Fibration compression in deep neural networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

The performance of a deep network grows with the size of the network and the training data in a predictable fashion. This has led to very large networks that require ever increasing memory and power. Several studies have reported that learning generates redundant nodes that, in principle, can be removed to produce more compact networks. Using the concept of fibration symmetry from category theory, we propose an exact algorithm to identify the neurons that execute redundant computations, based on the weights of the network alone. We report here that such fibration symmetries emerge in many of the major network architectures. By pruning these redundant nodes we achieve nearly-lossless dramatic model compression: We achieve 31x compression of over-parametrized Transformers, while improving their scaling law; compress MLPs and CNNs to 17-20% of the original size, and reduce LSTMs in reinforcement learning to 20% of parameters without return loss. Fibration compression is complementary to existing quantization methods. Together, these methods may allow for the deployment of powerful models on edge devices.

1 Introduction

The recent surge in artificial intelligence (AI) has been propelled by the empirical scaling laws of Kaplan et al. [2020] and Hoffmann et al. [2022], which resulted in the "bigger is better" doctrine: more parameters, more layers, more data, and more compute. But scaling has limits. Frontier models now demand training and inference budgets that strain hardware and energy supply, and there is mounting pressure to compress them aggressively for deployment. Vendors typically respond by releasing a flagship "XL" model alongside scaled-down variants of reduced parameter count and depth—yet performance, both in loss and on downstream benchmarks, degrades sharply with size, often to the point where the smaller variants are no longer usable. The Llama 3.1 family with the canonical XL/M/S structure (405B / 70B / 8B) illustrates the pattern: as the parameter count drops from 405B to 70B to 8B, the accuracy of the MMLU benchmark falls from 85.2 to 79.3 to 66.7 and the AGIEval benchmark from 71.6 to 64.6 to 47.8 Dubey et al. [2024]). These are gaps in reasoning, mathematics, and code benchmarks that no amount of fine-tuning fully closes. Naive scaling-down is therefore neither sustainable nor satisfactory as a deployment strategy.

On the other hand, heuristic compression methods sidestep retraining from scratch but introduce their own problems. Existing pruning techniques are based on randomly chosen parameters and rely on activity or weight-based proxies for parameter importance: removing small magnitude weights Frankle and Carbin [2018], Han et al. [2015] or pruning nodes deemed redundant in a calibration set Yu et al. [2017], Ma et al. [2023]. These methods result in degradation of the accuracy of the model. Coarse layer-removal strategies have shown that large LLMs tolerate the deletion of entire blocks while preserving performance on downstream benchmarks, despite the degradation in validation loss Gromov et al. [2024]. These demonstrations of redundancy typically do not offer a first-principles account of which parameters are truly redundant.

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

Parameter pruning is orthogonal to quantization Rokh et al. [2023], which reduces the memory footprint by storing weights at lower precision Lin et al. [2024], Dettmers et al. [2023] rather than reducing their number; the two strategies are complementary and can be stacked. What is missing across all of these techniques is a principled criterion that mathematically guarantees the preservation of performance: each is a trial-and-error heuristic that, beyond modest compression ratios, causes decay in both loss and benchmark performance.

Deep neural networks are well known to harbor extensive redundancy in their activities: Several studies report widespread redundancy at node-level and "neural collapse" during the terminal phase of training Papyan et al. [2020], Doimo et al. [2022]. Velarde et al. [2026] previously observed the emergence of synchronized nodes: neurons whose activities track each other across all training inputs. Furthermore, stochastic gradient descent (SGD) drives different nodes to develop similar input and output weights Chen et al. [2023].

Such synchronization and weight-level redundancies are not accidents of optimization: in dynamical systems, they are the signature of an underlying graph symmetry known as a *fibration* Makse et al. [2026], Boldi and Vigna [2002], Morone et al. [2020]; a concept originally developed by Grothendieck in category theory Grothendieck [1959-1960] and later adapted to graphs by Boldi and Vigna [2002] and dynamical systems by Golubitsky and Stewart [2006] and applied to graph neural networks by Velarde et al. [2026]. Fibrations are *local* graph symmetries, less strict and much more flexible than the classical *global* automorphism groups familiar from geometric deep learning Bronstein et al. [2021]. Fibration symmetries have been shown to organize the structure and dynamics of biological networks across scales, from gene regulatory networks Morone et al. [2020], Leifer et al. [2021], Álvarez-García et al. [2025] to brain connectomes Avila et al. [2025], Gili et al. [2025]. Building on this, we propose a new theoretical formalism for symmetry in the computation graph that reflects the weight structure of deep neural networks.

To identify fibers in a computational graph, we develop a weighted "balanced coloring" algorithm that extends coloring algorithms for binary graphs Kamei and Cock [2013] to weighted graphs of neural networks. Then, we use the fibrations for substantial compression by pruning redundant nodes, thus preserving performance.

We tested the compression algorithm in a range of architectures, including multilayer perceptrons (MLP), CNNs, recurrent networks (e.g., long short-term memory, LSTM), and Transformers in both supervised and reinforcement learning (RL) settings. The targeted compression results in a more efficient parameter scaling law Kaplan et al. [2020] to achieve the same performance with fewer parameters. Previous network pruning methods are random heuristics based on the activity in the network. The present technique relies on the weight structure alone, providing a principled, data-independent route to massive compression that preserves both loss and performance.

2 Fibration symmetries in computational graphs

The nodes have *fibration symmetry* and are said to belong to the same *fiber* if they have isomorphic *input trees* Makse et al. [2026], Morone et al. [2020]. This means that the entire structure of connections from the graph's input to the nodes is isomorphic. In practice, fibers are identified by a balanced coloring algorithm from graph theory Makse et al. [2026], Kamei and Cock [2013]. This algorithm partitions the network into balanced coloring classes of nodes (fiber partition) by iteratively assigning the same color to nodes that receive the same set of input colors, hence the term *balanced coloring* Golubitsky and Stewart [2006]. An example is shown in Fig. 1a, where nodes in the same fiber are colored the same.

Fibration symmetries allow the graph to be compressed into a smaller *base* graph. This compression works by merging all nodes that share the same color (i.e., belong to the same fiber) while conserving the input trees. The resulting base graph performs the same computation as the original graph Makse et al. [2026]. In DNNs, it means that we can compress the network without losing performance.

2.1 Computational graphs as the basis of analysis

Fibration symmetries can emerge in all network architectures like MLP, CNNs, RNNs and Transformers (see Methods 5.1), yet, their analysis shares common features with the computational graph of the canonical MLP. So, we first analyze this architecture and then generalize. In this architecture,

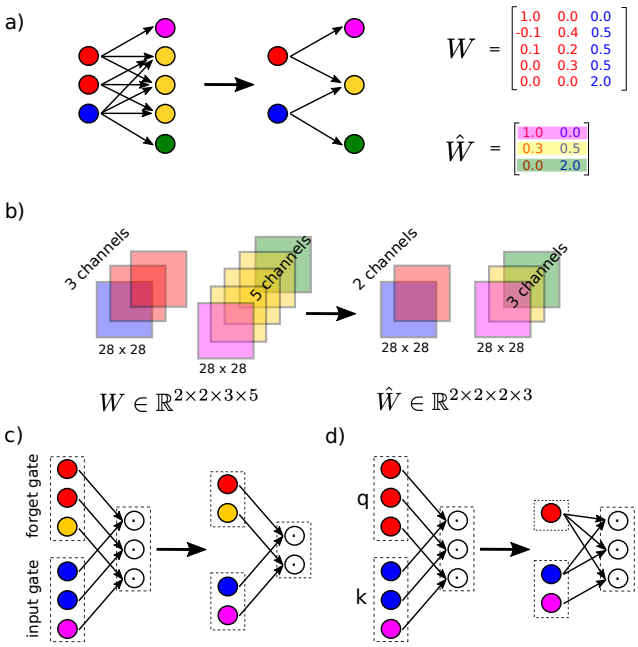

Figure 1: **Fibrations of different architectures**. **(a)** Example of fibration compression of a dense layer. Edges with zero weight are not plotted for simplicity. $\hat{W}$ is calculated based on W using Eq. (5). **(b)** Example of fibration compression of a convolution layer. It reduces the number of channels (3 to 2 and 5 to 3). **(c)** Example of fibration compression of gates in LSTM. The base has an identical number of nodes across all gates (here 2 nodes per gate). **(d)** The fibration compression for attention modules preserves the count of interactions (3 multiplications) even when the number of fibers of the k and q differ.

weighted edges $W_{ik}^{(\ell)}$ connect the node k in layer $\ell-1$ with the node i in layer ℓ, creating a directed weighted computational graph. The activity $h_i^{(\ell)}$ propagates through a weighted sum followed by a nonlinear activation function σ,

$$h_i^{(\ell)} = \sigma\left(\sum_k W_{ik}^{(\ell)} h_k^{(\ell-1)}\right). \tag{1}$$

from the input $x = h^{(0)}$ to the output $y = h^{(L)}$.

2.2 Fibrations and activity synchronization

A fibration symmetry in the weighted computational graph occurs when nodes share isomorphic input trees, meaning that they receive the same aggregated influence from the preceding layer's fibration symmetry classes. Formally, in a weighted graph, two nodes i and j in layer ℓ belong to the same fiber ($i \underset{\text{fib}}{\sim} j$) if they satisfy an equal-sum criterion for all colors c in the partition of the previous layer $\mathcal{C}_{\ell-1}^{\text{fib}}$:

$$\sum_{k\in c} W_{ik}^{(\ell)} = \sum_{k\in c} W_{jk}^{(\ell)}. \tag{2}$$

This criterion partitions the nodes in the layer ℓ into a coloring $\mathcal{C}_\ell^{\text{fib}}$ based on the input weights $W_{ik}^{(\ell)}$ and the color partitioning of the previous layer $\mathcal{C}_{\ell-1}^{\text{fib}}$. In words, this definition states that two nodes are in the same fiber iff for all colors from the previous layer, the sums of weights from these colors are the same. Or, more simply, the total input from each color is the same, which also ensures that

their activity is the same. This recursive definition starts in the input layer, $\ell = 0$, with all nodes (d_0 nodes) having distinct colors $\mathcal{C}_0^{\text{fib}} = \{1, 2, ..., d_0\}$, i.e., trivial fibers. The definition of fibers in Eq. (2) based on the sum of weights is the generalization from binary graphs to weighted graphs Boldi and Vigna [2002].

Similarly to symmetry-induced invariance theorems in physics Wilczek [2016], fibration symmetry induces activity synchronization during inference. This is: *Given two nodes i and j in layer ℓ, the following hold in networks with inputs x:*

$$i \underset{\text{fib}}{\sim} j \implies h_i^{(\ell)} = h_j^{(\ell)} \qquad \text{for any network input } x \tag{3}$$

The proofs for DNNs are shown in Anonymous [2026] which are based on analogous proofs for dynamical systems shown in Makse et al. [2026], DeVille and Lerman [2015], Nijholt et al. [2016]. Equation (1) establishes the relationship between the network structure captured by its weights and the network function captured by its activity. Therefore, the input tree structure governs the synchronization patterns.

2.3 Algorithm for coloring in feedforward networks

We designed a new method for coloring a layered feedforward weighted network to identify fibration symmetries. First, let us reformulate Eq. (2)

$$\begin{aligned} i \underset{\text{fib}}{\sim} j \iff \forall r : \sum_{k|c(k)=r} W_{jk}^{(\ell)} = \sum_{k|c(k)=r} W_{ik}^{(\ell)} &\iff \forall r : \hat{W}_{ir}^{(\ell)} = \hat{W}_{jr}^{(\ell)} \\ &\iff \hat{W}_{i,:}^{(\ell)} = \hat{W}_{j,:}^{(\ell)} \\ &\iff d(\hat{W}_{i,:}^{(\ell)}, \hat{W}_{j,:}^{(\ell)}) = 0 \\ &\iff 1 - \hat{W}_{i,:}^{(\ell)} . \hat{W}_{j,:}^{(\ell)} = 0 \qquad \text{if } \hat{W} \text{ is normalized} \end{aligned} \tag{4}$$

where $\hat{W}_{ir}^{(\ell)} = \sum_{k|c(k)=r} W_{ik}^{(\ell)}$.

Given that these conditions are impossible to satisfy exactly for continuous weights, we transform Eq. (4) into a set of inequality constraints parameterized by a tolerance ε. More precisely, we define the distance matrix $D_W^{(\ell)} = 1 - \hat{W}^{(\ell)} \hat{W}^{(\ell)T}$ and use it to calculate the colors/clusters ($i \underset{\text{fib}}{\sim} j$) using agglomerative clustering with distance threshold ε:

1. $c^{(\ell=0)} \in \mathcal{C}_0^{\text{fib}}$ assigns unique colors to each of the input features ($\ell = 0$).
2. For $\ell = 1, ..., N$:
   - 2.1. Calculation of $\hat{W}_{ir}^{(\ell)} = \sum_{k|c(k)=r} W_{ik}^{(\ell)}$ $\forall r \in \mathcal{C}^{(\ell)}$
   - 2.2. Normalization of $\hat{W}^{(\ell)}$.
   - 2.3. Calculation of the matrix $D_W^{(\ell)} = I - \hat{W}^{(\ell)} \hat{W}^{(\ell)T}$
   - 2.4. Agglomerative Clustering of the nodes in ℓ based on the distance matrix $D_W^{(\ell)}$ (see Alg. 1). We use a distance threshold $\varepsilon \in (0, 2)$.
   - 2.5. Fibers $c^{(\ell)} \in \mathcal{C}_\ell^{\text{fib}}$ are defined as the clusters.

Thus, agglomerative clustering employs a distance threshold ε to determine when to merge clusters. According to Eq. (4), this is equivalent to treating the equalities $\sum_{k \in c} W_{ik}^{(\ell)} = \sum_{k \in c} W_{jk}^{(\ell)}$ with a tolerance parameter ε, which defines the tolerance to weak fibration breaking (see Step 2.4). In other words, all approximate fibration symmetries should be regarded as quasi-fibrations Boldi et al. [2022] or pseudo-balanced colorings Leifer et al. [2022].

The coloring produced in a feedforward pass generates $\mathcal{C}_\ell^{\text{fib}}$ for all layers. The number of operations to execute this coloring algorithm in a layered feed-forward graph scales linearly with the parameter

Algorithm 1 Agglomerative Clustering

Initialize clusters $\mathcal{C} = 1, 2, ..., n$
while $|\mathcal{C}| > 1$ **do**
  $\forall c, c' \in \mathcal{C} : d_{\text{complete}}(c, c') = \max_{i \in c, j \in c'} D_W(i, j)$
  $d_{min} = \min_{c,c' \in \mathcal{C}} d_{\text{complete}}(c, c')$
  If $d_{min} > \varepsilon$: stop
  $\mathcal{C} \leftarrow$ Merge the clusters whose distance is less than d_{min}.
end while
return $\mathcal{C}$

size in each layer and linearly with the number of layers N. For recurrent networks, the fibration definition applies similarly, except that partitioning $\mathcal{C}_\ell^{\text{fib}}$ must be recursively iterated forward until convergence. Convergence requires, at most, the times of the longest loop in the recurrence following a theorem of Norris [1995].

The code is available at Github `github.com/fibration-anon/fibrations_in_dnns`.

2.4 Fibration compression preserves activity and loss

As mentioned above, the training process itself creates structure. For example, Chen et al. [2023], Papyan et al. [2020], Doimo et al. [2022] have observed redundancies, while Velarde et al. [2026] have reported the emergence of cluster synchronization during training, suggesting fibration symmetries.

In Methods section 2.3, we detail the color algorithms to identify fibers based on the weight matrices of the network. Once fibers are identified, one can produce a more compact base graph by merging all nodes in a fiber. To maintain identical forward computation, the new weights $\hat{W}$ are calculated by summing input weights with the same fiber color c and averaging over output weights with the same fiber color c' (proof in Anonymous [2026])

$$\hat{W}^{(\ell)}_{c'c} = \frac{1}{|c'|} \sum_{i \in c', k \in c} W^{(\ell)}_{ik}. \tag{5}$$

This targeted compression method (exemplified in Fig. 1a) preserves both activity and loss, allowing for drastic reductions in model size without performance degradation. This framework generalizes across architectures. In CNNs, feature channels are compressed rather than nodes (Fig. 1b). In RNNs and LSTMs, symmetries emerge in the gating mechanisms, leading to a base with identical node counts across all gates (Fig. 1c). In Transformers, symmetries emerge in the query, key, and value matrices, with compression preserving the count of interactions in attention modules, even when fiber counts for queries q and keys k differ (Fig. 1d).

3 Empirical results

We will demonstrate that these compression methods allow substantial compression of networks without changing their performance, and that this targeted fibration compression outperforms existing pruning methods (Section 3.2). We then show that more aggressive compression followed by retuning results in a new scaling law that outperforms the existing parameter scaling law for Transformers Kaplan et al. [2020] (Section 3.3). But first, we empirically confirm how fiber symmetries emerge during learning (Section 3.1).

3.1 Emergence of fibers during learning

We empirically validate the emergence of fibers during SGD for an MLP trained in MNIST and a CNN trained in ImageNet (see Fig. 3a, c). At the start of learning, all nodes have unique random connectivity, constituting trivial (single-node) fibers. As learning progresses, the nodes start to group into fibers. Hence, the number of unique fibers decreases. When the fibers stabilize and the accuracy no longer increases, the network has learned the training set. Details about the MLP and CNN used

here are provided in Methods 5.2.1 and 5.2.2. The emergence of symmetries is illustrated in the Supplementary Material *Movie S1* and Fig. 2. This shows results for the same CNN structure and MNIST training data as in Fig. 3a and c. The prevalence of large fibers in later layers of the network is explained by the increasingly more relaxed conditions for the fibers: Each layer removes an additional degree of freedom whenever two nodes satisfy the fiber equation (2). As we move across layers, the subspace of solutions increases and the fibers grow in size. This observation is consistent with the phenomenon of "node collapse" previously reported in the literature Papyan et al. [2020], Doimo et al. [2022].

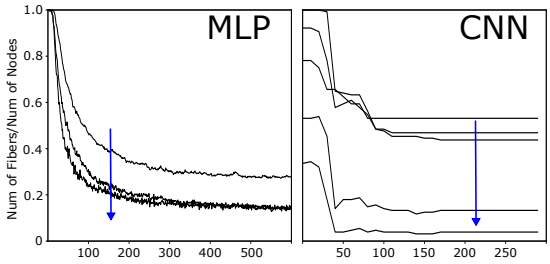

Figure 2: **Number of fibers in distinct layers across epochs:** As nodes group into fibers the number of distinct fibers decreases and they grow in size (individual nodes count as trivial fiber). This is the same as in Fig. 3a and c, but here each curve represents a separate layers. Higher layers in the networks (blue arrow) cluster into fibers more quickly and have a coarser partition at the end of learning (fewer larger fibers).

3.2 Fibration compression outperforms existing pruning methods

To identify fibers, we have to check if the weights satisfy Eq. (2). In practice, this can only be determined with a precision ε; that is, the coloring algorithm that identifies isomorphic input trees has a fiber precision ε as a parameter (Section 2.3). With increasing ε, a larger fraction of nodes can be compressed into (quasi-) fibers Boldi et al. [2022], but the forward computation is no longer exactly preserved and we observe a loss of performance.

For the MLP trained with MNIST data, fibration compression reduces the model to 17% of its original parameter count without an appreciable loss of classification accuracy (black point in Fig. 3b). We observe comparable compression for a CNN trained on ImageNet, where fibration compression achieves reductions of similar magnitude without degradation in test performance (Fig. 3d).

To contextualize the efficiency of our fibration compression, we compare it against two standard pruning baselines: random pruning and L2-norm-based pruning (see Fig. 3b-d). Pruning is a widely used model compression technique aimed at reducing a neural network's size and computational footprint by removing redundant or less important parameters Meng et al. [2020]. Traditionally, pruning algorithms often rely on parameter magnitude as a proxy for importance, under the assumption that smaller weights contribute less to the network's output.

1. Random pruning serves as a naive baseline, where nodes are removed uniformly at random, independent of their structural or functional role in the network.
2. L2-norm pruning employs a more structured criterion: it removes the nodes with the smallest input-weight L2 norm, based on the heuristic that nodes with weaker incoming connections are less critical.

In contrast, our fibration compression identifies and merges nodes that play identical functional roles, resulting in a compressed network that preserves the forward computation. For a fair comparison, all methods are constrained to produce a compressed network with the same number of nodes as the fibration base. For both MLP and CNN, we find that alternative compression methods are less effective. In Fig. 3b-d, fibration compression (red curve) outperforms random pruning nodes (blue curve) and pruning nodes with small weights (L2-norm, pink curve).

Next, we evaluate loss-preserving compression in a reinforcement learning context using an LSTM-based agent trained to play the Atari game Beam Rider (details in Methods 5.2.3, see Fig. 3e, green

curve). In training epoch 50, we compress the network using $\varepsilon = 0.3$ (black), achieving a network with 40% of the original parameters. The compressed network did not exhibit degraded performance after continued training. The latter continues to create fibers, allowing for further compression to 20% of the original model size in episode 300 without appreciable loss of performance.

3.3 Fibration compression with retuning reveals efficient parameter scaling for Transformers

For larger compression factors (achieved with larger ε) the loss in performance can be compensated for by retuning the network after compression, a standard approach in existing pruning methods Meng et al. [2020], Vadera and Ameen [2020]. We explore this in the context of a sequence-to-sequence Transformer trained for German-English translation (Methods 5.3) following the original work on Transformers Vaswani et al. [2023].

For this purpose, we develop a fibration compression and retuning method with adaptive fiber precision (Methods 5.4 - Fig. S5) resulting in retuning curves (Fig. 3f, orange and purple curves). For a model with 78M parameters, we compress the transformer by a factor of 31x from its original size with a 4.1% loss in performance (black arrow on the orange retuning curve in Fig. 3f). In Fig. 3f, we compare this method with other approaches: standard pruning methods (blue and pink curves), and fibration compression without retuning while searching for the optimal ε value for each layer individually (red curve).

Next, we explore the performance gains that can be achieved with increasing network size. It is well established that performance can be improved by increasing the number of parameters under a fixed size of training data Kaplan et al. [2020]. We trained Transformers with varying network sizes (e.g. 50M and 78M parameters - purple and orange curves in Fig.3f and orange retuning curves in Fig.3g) and then compress and retune each network. The baseline model (without compression) follows the established parameter scaling law Kaplan et al. [2020] between the loss function $\mathcal{L}$ and the number of parameters of the baseline model P, with a power-law behavior (black dashed line):

$$\mathcal{L} = L_0^p \; P^{-\alpha_p} \,, \tag{6}$$

This scaling law is fitted with a linear function between $\log(\mathcal{L})$ and $\log(P)$ (dashed black line in Fig. 3g), resulting in coefficients $L_0^p = 3.0$ (from the offset) and $\alpha_p = 0.028$ (from the slope). Following the original work Kaplan et al. [2020], the baseline model is obtained by increasing the number of parameters by increasing the dimension of all layers by the same amount. For each baseline model, we then apply the fibration compression and retuning method and find a more efficient (larger exponent) scaling law for the compressed network (orange dashed line):

$$\mathcal{L} = L_0^f \; (P_{min|L})^{-\alpha_f} \,. \tag{7}$$

For each P, this gives a tuning curve (orange curves on Fig. 3g). Then, for a given loss value L, we find the smallest network (at all points on the orange curves and at all curves). This gives the minimum number of parameters $P_{min|L}$ of the compressed model with a given loss value and the baseline model. For this set of loss values in the range of 1.9-2.2 we fit a line as before (orange dashed line, in Fig. 3g) resulting in $L_0^f = 3.8$ (from the offset) and $\alpha_f = 0.049$.

The power-law exponent obtained with the fibration compression protocol is larger than the value obtained without compression, implying a faster-decaying scaling curve. In total, in the constant data regime, performance can be improved by first training a larger network followed by loss-preserving compression and retuning. With this, compression efficiency increases with model size since $\alpha_f > \alpha_p$.

3.4 Limitations and future work

One caveat to our results on the transformer is that we used over-parametrized models with a large parameter count relative to the data size (ratio of $\sim$ 1:68 in the number of token versus parameters). However, modern large language models, starting with Chinchilla, use a ratio of 20:1 or more Hoffmann et al. [2022].

Several important questions remain. First, future work will have to determine how much transformers can be compressed at these higher token-to-parameter ratios. Second, modern LLM architectures

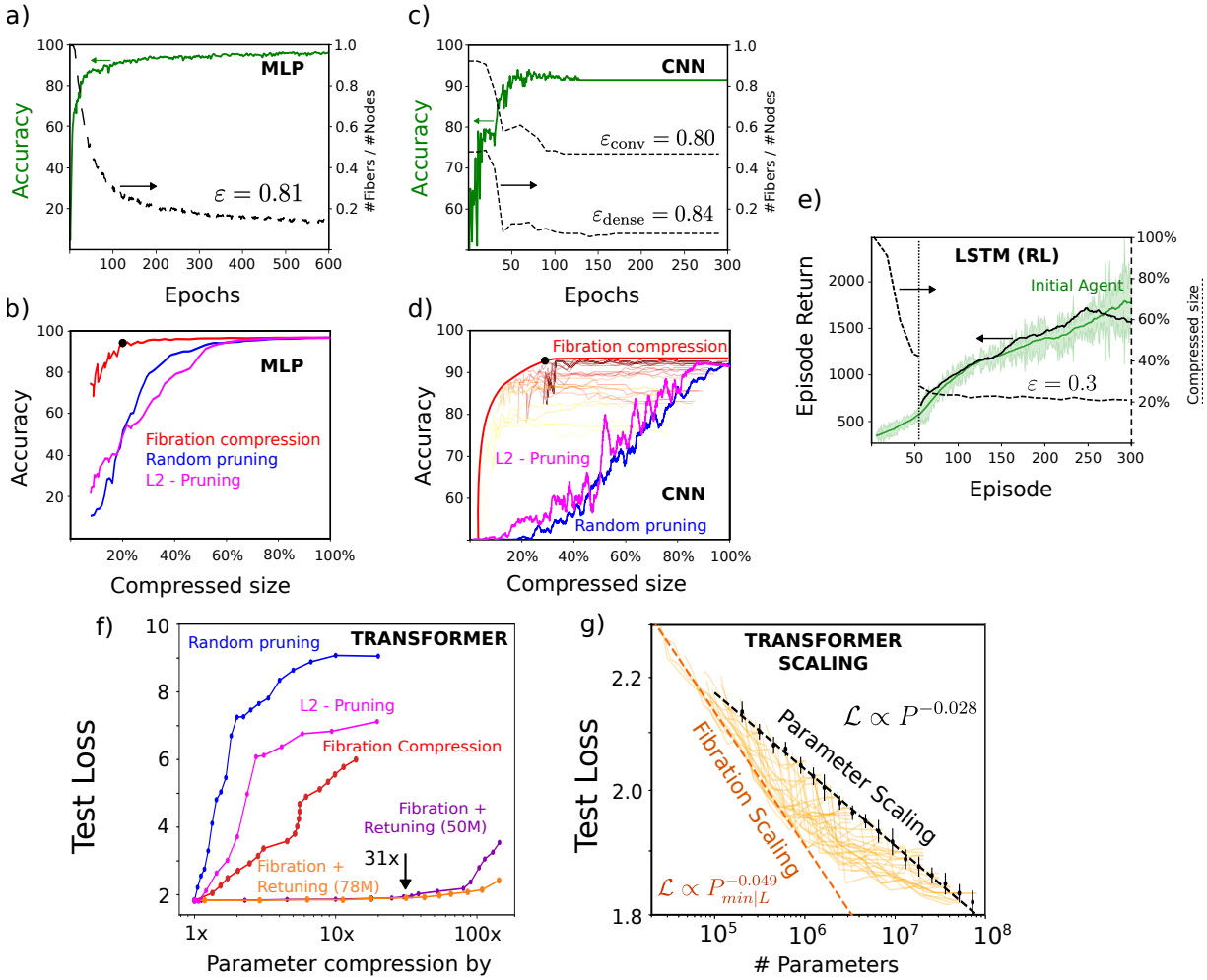

Figure 3: **Empirical results. (a)** MLP trained on MNIST dataset. Classification accuracy increases across learning epochs (green), while the number of trivial (single-node) fibers decreases in favor of larger multi-node fibers (black dashed curves). **(b)** Accuracy of the compressed network depends on the level of compression (red). Alternative pruning methods at different levels of compression (blue and pink curves). **(c)** Same as (a) but for CNN trained on ImageNet. In panels (a)-(c) arrows point to the axis associated with each curve. **(d)** Same as (b) for CNN. Yellow to red curves correspond to different values of $\varepsilon_{\text{conv}}$. Bold red curve represents maximum performance for different combinations of ($\varepsilon_{\text{conv}}$, $\varepsilon_{\text{dense}}$). **(e)** Agent with LSTM trained via reinforcement learning to play the Atari game Beam Rider. Returns increase with learning episode (solid curves). Learning with no compression (green). At episode 50 (vertical line), the network is compressed (black) and training continues. Size of fibration base (dashed curves). **(f)** Transformers trained on the Multi30k dataset. Fibration compression using layer-specific optimal ε (red). Fibration compression followed by retuning for a model with 50M parameters (purple) and 78M (orange). Blue and pink as in (b). **(g)** Power-law scaling of the loss value as a function of the number of parameters in the original transformer (black) and after fibration compression and retuning (orange curves). Parameter count excludes the embedding layers. Dashed scaling lines are fit to the original networks and to the most compressed network at a fixed loss. More details in the text.

have a number of architectural features that will require further exploration. For example, single-token prediction has become the dominant architecture. Third, combining fibration compression with existing compression techniques, such as quantization Rokh et al. [2023] or Singular Vector Decomposition (SVD) Sharma et al. [2023], Tomut et al. [2024] requires careful ordering and

joint optimization. Initial analysis suggests fibration compression should precede these compression methods, but optimal pipelines need investigation. Fourth, identifying the optimal selection of the pseudo-fiber threshold for each layer in a network is a non-trivial optimization problem that deserves further attention.

4 Conclusions

We have shown that fibration symmetries emerge during training in major network architectures and for various tasks. Fiber compression yields substantial efficiencies in memory and computation during inference. In particular, in the fixed data regime, an established parameter-scaling law tells us that performance can increase with increasing network size Kaplan et al. [2020]. We have shown that this increase in performance can be achieved with a more favorable scaling exponent, resulting in substantially smaller transformer networks.

Fibration compression is complementary to quantization Rokh et al. [2023], which reduces the footprint in memory by using lower-precision weights, typically by a factor of 3-4x without loss of performance Lin et al. [2024], Dettmers et al. [2023]. Fibration compression operates by reducing the number of nodes and weights, while quantization reduces the memory size of the weight matrices. Stacking these approaches could potentially enable larger compressions. SVD compression of the weight matrix can identify a linear subspace that can be pruned with little effect on the resulting computation. This heuristic Sharma et al. [2023], Tomut et al. [2024] can be stacked to fibration.

Because fibration compression is independent of the data, it can be applied to models after they have been trained. In contrast, existing pruning methods are based on the activity in a particular data set Frankle and Carbin [2018], Han et al. [2015], Yu et al. [2017], Ma et al. [2023]. The fine-tuning we have done here can be done on a small subset of the data, similar to other pruning methods Gromov et al. [2024]. Thus, in general, fibration compression can be applied to any existing trained model, which promises a wide applicability of the approach.

We anticipate that increased efficiency may result in a wider adoption of powerful models in resource constrained settings.

Acknowledgments

Omitted for anonymization.

Code

Anonymized at `github.com/fibration-anon/fibrations_in_dnns`.

References

L. A. Álvarez-García, W. Liebermeister, I. Leifer, and H. A. Makse. Complexity reduction by symmetry: Uncovering the minimal regulatory network for logical computation in bacteria. *PLoS Comput. Biol.*, 21:e1013005, 2025.

Anonymous. Emergent fibration symmetries enable loss-preserving compression and state-of-the-art continual learning in deep neural networks. *Preprint*, 2026. URL `https://github.com/fibration-anon/fibrations_in_dnns/blob/main/emergent_fibration_ai.pdf`.

B. Avila, P. Augusto, A. Hashemi, D. Phillips, T. Gili, M. Zimmer, and H. A. Makse. Symmetries and synchronization from whole-neural activity in *c. elegans* connectome: Integration of functional and structural networks. *Proc. Natl. Acad. Sci. USA*, 122:e2417850122, 2025.

P. Boldi and S. Vigna. Fibrations of graphs. *Discrete Math.*, 243:21–66, 2002.

P. Boldi, I. Leifer, and H. A. Makse. Quasifibrations of graphs to find symmetries and reconstruct biological networks. *J. Stat. Mech.: Theor. Exp.*, 2022:113401, 2022.

M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv*, 2021. URL `https://arxiv.org/abs/2104.13478`.

F. Chen, D. Kunin, A. Yamamura, and S. Ganguli. Stochastic collapse: How gradient noise attracts SGD dynamics towards simpler subnetworks. *Advances in Neural Information Processing Systems*, 36:35027–35063, 2023.

T. Dettmers, R. Svirschevski, V. Egiazarian, D. Kuznedelev, E. Frantar, S. Ashkboos, A. Borzunov, T. Hoefler, and D. Alistarh. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*, 2023.

L. DeVille and E. Lerman. Modular dynamical systems on networks. *J. Eur. Math. Soc.*, 17:2977–3013, 2015.

D. Doimo, A. Glielmo, S. Goldt, and A. Laio. Redundant representations help generalization in wide neural networks. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 19659–19672. Curran Associates, Inc., 2022.

A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL `https://arxiv.org/abs/2407.21783`.

D. Elliott, S. Frank, K. Simaan, and L. Specia. Multi30k: Multilingual english-german image descriptions. In *Proceedings of the 5th Workshop on Vision and Language*, pages 70–74. Association for Computational Linguistics, 2016.

J. Frankle and M. Carbin. The lottery ticket hypothesis: Training pruned neural networks. *CoRR*, abs/1803.03635, 2018. URL `http://arxiv.org/abs/1803.03635`.

T. Gili, B. Avila, L. Pasquini, A. Holodny, D. Phillips, P. Boldi, A. Gabrielli, G. Caldarelli, M. Zimmer, and H. A. Makse. Fibration symmetry-breaking supports functional transitions in a brain network engaged in language. *arXiv*, 2025. URL `https://arxiv.org/abs/2409.02674`.

M. Golubitsky and I. Stewart. Nonlinear dynamics of networks: the groupoid formalism. *Bulletin of the American Meteorological Society*, 43:305–364, 2006.

A. Gromov, K. Tirumala, H. Shapourian, P. Glorioso, and D. A. Roberts. The unreasonable ineffectiveness of the deeper layers. *arXiv preprint arXiv:2403.17887*, 2024.

A. Grothendieck. Technique de descente et théorémes d'existence en géométrie algébrique, I. Généralités. Descente par morphismes fidélement plats. *Seminaire Bourbaki*, 190, 1959-1960.

S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626, 2015. URL `http://arxiv.org/abs/1506.02626`.

J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. Casas, L. A. Hendricks, J. Welbl, A. Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 10, 2022.

H. Kamei and P. J. A. Cock. Computation of balanced equivalence relations and their lattice for a coupled cell network. *SIAM J. Appl. Dyn. Syst.*, 12:352–382, 2013.

J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv*, 2020. URL `https://arxiv.org/abs/2001.08361`.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv*, 2017. URL `https://arxiv.org/abs/1412.6980`.

I. Leifer, M. Sánchez-Pérez, C. Ishida, and H. A. Makse. Predicting synchronized gene coexpression patterns from fibration symmetries in gene regulatory networks in bacteria. *BMC Bioinformatics*, 22:363, 2021.

I. Leifer, D. Phillips, F. Sorrentino, and H. A. Makse. Symmetry-driven network reconstruction through pseudobalanced coloring optimization. *J. Stat. Mech.: Theor. Exp.*, 2022:073403, 2022.

J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6:87–100, 2024.

X. Ma, G. Fang, and X. Wang. Llm-pruner: On the structural pruning of large language models, 2023. URL `https://arxiv.org/abs/2305.11627`.

H. A. Makse, P. Boldi, F. Sorrentino, and I. Stewart. *Symmetries of Living and Artificial Intelligence Systems: Fibrations and Synchronization in Networks*. Cambridge University Press, forthcoming, 2026. URL `https://arxiv.org/pdf/2502.18713`.

F. Meng, H. Cheng, K. Li, H. Luo, X. Guo, G. Lu, and X. Sun. Pruning filter in filter. In *Advances in Neural Information Processing Systems*, volume 33, pages 17629–17640. Curran Associates, Inc., 2020.

F. Morone, I. Leifer, and H. A. Makse. Fibration symmetries uncover the building blocks of biological networks. *Proc. Nat. Acad. Sci. USA*, 117:8306–8314, 2020.

E. Nijholt, B. Rink, and J. Sanders. Graph fibrations and symmetries of network dynamics. *Diff. Equ.*, 261:4861–4896, 2016.

N. Norris. Universal covers of graphs: Isomorphism to depth n-1 implies isomorphism to all depths. *Discrete Applied Mathematics*, 56:61–74, Jan. 1995. ISSN 0166-218X.

V. Papyan, X. Han, and D. L. Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. *Proc. Natl. Acad. Sci. USA*, 117:24652–24663, 2020.

B. Rokh, A. Azarpeyvand, and A. Khanteymoori. A comprehensive survey on model quantization for deep neural networks in image classification. *ACM Transactions on Intelligent Systems and Technology*, 14(6):1–50, 2023.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv*, 2017. URL `https://arxiv.org/abs/1707.06347`.

P. Sharma, J. T. Ash, and D. Misra. The truth is in there: Improving reasoning in language models with layer-selective rank reduction. *arXiv preprint arXiv:2312.13558*, 2023.

A. Tomut, S. S. Jahromi, A. Sarkar, U. Kurt, S. Singh, F. Ishtiaq, C. Muñoz, P. S. Bajaj, A. Elborady, G. del Bimbo, et al. Compactifai: extreme compression of large language models using quantum-inspired tensor networks. *arXiv preprint arXiv:2401.14109*, 2024.

S. Vadera and S. Ameen. Methods for pruning deep neural networks. *IEEE Access*, 2020.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *arXiv*, 2023. URL https://arxiv.org/abs/1706.03762.

O. Velarde, L. C. Parra, P. Boldi, and H. A. Makse. The role of fibration symmetries in geometric deep learning. *Proc. Natl. Acad. Sci. USA*, 123:e2416552123, 2026.

F. Wilczek. *A Beautiful Question: Finding Nature's Deep Design*. Penguin Press, 2016.

R. Yu, A. Li, C. Chen, J. Lai, V. I. Morariu, X. Han, M. Gao, C. Lin, and L. S. Davis. NISP: pruning networks using neuron importance score propagation. *CoRR*, abs/1711.05908, 2017. URL http://arxiv.org/abs/1711.05908.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main contribution of the paper are described in the abstract. Section 4 shows the set of experiments to verify the claims.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We have added a section title "Limitations and future work".

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: The paper is mainly empirical, but the theoretical statements have the full set of assumptions and the complete proofs explained in an anonymized companion paper provided via a reference to a github link.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The experiments setup and data are described in Methods in detail. The code is publicly available.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code is publicly available.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: See Methods

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes] and [No]

   Justification: The effects of the central claims are substantial, evident in single runs. One exception was for the results on the reinforcement learning example (Fig. 3e). There, we have performed multiple runs and given 95% confidence intervals.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: See Methods. All experiments utilize an NVIDIA RTX 4090 GPU (CUDA Version 12.5) and an AMD Ryzen 9 7950X 16-Core Processor.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: No animal of human subjects were involved in this research. Only publicly available data was used.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This is foundational research on symmetries and compression in DNNs. While compressing large models should be beneficial to reduce resource consumption, Jevons's paradox does not assure us real savings. So we prefer to stay silent on this.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.

- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [N/A]

    Justification: The paper does not use existing assets.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: the paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.

5 Methods

5.1 Deep neural networks and computational graphs

Here, we define the main network architectures considered in this study. The structure of MLP and CNN can be captured by a simple *graph*, which connects one node to the next with an edge (Fig. S4a and b). More generally, modern network architectures such as LSTM and Transformers can be described as *hypergraph* (Fig. S4c and d). In a hypergraph, two or more nodes interact before providing input to the next node. This interaction – often a product or a sum – can be represented as a "factor" (the circles in Fig. S4c and d).

Multilayer Perceptron (MLP)

A MLP with $N-1$ hidden layers is described by the following equations.

$$\begin{aligned} z^{(\ell)} &= W^{(\ell)} h^{(\ell-1)} + b^{(\ell)}, \\ h^{(\ell)} &= \sigma(z^{(\ell-1)}), \\ \hat{y} &= h^{(N)}, \end{aligned}$$

where $z_i^{(\ell)}$ and $h_i^{(\ell)}$ are the input and activity of the node $i = 1, ..., d_\ell$ in the layer $\ell = 1, ..., N$, respectively. The vector $b^{(\ell)} \in \mathbb{R}^{d_\ell}$ is called the bias of the layer ℓ and the matrix $W^{(\ell)} \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ is the weight matrix from layer $\ell - 1$ to layer ℓ. σ is a non-linear activation and $\hat{y}$ is the prediction of the network for input $x = h^{(0)}$.

Convolutional Neural Network (CNN)

A CNN with N convolutions (symbol $*$) and average pooling is described by,

$$\begin{aligned} \mathbf{H}^{(\ell)} &= \text{AvgPool}[(\sigma(\mathbf{H}^{(\ell-1)} * \mathbf{W}^{(\ell)} + \mathbf{b}^{(\ell)})], \\ \hat{y} &= \text{AvgPool}(\mathbf{H}^{(N)}) \in \mathbb{R}^{d_N} \end{aligned}$$

where $\mathbf{W}^{(\ell)} \in \mathbb{R}^{D \times D \times d_{\ell-1} \times d_\ell}$ is called kernel (size $D \times D$), $\mathbf{b}^{(\ell)} \in \mathbb{R}^{d_\ell}$ is the bias and $\mathbf{H}^{(\ell)} \in \mathbb{R}^{L \times L \times d_\ell}$ is the activity for all layers $\ell = 1, ..., N$. The dimension d_ℓ is the number of channels in each layer ℓ.

Recurrent Neural Networks (RNN) - LSTM

The LSTM is a particular form of a recurrent neural network (RNN). Although LSTMs have been replaced by Transformers, recurrence remains important because it can implement multiple processing stages in a more compact network, and because recurrence is the dominant architecture in biological neural network. The LSTM is shown as a hypergraph in Fig. S4c. The dynamic in time t is defined as:

$$\begin{aligned} \text{Input gate:} \quad & i_t = \sigma(W_{hi} h_{t-1} + W_{ii} x_t + b_i) \\ \text{Forget gate:} \quad & f_t = \sigma(W_{hf} h_{t-1} + W_{if} x_t + b_f) \\ \text{Output gate:} \quad & o_t = \sigma(W_{ho} h_{t-1} + W_{io} x_t + b_o) \\ \text{Cell gate:} \quad & g_t = \tanh(W_{hg} h_{t-1} + W_{ig} x_t + b_g) \\ \text{Cell state:} \quad & C_t = f_t \odot C_{t-1} + i_t \odot g_t \\ \text{Hidden state:} \quad & h_t = o_t \odot \tanh(C_t) \end{aligned}$$

where x_t is the input at time t, and $\odot$ element-wise multiplication. In an LSTM and Transformer the input is a sequence of values, and time t selects the elements in that sequence. Otherwise, in the rest of the text, t refers to a learning time step.

As with most RNN, the LSTM contains "gates" that multiply with the state of a node. For example, Cell State C_t takes as input the Cell gate g_t and input i_t combining them with $\odot$ — an element-wise

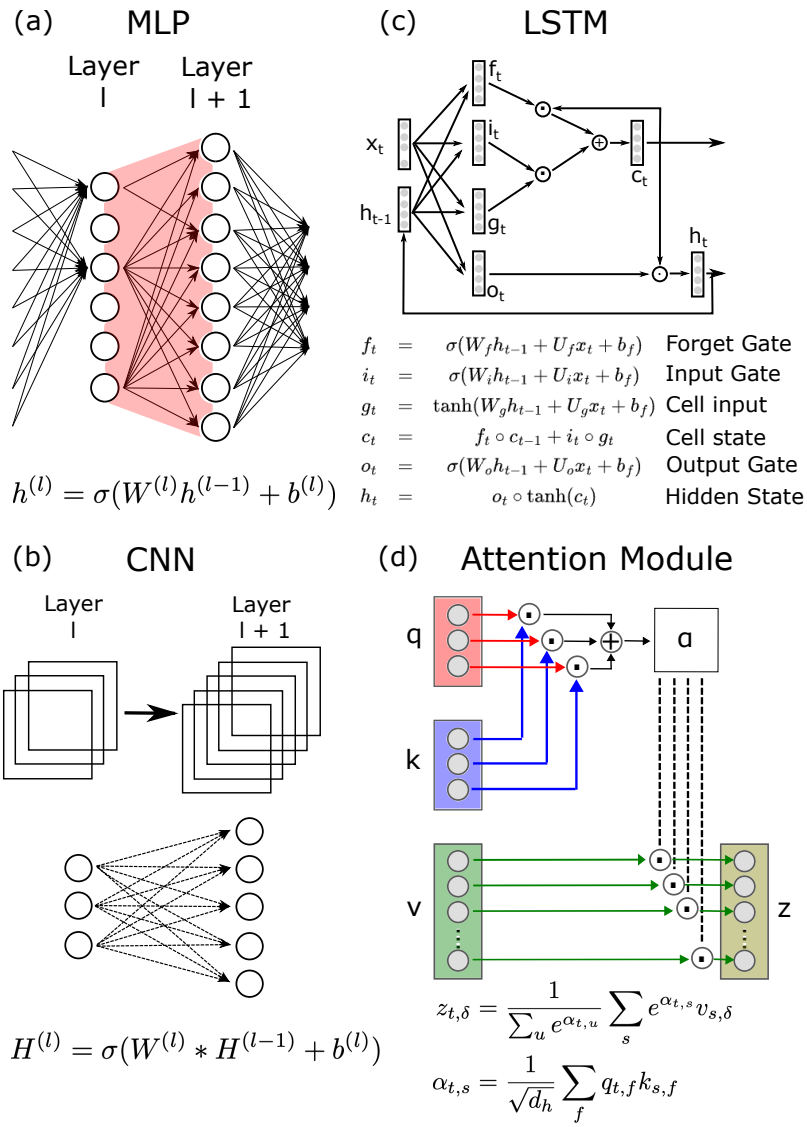

Figure S4: **Schematic representation of deep learning architectures as computational graphs and hypergraphs.** The figure illustrates the core topologies of: (a) MLP (Multilayer Perceptron), (b) CNN (Convolutional Neural Network), (c) LSTM (Long Short-Term Memory), and (d) Attention Module, along with their characteristic equations. In MLP and CNN, f is a non-linear activation. For LSTM, activations are represented with sigmoid and hyperbolic tangent. In all the cases, W, U and b are trainable parameters. The symbol $\circ$ and $+$ represents the operations product and sum element-wise, resp. In Attention Module, α is called attention score calculated based on **q**uery and **k**ey. The output z is proportional to **v**alue.

multiplication: $C_t = f_t \odot C_{t-1} + i_t \odot g_t$. Let us consider the second term in this sum. It represents a multiplicative two-node interaction. For a fiber to emerge in this product, two nodes must be a fiber in the corresponding nodes of i_t **and** in a fiber of nodes of g_t, as exemplified with the colors in Fig. 1c. Two nodes in the product have the same colors, if they have the same colors in both factors. The same is true for the element-wise product of f_t and C_{t-1}. The sum can be treated as a concatenation of nodes $[f_t \odot C_{t-1}; i_t \odot g_t]$ with a weight matrix $[Id|Id]$ and the same conditions as in Eq. (2). This definition constitutes a general algorithm to identify fibers in any feedforward hypergraph composed of sums and products.

Attention modules and Transformers

Transformers are architectures that use the "attention" mechanism (see Fig. S4d) to process sequential data. The attention mechanism is defined by the following equations

$$\begin{aligned} \text{Query, Key, Value:} \quad & \mathbf{Q} = \mathbf{X}\mathbf{W}^Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}^K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}^V \\ \text{Score:} \quad & \alpha = \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} \\ \text{Context vector:} \quad & \mathbf{z} = \text{softmax}(\alpha)\mathbf{V} \end{aligned}$$

where $\mathbf{X} \in \mathbf{R}^{T\times N}$ is the embedding of tokens (items in a sequence), $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V$ are the trainable parameters, and d_k is the dimension of $\mathbf{K}$. It can be described as a sequence of weighted sums and multiplications similar to the gating mechanism in LSTMs. In total, all operations are a product or a weighted sum of nodes, which can be treated with the same two rules for weighted sums and element-wise products described above to identify fibers. The attention mechanism is typically followed by dense layers that operate over features for each time point (token) independently.

Note that the gating in the LSTM and the multiplication in the transformer can be treated the same for the coloring algorithms, but the two structures do differ for the fiber compression rule as shown in Fig. 1.

The attention mechanism has both a feature dimension and a time (token) dimension. Here we have implemented a coloring algorithm to find fibration symmetries for the matrices $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ in the feature dimension, which can be propagated across all layers as before. The computation of symmetries for attention scores α requires defining symmetries across the token dimension while also incorporating the quadratic operation inherent in the feature dot product and will be part of future work.

Finally, Transformers and many other deep networks have "residual connections", which simply add the input to the output of a block of multiple layers. These "skip connections" combine entirely different colors (from different layers), in which case the addition operation simplifies to the same *And operation* as for the element-wise multiplication. If the input does not have large fibers, this *And operation* will tend to break up the fibers that have formed in the block.

5.2 Datasets and tasks used in empirical tests

5.2.1 MNIST digit classification using a multilayer perceptron

We trained a MLP with a 784-dimensional input layer, three hidden layers of 500 nodes each, and a 10-node output layer. The model was trained to classify the 10 digit classes of the MNIST dataset (28×28 binary images). The MNIST dataset contains 10 digit classes (0–9), with a nearly uniform distribution across categories. It is split into a training set of 60,000 images and a test set of 10,000 images. Training was performed using mini-batch gradient descent on Cross Entropy Loss with a batch size of 100 and Adam optimizer with a learning rate of 0.001. Ten independent training runs were conducted. The average training accuracy over 600 epochs is shown in Fig. 3a, where an epoch is a full pass through the entire training data.

5.2.2 ImageNet classification using a CNN

We trained a CNN with two convolutional layers (32 5x5-kernels and 64 3x3-filters) and two dense layers of 128 nodes each. The model was trained to classify images in the ImageNet dataset (32 x 32 RGB images). ImageNet dataset contains 1,000 classes, each of 700 images (600 for the training set, 100 for the test set). Training was performed using mini-batch gradient descent with the Cross Entropy Loss with a batch size of 100 and the SGD optimizer with a learning rate of 0.001 and momentum 0.9. Ten independent training runs were conducted. The average training accuracy over 300 epochs is shown in Fig. 3c.

5.2.3 Training Atari - Beam Rider with RL

We investigated a typical use of LSTMs in Reinforcement Learning (RL) by training an agent on the Atari Beam Rider game using the Proximal Policy Optimization (PPO) algorithm Schulman et al. [2017]. The agent is a deep neural network consisting of a sequential stack of three Conv2d layers, a linear layer, and a LSTM. From the LSTM output, the network branches into two heads: an Actor head, which transforms the output into a probability distribution over actions using a linear projection and Softmax activation; and a Critic head, which estimates the state value via a simple linear projection.

The agent interacts with the environment in a continuous cycle. The actor samples and executes an action; the environment responds with a reward and a new state; and the critic assesses the outcome to calculate a temporal difference error. This error guides the update of both networks, pushing the Actor to refine its policy toward more rewarding actions and helping the Critic improve the accuracy of its value predictions.

For training, we use the hyperparameters shown in Table S1.

Total Agent Time Steps	9216 K
Batch size	30720
Minibatch size	1024
Environment	Atari-Beam Rider
Num of Environment	65
BPTT Horizon	2
Learning rate	4e-4

Table S1: Training hyperparameters on RL.

5.3 Sequence-to-Sequence Transformers

In order to study the application of our fibration symmetry in Transformer models with attention mechanism, we turn to a sequence-to-sequence translation model for a German-to-English translation task using the Multi30k dataset Elliott et al. [2016]. This dataset consists of 31,014 sentences split into 29k training, 1k test, and 1k validation sets. For the model, we use a vanilla sequence-to-sequence transformer model as introduced in Vaswani et al. [2023] with Adam optimizer Kingma and Ba [2017] and early stopping regularization. The detailed specifications of the model and the training process are given in Table S2.

Number of encoder layers	3
Number of decoder layers	3
Number of attention heads	8
Source vocabulary size	10837
Target vocabulary size	19214
Batch size	128
Learning rate	1e-4
Adam Optimizer β_1	0.9
Adam Optimizer β_2	0.98
Adam Optimizer ϵ	1e-9
Early stopping patience	20

Table S2: Sequence-to-Sequence transformer model and training hyperparameters

We keep the dimension of the token embeddings (d_{emb}) and the size of the feedforward layer equal and use it to control the number of parameters in the model, similar to Kaplan et al. [2020]. Then, for each model size, we follow the procedure explained in Algorithm 2 to compress the trained model and iteratively retune. For each layer, we select a compression threshold $\varepsilon \in (0, 2)$ (as defined in 2.3) using a greedy algorithm explained in Section 5.4.

This Algorithm starts training with a vanilla Transformer model with early stopping. We then find a set of fiber precision $\{\varepsilon\}$ with the greedy Algorithm 3 described below. We then attempt to overcompress with somewhat larger thresholds (quenching rate $q > 1$) and regain performance

Algorithm 2 Compress-Retune Procedure for Transformer Model

1: Initialize model $\mathcal{M}$ and weights
2: Train $\mathcal{M}$ until convergence with early stopping
3: $L \leftarrow \mathcal{L}(\mathcal{M}, \text{test set})$ ▷ $\mathcal{L}$ is the loss function evaluated on the test set
4: $L_c \leftarrow L$
5: **while** $L_c \leq (1+\rho) \times L$ **do** ▷ Attempt over-compression with loss tolerance ρ
6: $\mathcal{M}_o \leftarrow \mathcal{M}$
7: $\{\varepsilon\} \leftarrow$ The set of compression thresholds for $\mathcal{M}$ using the greedy Algorithm 3
8: $\mathcal{M} \leftarrow$ Compressed $\mathcal{M}$ using $\{\varepsilon \times q\}$ ▷ over-compress with quenching rate $q > 1$
9: $\mathcal{M} \leftarrow$ Retuned $\mathcal{M}$ with with early stopping
10: $L_c \leftarrow \mathcal{L}(\mathcal{M}, \text{test set})$
11: **end while**
12: $\{\varepsilon\} \leftarrow$ The set of compression thresholds for $\mathcal{M}_o$ using the greedy Algorithm 3
13: $\mathcal{M}_o \leftarrow$ Compressed $\mathcal{M}_o$ using $\{\varepsilon\}$
14: **return** $\mathcal{M}_o$

by retuning with early stopping. If this regained performance is within tolerance ρ we attempt further over-compression. Otherwise, we compress with the last effective attempt and return this compressed network. We attempted q between 1.01-1.1 and found good performance with a fixed $q = 1.05$. Figure S5 shows this iterative process of compressing and retuning the model for a model with 50 million initial non-embedding parameters.

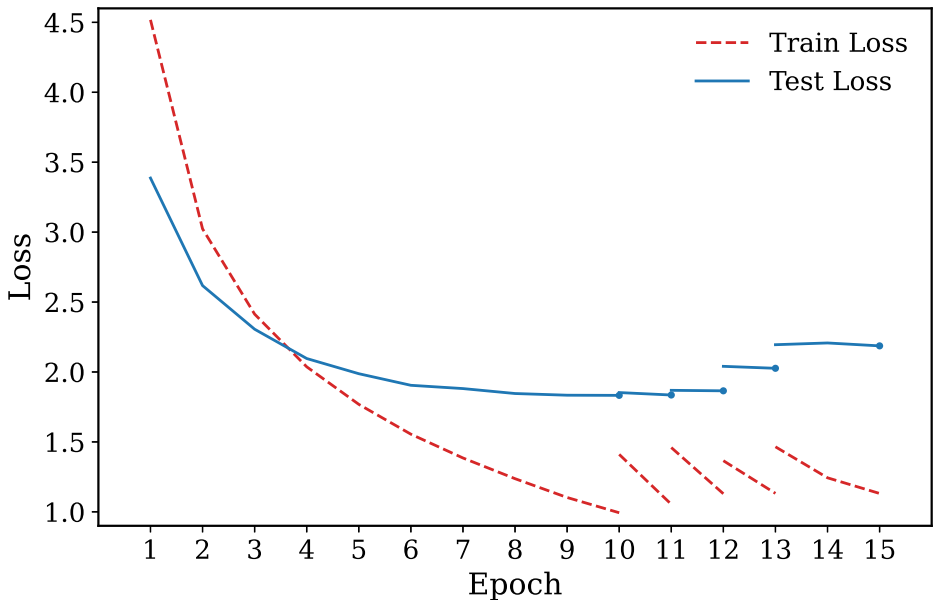

Figure S5: **Compress-retune procedure for a Transformer model.** Starting with a model with 50 million non-embedding parameters, on every iteration, we train the model with early stopping, then at each endpoint (shown with circles), we over-compress the model using larger compression thresholds, and attempt retuning to recover performance. Each blue point marks the model after retuning (line 6 in Algorithm 2. The numbers above the point, indicate the number of parameters of this model after one additional loss-preserving compression with thresholds $\{\varepsilon\}$ (from line 7 of Algorithm 2). These loss values and number of parameters are reported in the main text and Fig. 3f, so as to document the entire compression-retuning history.

5.4 The greedy algorithm for calculating fiber precision

To find a set of compression thresholds that minimizes the number of parameters while preserving the loss, the algorithm selects a different fibration tolerance threshold ε for the weight matrix of each layer in the Transformer model, forming the set $\{\varepsilon\}$. For this, we employ a greedy sequential strategy in which the compression threshold ε is maximized using binary search for each layer individually, so as not to increase the loss by more than a fraction τ (Here we used $\tau = 0.01$). The complete algorithm is detailed in Algorithm 3.

Algorithm 3 The greedy algorithm for optimal compression thresholds

Require: Transformer Model M, Test Dataset $\mathcal{D}$, Tolerance τ
Ensure: Set of compression thresholds $\{\varepsilon\}$

$L_{base} \leftarrow \mathcal{L}(\mathcal{M}, \mathcal{D})$
$L_{limit} \leftarrow L_{base} \cdot (1 + \tau)$
$\{\varepsilon\} \leftarrow \emptyset$
for each layer ℓ in $\mathcal{M}$ **do**
    $W \leftarrow$ GetWeights(ℓ)
    via Binary Search, find $\varepsilon^* = \max \varepsilon \, (\varepsilon \in (0, 2))$ **subject to**:
        Condition 1: $W' =$ Compress(W, ε)
        Condition 2: $\mathcal{L}(\mathcal{M}|_{W \leftarrow W'}, \mathcal{D}) \leq L_{limit}$
    SetWeights$(\mathcal{M}, l,$ Compress$(W, \varepsilon^*))$ ▷ Permanently update $\mathcal{M}$
    $\{\varepsilon\} \leftarrow \{\varepsilon\} \cup \varepsilon^*$
end for
return $\{\varepsilon\}$

Caption for Supplementary Movie S1. Formation of symmetries in a sample MLP training for MNIST.

File accuracy.gif: Accuracy of the network as a function of epochs along with the number of fibers divided by the total number of nodes (this includes trivial and non-trivial fibers, so it represents the possible compression).

File fiber.gif: Fibers indicated with the coloring of the nodes. White nodes are trivial single-node fibers. Connections are shown only for the strongest weights. Input units are not shown and are represented pictorially by the numbers learned from MNIST.