf{m}^t_{V\text{-}F}(\pi(i)),\end{aligned}$$

where the second equality is the induction hypothesis and the third holds because π is a bijection from $\mathcal{N}_F(i)$ to $\mathcal{N}_F(\pi(i))$, so both aggregations are over the same multiset. The same argument with $\mathcal{N}_B$ gives $\mathbf{m}'^{t}_{V\text{-}B}(i) = \mathbf{m}^t_{V\text{-}B}(\pi(i))$. Therefore,

$$\begin{aligned}\mathbf{h}'^{t}_V(i) &= UPD^t_V\big(\mathbf{h}'^{t-1}_V(i), \mathbf{m}'^{t}_{V\text{-}F}(i), \mathbf{m}'^{t}_{V\text{-}B}(i)\big)\\ &= UPD^t_V\big(\mathbf{h}^{t-1}_V(\pi(i)), \mathbf{m}^{t}_{V\text{-}F}(\pi(i)), \mathbf{m}^{t}_{V\text{-}B}(\pi(i))\big) = \mathbf{h}^t_V(\pi(i)),\\ \mathbf{h}'^{t}_E(i,j) &= UPD^t_E\big(\mathbf{h}'^{t-1}_V(i), \mathbf{h}'^{t-1}_V(j), \mathbf{h}'^{t-1}_E(i,j)\big)\\ &= UPD^t_E\big(\mathbf{h}^{t-1}_V(\pi(i)), \mathbf{h}^{t-1}_V(\pi(j)), \mathbf{h}^{t-1}_E(\pi(i),\pi(j))\big) = \mathbf{h}^t_E(\pi(i),\pi(j)),\end{aligned}$$

where $(\pi(i),\pi(j)) \in \mathcal{E}_P$ by Lemma A.1(i). This completes the induction. Finally, since π is a bijection of $\mathcal{V}_P$, the multisets $\{\{\mathbf{h}'^T_V(i) : i \in \mathcal{V}_P\}\}$ and $\{\{\mathbf{h}^T_V(i) : i \in \mathcal{V}_P\}\}$ coincide, and so do the outputs of any permutation-invariant readout. □

Remark A.4. With biases $\mathbf{b}^h_q, \mathbf{b}^h_k \in \mathbb{R}^{d_k}$, $\mathbf{b}^h_v \in \mathbb{R}^{d_v}$ and $\mathbf{b}_o \in \mathbb{R}^{d_{\text{out}}}$, we set $\mathbf{x}_V(j) = [\mathbf{b}^h_q(j), \mathbf{b}^h_k(j)]$ for $j \in \mathcal{V}^h_k$, $\mathbf{x}_V(j) = [\mathbf{b}^h_v(j)]$ for $j \in \mathcal{V}^h_v$ and $\mathbf{x}_V(j) = [\mathbf{b}_o(j)]$ for $j \in \mathcal{V}_o$, and g transforms the biases as $\mathbf{b}'^h_q = \mathbf{P}^h_k \mathbf{b}^{\tau(h)}_q$, $\mathbf{b}'^h_k = \mathbf{P}^h_k \mathbf{b}^{\tau(h)}_k$, $\mathbf{b}'^h_v = \mathbf{P}^h_v \mathbf{b}^{\tau(h)}_v$ and $\mathbf{b}'_o = \mathbf{b}_o$, which again preserves $f_{C,\boldsymbol{\theta}}$. As in Lemma A.1(ii), this gives $\mathbf{x}'_V(i) = \mathbf{x}_V(\pi_g(i))$ on $\mathcal{V}_o$ and on all $\mathcal{V}^h_k$, $\mathcal{V}^h_v$, so the condition on $\mathbf{x}_V$ in Assumption A.2 is only needed on $\mathcal{V}$, where it holds trivially. Hence, (23) still holds, and so does Proposition A.3, with the same proof. Moreover, since $\mathcal{V}^h_k$ and $\mathcal{V}^h_v$ are only incident to the edges in (20), the proof of Proposition A.3 applies when the block is part of a larger parameter graph, with π_g extended by the identity to all other vertices.