The Sign Code: The Hidden Binary Nature of Deep Networks

Niclas Göring[1*] **Shuofeng Zhang**[2] **Roi Holtzman**[1]
Yoonsoo Nam[3] **Ard A. Louis**[1]

[1]Rudolf Peierls Centre for Theoretical Physics, University of Oxford, Oxford, UK
[2]Meta
[3]Kim Jaechul Graduate School of AI, KAIST, Seoul, South Korea

Abstract

Every neuron in a trained network outputs a number that can be separated into its sign (positive or negative) and its magnitude (absolute size). We call the set of all neuron signs the *sign code*. This raises a key question: does classification depend more on the sign code or on the magnitudes? For overparameterized vision classifiers with sign-preserving activations, the answer is overwhelmingly the sign code. A k-nearest neighbor classifier operating only on this binary sign code achieves accuracy comparable to the full model across different architectures, activation functions, and datasets. Moreover, replacing all magnitudes in every layer with random noise while keeping the original signs barely changes residual network performance. In contrast, flipping just 20% of the signs reduces accuracy to near chance. Models generalize well when same-class inputs share similar sign codes. When this overlap is disrupted by a regularizer, training accuracy remains high, but test performance collapses. A margin argument helps explain when the sign code suffices. The natural unit of analysis for these networks may not be through their continuous weight space but instead through their discrete sign codes.

1 Introduction

ReLU networks admit a clean piecewise-linear decomposition: each input activates a subset of neurons, and within the resulting region the network acts as an affine map. This combinatorial structure has been a productive lens on deep learning, underpinning analyses of linear-region counts [Pascanu et al., 2013, Montúfar et al., 2014, Raghu et al., 2016, Serra et al., 2017, Hanin and Rolnick, 2019, Gamba et al., 2022], spline and max-affine views [Balestriero and Baraniuk, 2018], and gate- and path-based studies of expressivity and generalization [Neyshabur et al., 2015a,b, Fiat et al., 2018, Lakshminarayanan and Singh, 2020, Gonon et al., 2023, Lebeurrier et al., 2026]. But modern networks rarely use ReLU, while SiLU, GELU, and tanh have no piecewise-linear structure to exploit so that the tools built around ReLU do not transfer. We ask whether the combinatorial object underlying these ReLU analyses, which side of each neuron's hyperplane the input falls on, survives the move to smooth activations, and whether it remains the dominant carrier of classification signal.

For any sign-preserving activation φ (ReLU, SiLU, GELU, tanh), each neuron output decomposes exactly as $h_i(x) = S_i(x) \cdot M_i(x)$, where $S_i \in \{-1, +1\}$ is a binary sign and $M_i = |h_i|$ is a non-negative magnitude (Figure 1a,b). The sign identifies which combinatorial region of representation space the input occupies; the magnitude specifies where within that region. We call the collection $\{S_i(x)\}$ the *sign code* of the network, and ask whether it alone carries the essential information for classification. Across architectures, activations, and datasets, our experiments reveal a sharp

*Corresponding author: niclas.goring@physics.ox.ac.uk

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

asymmetry: the sign code is sufficient to recover full-model accuracy and causally dominant under intervention, while magnitudes are largely redundant (Figure 1c).

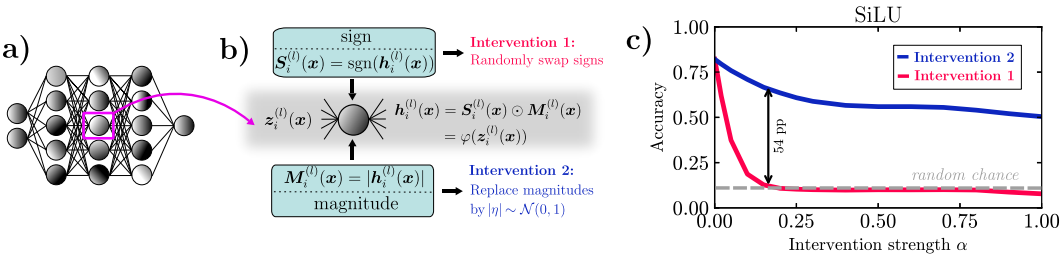


Figure 1: **a)** Every neuron in a feedforward network computes a pre-activation $z_j^{(\ell)}$, which the activation function maps to $h_j^{(\ell)} = \varphi(z_j^{(\ell)})$. **b)** For any sign-preserving activation, this output decomposes as $h_j = S_j \odot M_j$: a binary sign $S_j \in \{-1, +1\}$ encoding *which* side of the neuron's hyperplane the input falls on, and a non-negative magnitude $M_j = |h_j|$ encoding *how far*. We test the causal role of each component with two interventions: randomly flipping signs, or replacing magnitudes while enforcing the sign code. **c)** Flipping just 20% of sign bits collapses accuracy to near chance (red); replacing *every* magnitude in every layer preserves far more classification performance (blue), and is nearly lossless for residual networks. The forward computation is dominated by binary routing decisions rather than raw activation strengths.

Our **contributions** are:

(i) **The sign code is informationally sufficient for classification.** A k-nearest-neighbour classifier on the last-layer binary sign code, with no access to magnitudes, weights, or logits, matches full model accuracy in the overparameterized regime, across architectures, activations, and datasets. We give a sufficient certificate for this transition (Sections 3 and 7).

(ii) **Networks are far more robust to magnitude perturbations than to sign code perturbations.** We compare two interventions that isolate the sign and magnitude channels: destroying signs is catastrophic, whereas randomizing magnitudes while preserving the clean sign trajectory is much less harmful. We check this asymmetry across four activations and two architecture families (Section 5, Figure 4).

(iii) **Class-coherent sign geometry is tied to generalization.** Models generalize when same-class inputs share similar sign codes. A regularizer that actively repels within-class sign codes maintains high training accuracy while sharply degrading test accuracy, suggesting that class-coherent sign structure is a necessary ingredient for generalization in these models (Section 6).

Taken together, these results suggest that the natural unit of analysis for these networks may not be their continuous weight space but their *discrete sign code*.

1.1 Related Work

A large body of work studies ReLU networks through their input-space partitions, counting linear regions and bounding expressive gains from depth [Pascanu et al., 2013, Montúfar et al., 2014, Raghu et al., 2016, Serra et al., 2017, Hanin and Rolnick, 2019, Gamba et al., 2022] or interpreting deep networks as input-dependent affine maps indexed by activation patterns [Balestriero and Baraniuk, 2018]. Binary activation patterns have been examined as hash representations [He et al., 2021], learned via specialised objectives [Park et al., 2021], and used for verification [Geng et al., 2023]; on the theoretical side, path-norm, path-kernel, and gate-decoupling analyses elevate gates to a primary object of study [Neyshabur et al., 2015a,b, Lakshminarayanan and Singh, 2020, Fiat et al., 2018, Gonon et al., 2023, Lebeurrier et al., 2026]. The gate-decoupling line is closest to our work: Fiat et al. [2018] separates gating from linearity and Lakshminarayanan and Singh [2020] builds kernels from the active subnetwork. Both treat gates as a useful analytical device for ReLU; we make three stronger claims: informational sufficiency, causal dominance under intervention, and a link between sign geometry and generalization (Sections 3, 5 and 6). We establish them across all sign-preserving activations, including smooth ones with no piecewise-linear structure. Neural collapse [Papyan et al., 2020, Han et al., 2021] describes convergence of continuous features toward class means; our results

hold outside this regime (see Section B.2 for a thorough comparison). A comprehensive discussion of related work is given in Appendix A.

2 Background: The Sign–Magnitude Decomposition

2.1 Setup

We consider an L-layer feedforward network $f\colon \mathcal{X} \to \mathbb{R}^K$ (FFNN) with hidden representations $\mathbf{h}^{(0)}(x) = x$ and, for $\ell = 1, \dots, L$,

$$\mathbf{z}^{(\ell)} = W^{(\ell)}\,\mathbf{h}^{(\ell-1)} + \mathbf{b}^{(\ell)}\,, \qquad \mathbf{h}^{(\ell)} = \varphi\big(\mathbf{z}^{(\ell)}\big)\,, \tag{1}$$

where $W^{(\ell)} \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$, $\mathbf{b}^{(\ell)} \in \mathbb{R}^{d_\ell}$, and φ is a coordinatewise activation function. The output layer produces logits $f(x) = W^{(L+1)}\mathbf{h}^{(L)} + \mathbf{b}^{(L+1)}$.

Each neuron j in layer ℓ computes a pre-activation $z_j^{(\ell)} = \mathbf{w}_j^\top \mathbf{h}^{(\ell-1)} + b_j$. The zero set $\{z_j^{(\ell)} = 0\}$ is a hyperplane in the representation space of the preceding layer. A layer of d_ℓ neurons therefore partitions its input space by d_ℓ hyperplanes, assigning each input a binary code, one bit per neuron, that records which side of every hyperplane it falls on. The question is whether the activation function preserves this partition.

2.2 Sign-preserving activations

Definition 2.1 (Sign-preserving activation)**.** An activation $\varphi\colon \mathbb{R} \to \mathbb{R}$ is **sign-preserving** if $\operatorname{sgn}\big(\varphi(z)\big) = \operatorname{sgn}(z)$ for every $z \neq 0$, where $\operatorname{sgn}(t) = +1$ for $t > 0$ and -1 otherwise.

ReLU, SiLU, GELU, and tanh are all sign-preserving which will be the central point of study. A ReLU neuron zeroes out the negative half-space entirely, producing the sharpest possible partition; smooth activations such as SiLU and GELU attenuate rather than annihilate the negative side, but the sign of the output still mirrors the sign of the pre-activation. We discuss non sign-preserving activations such as softplus in Appendix C.

2.3 The decomposition

Every sign-preserving activation allows us to split each neuron's output into two factors: a binary routing decision and a continuous magnitude.

Definition 2.2 (Sign–magnitude decomposition)**.** Let φ be sign-preserving and let $h_j^{(\ell)}(\boldsymbol{x}) = \varphi\big(z_j^{(\ell)}(\boldsymbol{x})\big)$ be the output of neuron j at layer ℓ. The **sign** and **magnitude** of neuron j are

$$S_j^{(\ell)}(\boldsymbol{x}) \;=\; \operatorname{sgn}\big(h_j^{(\ell)}(\boldsymbol{x})\big) \;\in\; \{-1, +1\}\,, \qquad M_j^{(\ell)}(\boldsymbol{x}) \;=\; \big|h_j^{(\ell)}(\boldsymbol{x})\big| \;\in\; \mathbb{R}_{\geq 0}\,, \tag{2}$$

so that the neuron's output is

$$h_j^{(\ell)}(\boldsymbol{x}) \;=\; S_j^{(\ell)}(\boldsymbol{x}) \;\cdot\; M_j^{(\ell)}(\boldsymbol{x})\,. \tag{3}$$

In vector form, the full hidden representation decomposes as $\mathbf{h}^{(\ell)} = \mathbf{S}^{(\ell)} \odot \mathbf{M}^{(\ell)}$. The **sign code** of the network is the concatenation of all per-layer sign vectors:

$$\mathbf{S}(\boldsymbol{x}) = \big(\mathbf{S}^{(1)}(\boldsymbol{x}),\; \dots,\; \mathbf{S}^{(L)}(\boldsymbol{x})\big) \;\in\; \{-1, +1\}^D\,, \qquad D = \textstyle\sum_{\ell=1}^{L} d_\ell\,. \tag{4}$$

Because φ is sign-preserving, $S_j^{(\ell)} = \operatorname{sgn}(z_j^{(\ell)})$: the sign code is determined entirely by the pre-activation signs, so at the same pre-activations every sign-preserving activation produces the same code. Intuitively, $\mathbf{S}(x)$ records *which* combinatorial region of representation space the input occupies, which side of every learned hyperplane it falls on, while $\mathbf{M}(x)$ records *where* within that region. The sign code recovers the classical ReLU gate pattern $\mathbf{g}^{(\ell)} = \mathbf{1}[z^{(\ell)} > 0]$ under the relabelling $+1 \leftrightarrow$ active, $-1 \leftrightarrow$ inactive. The sign–magnitude decomposition for smooth activations (SiLU, GELU, tanh) captures the same combinatorial structure without requiring exact piecewise-affine behaviour. We discuss the natural generalization of this decomposition to convolutional layers as well as residual blocks in Section B.1.

3 Experiment: Sign Codes Are Informationally Sufficient

The sign–magnitude decomposition (Definition 2.2) splits every hidden representation into a discrete sign code and a continuous magnitude. We begin with a natural question:

Question 1: Does the sign code alone, without any magnitude information, already contain enough structure to predict class labels?

We test this by building the simplest possible classifier that operates *only* on the binary sign code of a single hidden layer, and comparing its accuracy to that of the full network.

3.1 Experimental setup: Sign-kNN

For each trained model we extract the sign code $\mathbf{S}^{(L)}(x) \in \{-1, +1\}^{d_L}$ of the *last hidden layer only*: $\mathrm{sgn}\big(\mathbf{h}^{(L)}(x)\big)$ for FFNNs, the pooled branch and output codes of the last `BasicBlock` for ResNets, and the output code of the final transformer block at the `[CLS]` token and the first patch token for ViTs. This is a strict subset of the full-network sign code (4). Full extraction details are given in Appendix D. We equip the sign code space with the normalised Hamming distance

$$d_H\big(\mathbf{S}^{(L)}(x),\, \mathbf{S}^{(L)}(x')\big) \;=\; \frac{1}{d_L} \sum_{j=1}^{d_L} \mathbf{1}\big[S_j^{(L)}(x) \neq S_j^{(L)}(x')\big]\,, \tag{5}$$

and classify each test point by majority vote among its k-nearest training-set neighbours in this distance. The resulting classifier, sign-kNN$_k$, uses only the binary sign code of a single layer. It has no access to activation magnitudes, to the sign codes of any earlier layer, or to the network's weights or logits. To quantify how well the sign code captures the network's classification, we define the **sign gap**:

$$\Delta_{\mathrm{sgn}} \;=\; \big|\, \mathrm{err}_{\mathrm{SignKNN}} - \mathrm{err}_{\mathrm{model}} \,\big|\,, \tag{6}$$

the absolute difference between the sign-kNN test error and the full model's test error. When $\Delta_{\mathrm{sgn}} \approx 0$, the binary sign code is sufficient to match the network's test error; when Δ_{sgn} is large, information outside the last-layer sign code remains relevant to the model's predictions.

3.2 Results: The sign gap across activation functions and architectures

Figure 2a) displays Δ_{sgn} as a function of network depth and width across all three architectures (FFNN, ResNet, ViT; SiLU activation, CIFAR-10). See Appendix E for a description of the exact architectures and the general training setup. For shallow or narrow networks the sign gap is substantial: the sign code loses information that the full network retains. As depth and width increase, Δ_{sgn} shrinks and becomes negligible (< 0.03) once both dimensions are sufficiently large. Sign-code sufficiency is not an unconditional property of the architecture; it is an *emergent* property of overparameterized networks. There is a clear phase boundary in the depth–width plane beyond which the last-layer sign code matches the full network's test performance. Figure 2b) plots sign-kNN against full model test accuracy for overparameterized networks (depth 6, width 1024 FFNN, standard ResNet-18 and depth 6, 256 embedding dimension, 8 heads ViT). We train 540 models spanning three architectures, four sign-preserving activations, three datasets (MNIST, CIFAR-10, CIFAR-100), 5 dataset sizes and 3 seeds. In the overparameterized regime, points cluster tightly around the diagonal with no systematic dependence on architecture, activation, or dataset. The experiment is conservative: it uses only the last layer's sign code, a small fraction of the full routing record, yet this already matches model performance across all four activations, including smooth ones (SiLU, GELU, tanh) with no piecewise-affine structure (thus generalizes the observations in [He et al., 2021]). This is not a consequence of neural collapse: our models are far from the collapsed regime (NC1 > 1, NCM accuracy well below model accuracy, see Section B.2).

Observation 1: Last-layer sign codes become informationally sufficient in the overparameterized regime. The last-layer sign code empirically suffices to match the model's test error; the sign gap Δ_{sgn} vanishes with depth and width.

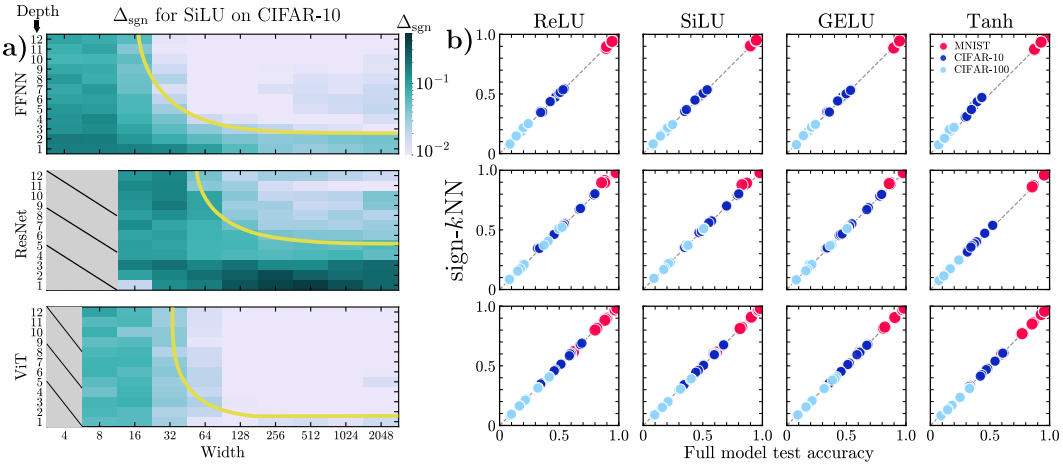


Figure 2: **a)** The sign gap Δ_{sgn} (Eq. 6) as a function of network depth and width for three architectures (FFNN, ResNet, ViT) with SiLU on CIFAR-10. The gap shrinks with increasing depth and width, with a clear phase boundary (yellow lines) beyond which $\Delta_{\text{sgn}} < 0.03$. **b)** Sign-$k$NN accuracy versus full model test accuracy across standard architectures (rows: FFNN, ResNet-18, ViT), activations (columns: ReLU, SiLU, GELU, Tanh), and datasets (colours: MNIST, CIFAR-10, CIFAR-100). The dashed line is $y{=}x$. Training and model details for all figures are in Appendix E.

4 Experiment: The Sign Code Tracks the Network's Scaling Law

Section 3 compared the code with the network by accuracy only. A network's test loss, however, falls with the amount of training data at a rate that characterises the network: its scaling law. We therefore ask:

Question 2: Does the sign code also carry the network's test loss, and does it improve with more data at the same rate as the network?

Comparing losses needs probabilities, not votes. Sign-kNN is the hard version of a soft nearest-codeword posterior, in which every training code votes for its label with a weight that decays with its Hamming distance,

$$P_\beta(y \mid \mathbf{s}) \;\propto\; \sum_i e^{-\beta\, d_H(\mathbf{s},\mathbf{s}_i)}\, \mathbf{1}[y_i = y]. \tag{7}$$

We fit the temperature β, and the temperature of the network's logits, on one half of the test set and report the calibrated test negative log-likelihood (NLL) on the other half. Each architecture is trained on growing subsets of its training set (data ladders from 2k to 65k images, two seeds): two MLPs, a CNN and three ResNet-18s on CIFAR-10 and SVHN, and a small ViT on CIFAR-10; five further ViTs are trained on all of CIFAR-10 (Section F).

The code's loss is the network's loss. In all 89 networks, the calibrated test NLL of the code lies within 15% of the network's, and it falls with data along the same curve (Figure 3a). Not every binary code does this: the code of the same network at initialisation, and the pixels, have about twice the network's loss.

The code improves at the network's rate. We fit the exponent α of the power law with which test error falls with the training-set size D, for the network and for sign-kNN on the network's own code. Across eight MLPs, two CNNs, three ResNet-18s and a ViT the two exponents agree to within 15%. Most of this rate comes from the code getting better, not from the bank of training codes growing: with the code of the network trained on all the data held fixed, a growing bank yields only 2–10% of the exponent in the ResNet-18s and the ViT.

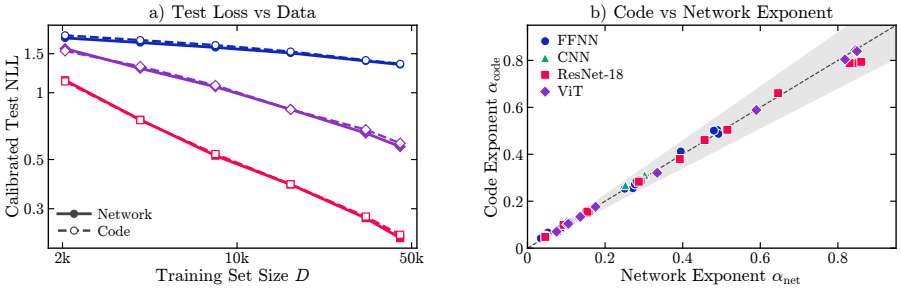


Figure 3: **a)** Calibrated test NLL against training-set size for three CIFAR-10 data ladders (an MLP, a ResNet-18 and the ViT; mean of two seeds): the network (solid) and the code's posterior of Equation (7) (dashed); colours as in b). **b)** Error exponent of sign-kNN on the network's own last-layer code against the network's exponent, for 14 data ladders and the 36 learning curves behind Figure 2b; dashed: equality, shaded: $\pm 15\%$.

Observation 2: The sign code tracks the network's scaling law. Read as a soft nearest-codeword posterior, the last-layer sign code matches the network's calibrated test loss, and its error falls with data at the network's rate, in MLPs, CNNs, ResNets and ViTs.

5 Experiment: Testing robustness to sign code and magnitude perturbations

The last sections showed that sign codes are informative: a nearest-neighbour classifier on the binary sign code of a single hidden layer tracks model accuracy. But informative does not mean causal. Both sign and magnitude are deterministic functions of the same weights, so either channel, or both, could carry the classification signal:

Question 3: How does the network respond to distinct perturbations: one that perturbs the discrete sign code, and one that perturbs only the continuous magnitudes?

If the sign code is merely a by-product and the real computation flows through magnitudes, then corrupting signs should be tolerable and corrupting magnitudes catastrophic. If the sign code is the dominant computational channel, the pattern should be the reverse.

5.1 Experimental setup: Two interventions

Recall that at every hidden layer ℓ, the representation decomposes as $\mathbf{h}^{(\ell)} = \mathbf{S}^{(\ell)} \odot \mathbf{M}^{(\ell)}$ (Definition 2.2). Both interventions select neurons independently with probability $\alpha \in [0, 1]$ at every hidden layer, but they are not meant to be symmetric natural forward passes. They are channel-isolating tests: one destroys routing while preserving magnitudes, and the other destroys magnitude information while enforcing the clean routing trajectory. The interventions differ in *which* channel is corrupted (Figure 4, right) and in the projection needed to keep that corruption isolated.

Intervention 1: destroy routing. Each selected neuron has its pre-activation negated, $h_j \mapsto \varphi(-z_j)$, which flips its sign. The corrupted representation propagates freely through all downstream layers, potentially altering their sign codes and magnitudes.

Intervention 2: destroy magnitudes, preserve routing. We first cache the clean sign code $\mathbf{S}^{(\ell)}(x)$ at every layer via an unmodified forward pass. Each selected neuron then has its magnitude replaced by $|\eta|$, $\eta \sim \mathcal{N}(0, 1)$, a class-agnostic random draw carrying zero information about the input. The replacement is *projected*: at every downstream layer, the cached clean sign code is enforced and only magnitudes are recomputed. This is a causal-mediation test: it asks what remains of the computation when the sign trajectory is held fixed, not whether the network could recover that trajectory after magnitude corruption. At $\alpha = 1$, every magnitude in every layer is a random number, yet the sign code is perfectly preserved.

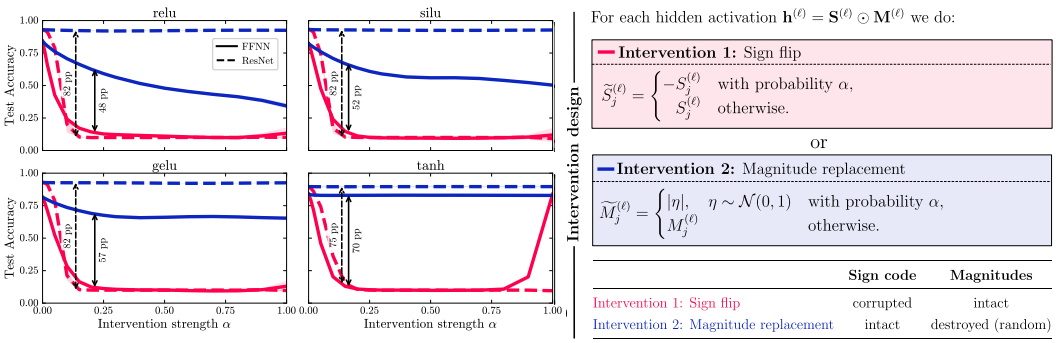


Figure 4: **Left (2×2):** Accuracy under channel destruction. Red: sign flip (sign code routing destroyed; the flipped unit outputs $\varphi(-z_j)$, so its magnitude is intact only for tanh). Blue: magnitude replacement with random $|\mathcal{N}(0,1)|$ draws (magnitudes destroyed, sign code routing intact). Solid: FFNN; dashed: ResNet; mean ± 1 s.d. over 3 seeds, each averaging 5 draws. **Right:** Intervention formulas and summary of what each intervention removes. Routing destruction collapses accuracy to near chance by $\alpha = 0.2$ across all conditions; projected replacement of every magnitude preserves far more classification performance, and is nearly lossless for residual networks.

5.2 Results of the robustness tests

We sweep $\alpha \in [0, 1]$ for four activations (ReLU, SiLU, GELU, tanh) and two architectures (FFNN on SVHN, ResNet-18 on CIFAR-10), with three seeds per network, each averaging five independent mask draws. Flipping just 20% of sign bits (α=0.2) collapses accuracy to near chance ($\sim$10%) across all eight conditions; the ResNet's skip connections do not compensate (Figure 4). At α=1, the ResNet maintains $\sim$90% accuracy across all four activations: every magnitude in every layer is an independent random number, yet the network classifies as well as with correct magnitudes. The FFNN retains near-full accuracy for tanh and degrades substantially for the rest, but remains far above the corresponding sign-flip failures.

The two interventions are deliberately asymmetric. Intervention 2 asks: *does the network need any magnitude information beyond what the sign code encodes?* Answering this requires isolating the magnitude channel, if corrupted magnitudes cascaded through downstream layers, they would induce secondary sign flips, corrupting both channels simultaneously. Projected replacement caches the clean sign code and enforces it at every layer, holding routing fixed while destroying magnitude content. Intervention 1 needs no analogous projection: it changes the sign of every selected pre-activation and keeps its size $|z_j|$. Under Intervention 2 the FFNNs lose 49.8 (ReLU), 31.9 (SiLU), 16.1 (GELU) and 1.0 (tanh) points. For tanh FFNNs, flipping every unit ($\alpha = 1$) is the odd symmetry $h \mapsto -h$ applied layer by layer, hence the recovery at $\alpha = 1$. Because the clean signs are enforced at every site, the ResNet's output at $\alpha = 1$ is a readout of its clean last-layer code. In Section G we also test a matched intervention, which removes either channel by replacing it with its conditional expectation while keeping one bit per unit. There the sign is the more important channel in ResNets and ViTs.

Observation 3: Networks are far more sensitive to sign code perturbations than to magnitude perturbations. Projected replacement of every magnitude preserves far more classification performance than sign corruption; flipping $\sim$20% of signs destroys the forward pass.

6 Experiment: Testing class-coherent sign geometry

Sections 3 to 5 showed that overparameterized networks can classify through sign code, and that corrupting the sign trajectory is far more destructive than corrupting magnitudes. But computation through signs does not by itself explain generalization. A network could classify through signs and still assign each example an idiosyncratic, non-transferable sign code, hence:

Question 4: What sign code structure supports generalization?

We measure this structure with a train-only statistic of the sign code and test it with a regularizer that destroys it.

6.1 Experimental setup: Varying class neighbourhood coherence through a regularizer $\mathcal{L}_{\text{rep}}$

Measuring sign code class structure. Let $T_{t,d}$, $d = 1, \ldots, k$, be k displacements of an image by t pixels, let h_{15} be the 15-nearest-code plurality reader of the labelled training bank on the concatenated signs $\mathbf{S}$ of all hidden layers, and let t^* be the displacement whose mean pixel change equals the leave-one-out nearest-image distance of the training set. We define

$$Q = \operatorname*{Interp}_{t=t^*} \frac{1}{k|I|} \sum_{i \in I} \sum_{d=1}^{k} \mathbf{1}\big[h_{15}\big(\mathbf{S}(T_{t,d}\, \boldsymbol{x}_i)\big) = y_i\big], \tag{8}$$

the class vote of the training codes one nearest-neighbour spacing away from a training image, interpolated between integer displacements. Here we choose $k = 4$, the circular translations along the two image axes ($t^* \simeq 2.2$ px on SVHN, 2.3 px on CIFAR-10). Q uses training images and training labels only.

Causal test: within-class sign repulsion. For each neuron we form a continuous proxy of its binary routing decision, concatenate across layers, and ℓ_2-normalise. The repulsion loss $\mathcal{L}_{\text{rep}}$ penalises routing overlap between same-class pairs:

$$\mathcal{L}_{\text{rep}} = \frac{1}{|\mathcal{P}|} \sum_{(i,j) \in \mathcal{P}} \big(\mathbf{S}(\boldsymbol{x}_i)^\top \mathbf{S}(\boldsymbol{x}_j)\big)^2, \qquad \mathcal{P} = \{(i,j) : i \neq j,\ y_i = y_j\}. \tag{9}$$

For ReLU, SiLU, and GELU we use a sigmoid gate proxy $S_j = \sigma(\boldsymbol{z}_j/\tau) \in (0, 1)$, which targets the active set that dominates the forward pass; for tanh, where positive and negative neurons contribute equally, we instead use a tanh sign proxy $S_j = \tanh(\boldsymbol{z}_j/\tau) \in (-1, 1)$. The choice of proxy is discussed in Appendix K.2.

6.2 Results

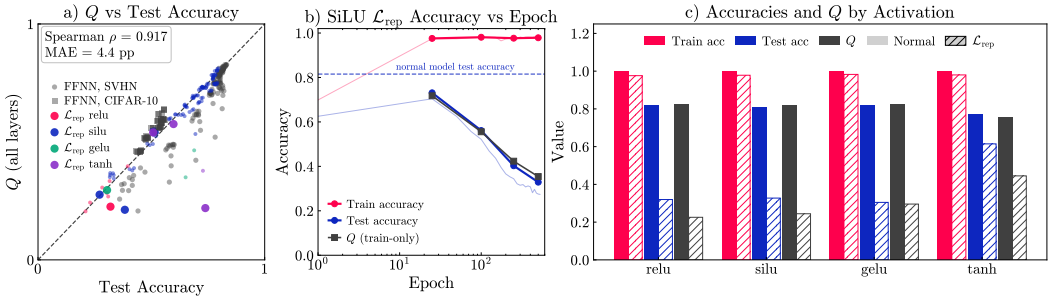


Figure 5: **a)** Q (Equation (8)) versus test accuracy across models (grey: FFNN on SVHN and CIFAR-10, widths 4–2048, depths 1–12) and models trained with $\mathcal{L}_{\text{rep}}$ or one of its components (coloured by activation). Q tracks generalization along the identity line (Spearman $\rho = 0.92$, mean absolute error 4 pp); the exceptions, narrow codes and the magnitude-carried tanh runs, sit below it. **b)** Training dynamics under $\mathcal{L}_{\text{rep}}$ (SiLU, width 1024, depth 8, SVHN): train accuracy stays near 98% while test accuracy collapses, and Q measured on the saved checkpoints follows it. Dashed: test accuracy of the normally trained baseline. **c)** Summary across activations (depth 8, width 1024, SVHN): repulsion degrades test accuracy and Q together at unchanged training accuracy; tanh is partially robust ($\delta \approx 0$; Appendix K).

Figure 5a) plots Q against test accuracy for a range of models. The vote one spacing away from the training points tracks generalization across widths, data sets and repulsion strengths (Spearman $\rho = 0.92$); the $\mathcal{L}_{\text{rep}}$-trained models land in the low-Q, low-accuracy corner *on the same line*, the only exceptions being codes with too few bits (Section 7) and the magnitude-carried tanh runs. Panel b) shows SiLU dynamics: train accuracy stays near 98% while test accuracy collapses, and Q collapses

with it. Panel c) summarises across activations: repulsion degrades test accuracy and Q together while training accuracy stays above 97%; tanh is partially robust ($\delta \approx 0$, Appendix K). What the regularizer destroys is the local class structure of the code: same-class neighbours no longer share their code, the network still fits the training set, and generalization collapses. This combination of high training and degraded test accuracy, read off the sign code by a train-only statistic, provides causal evidence that within-class sign-code coherence plays a direct role in generalization for the models studied.

Observation 4: Class-coherent sign geometry at the data's resolution tracks generalization. The vote of the training codes one nearest-neighbour spacing away from a training image tracks the network's test accuracy. A regularizer that repels within-class sign codes destroys this local coherence, preserves training accuracy and collapses test accuracy.

7 Theory: Overparameterization Makes Neural Networks Binary

Section 3 showed that sign-kNN on the last-layer code matches the network once that layer is wide and deep. We now explain why. Each of the d_L units of the last hidden layer is a hyperplane through the representation $\mathbf{z}^{(L)}$, and its sign records on which side an input falls. The normalised Hamming distance (Equation (5)), which we write $d_H(x, x')$, is therefore the fraction of hyperplanes that separate two inputs: a measurement of how far apart they lie in the layer [Charikar, 2002, He et al., 2021]. Sign-kNN predicts the most common class among a test point's k nearest training codes, so it is correct whenever more than half of them share the test point's class. Two properties guarantee this. **(a) A class margin:** in the layer, training points of the test point's class lie nearer to it than points of other classes. **(b) Resolution:** the code measures distances with an error smaller than this margin. In trained networks, depth provides (a) and width provides (b).

To make this precise, fix the trained representation, the training set and a test point x of class y, and let $p(x, x')$ be the probability that a hyperplane drawn from a fixed distribution separates x and x'. For hyperplanes through the origin with isotropic random normals, $p = \theta/\pi$, with θ the angle between $\mathbf{z}^{(L)}(x)$ and $\mathbf{z}^{(L)}(x')$ [Goemans and Williamson, 1995, Charikar, 2002]. With $j = \lfloor k/2 \rfloor + 1$, let p_{same} be the j-th smallest value of $p(x, \cdot)$ over training points of class y, and p_{other} the $(k - j + 1)$-th smallest over the other training points. The margin is $g(x) = p_{\text{other}} - p_{\text{same}}$; for $k = 15$ it compares the 8th-nearest training point of each kind.

Proposition 7.1 (Margin and resolution)**.** *Suppose $g(x) > 0$. **(i)** If $|d_H(x, x_i) - p(x, x_i)| < g(x)/2$ for every training point x_i, sign-kNN is correct at x. **(ii)** If the code's n hyperplanes are drawn independently from the distribution that defines p, then (i) holds with probability at least $1 - 2m\, e^{-ng(x)^2/2}$, where m is the number of training points.*

Under (i), at least j training points of class y but at most $k - j$ of other classes lie nearer than $p_{\text{same}} + g/2$, so class y holds a majority of the k nearest codes; (ii) adds Hoeffding's inequality [Hoeffding, 1963] and a union bound (Section H, which also gives a sharper form in which only training points near the margin count). The margin is the signal and the code's error is the noise. The condition is sufficient but not necessary, since a plurality can also win, and it is conservative. We use it to name the two quantities, and test below what it predicts for width and depth.

Width provides resolution (b). The units cut the layer into cells, and the code of an input names its cell. With few units the cells are large: training points at quite different distances from a test point lie behind the same number of cuts, and the code cannot tell its near neighbours from its far ones. Each added unit adds a cut, so the cells shrink and the number of cuts between two inputs follows their distance in the layer more closely. *Proved for random hyperplanes:* each one separates x and x' independently with probability $p(x, x')$, so d_H averages n coin flips and its error shrinks as $1/\sqrt{n}$; n hyperplanes then classify a test point correctly with probability at least $1 - \delta$ once its margin exceeds $\sqrt{2\ln(2m/\delta)/n}$ (Proposition 7.1(ii)). *Measured for trained networks:* a layer's own units cut it like random hyperplanes with matched activity, with sign-kNN errors within two points of theirs in 65 of 70 networks of width at least 32 (a retrained copy of the grid of Figure 2a). Adding hyperplanes to a fixed trained layer lowers the error until the cells are fine enough, and then it levels off: at the network's own width of 1024, going to 8,192 hyperplanes gains at most 2.3 points, and less than

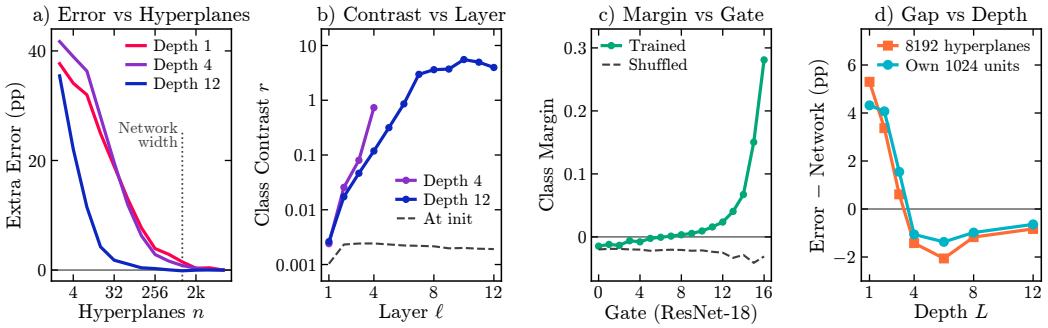


Figure 6: **a)** Extra sign-kNN error (k=15) when one trained layer (CIFAR-10 SiLU FFNN, width 1024) is read by n activity-matched random hyperplanes instead of 8,192; dotted: the network's width. **b)** Class contrast $r = S_B/S_W$ of the code by layer (SVHN SiLU FFNNs, width 1024); dashed: at initialisation. **c)** Class margin of the code by gate (the CIFAR-10 ResNet-18s of Section 5); dashed: shuffled weights. **d)** Sign-kNN error minus network error by depth (networks of a), for the network's own code and for 8,192 random hyperplanes. Details in Section I.

one from depth 4 on (Figure 6a). In the ResNet-18s of Section 5 the cells are already fine enough: binarisation barely changes the classes among each test point's neighbours (Section I.2).

Depth provides the margin (a). Depth separates the classes. Layer by layer, the codes of each class cluster more tightly relative to the distances between classes, so a test point's nearest training codes increasingly share its class. In MLPs on SVHN, the class contrast of the code, the spread between class means over the spread within classes, grows by more than three orders of magnitude over twelve layers, while at initialisation it stays flat (Figure 6b). In ResNet-18s on CIFAR-10, a test image's code comes to lie closer to its own class than to any other, most strongly in the last blocks, and not at all when the trained weights are shuffled (Figure 6c). This, not resolution, is what shallow networks lack: even 8,192 hyperplanes leave sign-kNN 4.6 points behind shallow wide networks, whereas in deep networks the same reader beats the network by 1.3 points and the network's own units come within 0.5 points of it (Figure 6d).

8 Discussion

In overparameterized networks, classification depends far more on a discrete sign code with class-coherent sign geometry than on precise activation magnitudes. Preserving sign trajectories while randomizing magnitudes retains high accuracy, whereas small sign perturbations collapse performance catastrophically. This suggests that existing generalization theory may be missing an important granularity: the binary sign code quotients weight space into equivalence classes, and generalization bounds formulated directly over this discrete object would be invariant to magnitude redundancies that can make norm-based bounds vacuous. Section 7 explains when the last-layer code can be read: the classes must be separated in the layer, which depth provides, and the code must resolve that separation, which width provides. Width and depth are not guarantees, however. Both properties can break, above all in adversarial situations: a training objective can destroy the margin while keeping the training fit, as the repulsion loss of Section 6 does, and a worst-case choice of hyperplanes or of test points defeats any width. Nor does a readable code have to be the one the network computes with. Section J discusses these cases and the bounds that hold regardless.

Limitations. Our theory explains when the last-layer code can be read, not why training separates the classes; its proved part concerns random hyperplanes, and the comparison with trained units is empirical. We restrict attention to sign-preserving activations. Our empirical evidence comes from small-scale vision benchmarks (CIFAR, SVHN, MNIST) and classification tasks; whether the sign code mechanism extends to large-scale settings, language modelling, or regression is left to future work. The code for the paper can be found in `https://github.com/NiclasGoering/sign-code`.

Acknowledgments and Disclosure of Funding

This work was supported by the Laboratory for AI Security Research (LASR). The views expressed in this paper are those of the authors and do not necessarily reflect the position of LASR or His Majesty's Government. This work was supported by the InnoCORE program of the Ministry of Science and ICT(N10250156).

References

Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *International conference on machine learning*, pages 254–263. PMLR, 2018.

Randall Balestriero and Richard Baraniuk. A spline theory of deep networks. In *International Conference on Machine Learning*, 2018. URL https://api.semanticscholar.org/CorpusID:51882762.

Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. *Advances in neural information processing systems*, 30, 2017.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 380–388, 2002. doi: 10.1145/509907.509965.

Kamalika Chaudhuri and Sanjoy Dasgupta. Rates of convergence for nearest neighbor classification. In *Advances in Neural Information Processing Systems*, volume 27, 2014.

Lénaïc Chizat and Francis R. Bach. Implicit bias of gradient descent for wide two-layer neural networks trained with the logistic loss. In *Annual Conference Computational Learning Theory*, 2020. URL https://api.semanticscholar.org/CorpusID:211076120.

Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In *International Conference on Machine Learning*, 2017. URL https://api.semanticscholar.org/CorpusID:7636159.

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. *ArXiv*, abs/2010.11929, 2020. URL https://api.semanticscholar.org/CorpusID:225039882.

Gintare Karolina Dziugaite and Daniel M Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *arXiv preprint arXiv:1703.11008*, 2017.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2017.12.012. URL https://www.sciencedirect.com/science/article/pii/S0893608017302976. Special issue on deep reinforcement learning.

Jonathan Fiat, Eran Malach, and Shai Shalev-Shwartz. Decoupling gating from linearity. *ArXiv*, abs/1906.05032, 2018. URL https://api.semanticscholar.org/CorpusID:127096892.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv: Learning*, 2018. URL https://api.semanticscholar.org/CorpusID:53388625.

Matteo Gamba, Adrian Chmielewski-Anders, Josephine Sullivan, Hossein Azizpour, and Marten Bjorkman. Are all linear regions created equal? In *International Conference on Artificial Intelligence and Statistics*, 2022. URL https://api.semanticscholar.org/CorpusID:247084088.

Chuqin Geng, Nham Le, Xiaojie Xu, Zhaoyue Wang, Arie Gurfinkel, and Xujie Si. Towards reliable neural specifications. In *International Conference on Machine Learning*, pages 11196–11212. PMLR, 2023.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2011. URL https://api.semanticscholar.org/CorpusID:2239473.

Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995. doi: 10.1145/227683.227684.

Antoine Gonon, Nicolas Brisebarre, Elisa Riccietti, and Rémi Gribonval. A path-norm toolkit for modern networks: consequences, promises and challenges. *ArXiv*, abs/2310.01225, 2023. URL https://api.semanticscholar.org/CorpusID:263605441.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *arXiv: Computer Vision and Pattern Recognition*, 2015. URL https://api.semanticscholar.org/CorpusID:2134321.

X. Y. Han, Vardan Papyan, and David L. Donoho. Neural collapse under mse loss: Proximity to and dynamics on the central path. *ArXiv*, abs/2106.02073, 2021. URL https://api.semanticscholar.org/CorpusID:235352583.

Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. In *International Conference on Machine Learning*, 2019. URL https://api.semanticscholar.org/CorpusID:59291990.

Fengxiang He, Shiye Lei, Jianmin Ji, and Dacheng Tao. Neural networks behave as hash encoders: An empirical study. *arXiv preprint arXiv:2101.05490*, 2021.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015. URL https://api.semanticscholar.org/CorpusID:206594692.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv: Learning*, 2016. URL https://api.semanticscholar.org/CorpusID:125617073.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. doi: 10.1080/01621459.1963.10500830.

W. Ronny Huang, Zeyad Ali Sami Emam, Micah Goldblum, Liam H. Fowl, Justin K. Terry, Furong Huang, and Tom Goldstein. Understanding generalization through visualizations. In *ICBINB@NeurIPS*, 2019. URL https://api.semanticscholar.org/CorpusID:182952973.

Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJgIPJBFvH.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *ArXiv*, abs/1609.04836, 2016. URL https://api.semanticscholar.org/CorpusID:5834589.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey E. Hinton. Similarity of neural network representations revisited. *ArXiv*, abs/1905.00414, 2019. URL https://api.semanticscholar.org/CorpusID:141460329.

Chandrashekar Lakshminarayanan and Amit Singh. Neural path features and neural path kernel : Understanding the role of gates in deep learning. *ArXiv*, abs/2006.10529, 2020. URL https://api.semanticscholar.org/CorpusID:219792921.

Arthur Lebeurrier, Titouan Vayer, and Rémi Gribonval. Path-conditioned training: a principled way to rescale relu neural networks. *ArXiv*, abs/2602.19799, 2026. URL https://api.semanticscholar.org/CorpusID:285973448.

Kaifeng Lyu and Jian Li. Gradient descent maximizes the margin of homogeneous neural networks. *ArXiv*, abs/1906.05890, 2019. URL https://api.semanticscholar.org/CorpusID:189898036.

Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Neural Information Processing Systems*, 2014. URL https://api.semanticscholar.org/CorpusID:5941770.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021.

Behnam Neyshabur, Ruslan Salakhutdinov, and Nathan Srebro. Path-sgd: Path-normalized optimization in deep neural networks. *ArXiv*, abs/1506.02617, 2015a. URL https://api.semanticscholar.org/CorpusID:2101905.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *Conference on learning theory*, pages 1376–1401. PMLR, 2015b.

Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. Towards understanding the role of over-parametrization in generalization of neural networks. *ArXiv*, abs/1805.12076, 2018. URL https://api.semanticscholar.org/CorpusID:44130076.

Vardan Papyan, Xuemei Han, and David L. Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. *Proceedings of the National Academy of Sciences of the United States of America*, 117:24652 – 24663, 2020. URL https://api.semanticscholar.org/CorpusID:221172897.

Yookoon Park, Sangho Lee, Gunhee Kim, and David Blei. Unsupervised representation learning via neural activation coding. In *International Conference on Machine Learning*, pages 8391–8400. PMLR, 2021.

Razvan Pascanu, Guido Montúfar, and Yoshua Bengio. On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv: Learning*, 2013. URL https://api.semanticscholar.org/CorpusID:9254343.

Maithra Raghu, Ben Poole, Jon M. Kleinberg, Surya Ganguli, and Jascha Narain Sohl-Dickstein. On the expressive power of deep neural networks. In *International Conference on Machine Learning*, 2016. URL https://api.semanticscholar.org/CorpusID:2838204.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *ArXiv*, abs/1710.05941, 2018. URL https://api.semanticscholar.org/CorpusID:10919244.

Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. *ArXiv*, abs/1711.02114, 2017. URL https://api.semanticscholar.org/CorpusID:34019680.

Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data. *J. Mach. Learn. Res.*, 19(1):2822–2878, January 2018. ISSN 1532-4435.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017.

A Related work

A central motivation for this paper is the generalization puzzle in overparameterized deep learning. Modern networks can interpolate random labels while still generalizing on natural data, forcing a re-examination of classical capacity arguments [Zhang et al., 2017]. This led to a large literature on norm-, margin-, PAC-Bayes-, and compression-based explanations of generalization [Dziugaite and Roy, 2017, Neyshabur et al., 2015b, Bartlett et al., 2017, Arora et al., 2018, Jiang et al., 2020], and to studies of the interpolation regime through double descent and overparameterization [Belkin et al., 2019, Nakkiran et al., 2021, Neyshabur et al., 2018]. A complementary line analyzes the implicit bias of optimization, showing convergence toward max-margin or otherwise structured solutions in linear, homogeneous, and wide-network settings [Soudry et al., 2018, Lyu and Li, 2019, Chizat and Bach, 2020]. Another influential debate ties generalization to sharpness or flatness of minima [Keskar et al., 2016, Dinh et al., 2017]. Our work is motivated by the same puzzle, but shifts the explanatory target from continuous weights to the discrete routing pattern induced by the network.

A second line studies neural networks through their activation patterns and the partitions they induce. Early work on rectifiers emphasized sparse activation and the practical advantages of non-saturating units [Glorot et al., 2011]. For ReLU and related piecewise-linear networks, several papers quantified the number of response or linear regions and the expressive gains from depth [Pascanu et al., 2013, Montúfar et al., 2014, Raghu et al., 2016, Serra et al., 2017]. Later work showed that the number of linear regions realized in practice can be far below worst-case counts [Hanin and Rolnick, 2019], and that linear regions are not equally important functionally [Gamba et al., 2022]. Closely related spline and max-affine views interpret deep networks as input-dependent affine maps indexed by activation patterns [Balestriero and Baraniuk, 2018]. Our sign code perspective is closest to this tradition, but differs in emphasis: the goal here is not primarily to count regions or characterize expressivity, but to argue that the binary code identifying a region can itself become the dominant carrier of class information.

Some aspects of binary activation patterns have been studied. It was shown in He et al. [2021] that the sign code induced by a trained classifier can behave as a hash representation. Park *et al.* introduces a specialized unsupervised objective that learns binary activation codes in order to increase encoder expressivity and improve downstream classification and retrieval Park et al. [2021]. Related work has shown that class-dependent partial sign codes can be extracted from trained ReLU networks and used to verify that the network assigns the same label to inputs consistent with those partial sign codes Geng et al. [2023].

This paper is especially close to the path- and gate-based literature. Path-SGD and path-norm analyses emphasized geometries and complexity measures that respect ReLU rescaling symmetries better than raw Euclidean weight space [Neyshabur et al., 2015a,b]. Lakshminarayanan and Singh introduced neural path features and the neural path kernel, explicitly encoding the active subnetwork selected by the gates [Lakshminarayanan and Singh, 2020]. Fiat et al. made the separation between gating and linear computation explicit by decoupling the two roles in the neuron [Fiat et al., 2018]. More recent work extended path-based analyses to modern DAG architectures and principled rescaling procedures [Gonon et al., 2023, Lebeurrier et al., 2026]. Among existing lines, this is the closest antecedent to our thesis: we likewise elevate gates from a bookkeeping device to a primary object of analysis.

Our empirical results also connect to work on representation geometry in trained classifiers. Visualization and representation-similarity studies show that networks that generalize organize their internal representations differently from networks that memorize [Huang et al., 2019, Kornblith et al., 2019]. Neural collapse sharpened this picture by showing that late-layer features can converge toward class means and nearest-class-mean decision rules during terminal training phases [Papyan et al., 2020, Han et al., 2021]. Our sign-kNN analyses are complementary to this literature: they replace continuous Euclidean geometry by discrete Hamming geometry on sign codes, and ask whether class structure is already recoverable before or even without full collapse of the continuous feature cloud.

Finally, because the present paper studies sign-preserving activations and multiple backbone families, it also interfaces with the activation-function and architecture literature. The experiments span residual networks and vision transformers [He et al., 2015, Dosovitskiy et al., 2020], and include GELU and SiLU/Swish-style smooth activations in addition to ReLU and tanh [Hendrycks and Gimpel, 2016, Ramachandran et al., 2018, Elfwing et al., 2018]. The redundancy and erasure-robustness story we observe is also adjacent to pruning, compression, and sparse-subnetwork results [Han et al., 2015,

Frankle and Carbin, 2018]: heavily overparameterized networks often contain substantial slack. The distinction is that pruning and compression typically operate in parameter space, whereas our thesis is about input-dependent routing structure. The claim is not merely that many weights are removable, but that overparameterization can induce a simpler, more redundant binary partition whose class geometry increasingly governs prediction.

B Additional material for the main paper

B.1 Extension of sign extraction

Convolutional blocks. For a convolutional output $\mathbf{h}^{(\ell)} \in \mathbb{R}^{C\times H\times W}$, Definition 2.2 applies entry-wise, yielding a sign code tensor $\mathbf{S}^{(\ell)} \in \{-1,+1\}^{C\times H\times W}$ that records both *which* channels and *where* in space each feature responds positively. We compress the spatial dimensions by average-pooling the activations on a $P\times P$ grid (P=2 throughout) and taking the sign of each pooled value, giving $C \cdot P^2$ bits per layer (Section D.4).

Residual blocks. A residual block adds a learned branch to a skip connection:

$$\mathbf{h}^{(\ell+1)} = \varphi\Big(\mathbf{h}^{(\ell)} + \mathcal{F}^{(\ell)}\big(\mathbf{h}^{(\ell)}\big)\Big)\,, \tag{10}$$

where $\mathcal{F}^{(\ell)}$ is a sub-network (typically two convolutional or fully-connected layers with their own activations). Two families of sign codes arise. The activations *inside* the branch $\mathcal{F}^{(\ell)}$ produce *internal* sign codes, computed exactly as in Definition 2.2. For a standard two-layer branch with intermediate activation, the internal code is $\mathbf{S}^{(\ell)}_{\text{int}} = \operatorname{sgn}\big(\varphi(W_1^{(\ell)}\mathbf{h}^{(\ell)} + \mathbf{b}_1^{(\ell)})\big)$. The *block-output* code $\mathbf{S}^{(\ell)}_{\text{out}} = \operatorname{sgn}(\mathbf{h}^{(\ell+1)})$ records the sign of the *sum* of skip and branch, and need not equal $\mathbf{S}^{(\ell)}$: it is precisely the sign flips introduced by the branch that allow each block to refine the combinatorial partition of the preceding layer. The full sign code of an L-block residual network concatenates both internal and output codes across all blocks:

$$\mathbf{S}(x) \;=\; \big(\,\mathbf{S}^{(1)}_{\text{int}},\, \mathbf{S}^{(1)}_{\text{out}},\, \ldots,\, \mathbf{S}^{(L)}_{\text{int}},\, \mathbf{S}^{(L)}_{\text{out}}\,\big) \;\in\; \{-1,+1\}^{D}\,. \tag{11}$$

Omitting either the internal or the output codes would lose information about the routing decisions the network makes.

B.2 Relationship to Neural Collapse

Neural collapse [Papyan et al., 2020, Han et al., 2021] describes a terminal-phase phenomenon in which (NC1) within-class feature variance vanishes relative to between-class variance, (NC2) class means converge to an equiangular tight frame, (NC3) the last-layer classifier aligns with the class means, and (NC4) a nearest-class-mean (NCM) classifier matches network accuracy. A natural question is whether sign code sufficiency is simply a consequence of neural collapse: if features have already collapsed onto class means, any reasonable representation—including the sign code—would trivially separate classes.

We measure all four NC metrics across the full FFNN width–depth grid (SiLU, SVHN, 80 training epochs) in Figure 7. The models are far from the collapsed regime. NC1 (S_W/S_B) remains above 1 in 106 of the 129 networks, indicating that within-class scatter still dominates between-class scatter; values below 10^{-1} would indicate meaningful collapse. NC3 (cosine alignment between the classifier weights and the class-mean directions) stays below 0.6 in 86% of the networks (median 0.43), far from the ideal value of 1. NC4 (NCM test accuracy) stays below the full model accuracy: in the 85 networks with test accuracy above 75% it has median 72% (at most 77%), against a median model accuracy of 80%. NC2 (ETF deviation) is near zero at large widths but this alone does not indicate collapse—it reflects only the angular structure of the class means, not the tightness of the feature clouds around them.

Despite this absence of collapse, sign-kNN on the last-layer code already matches the network in the wide and deep models of this grid: in most networks of width at least 128 and depth at least 3 it is within two points of the network's test accuracy (Section 3). Class structure is thus readable in discrete Hamming space before the continuous features converge toward their class means. Sign-code sufficiency therefore does not require terminal-phase collapse.

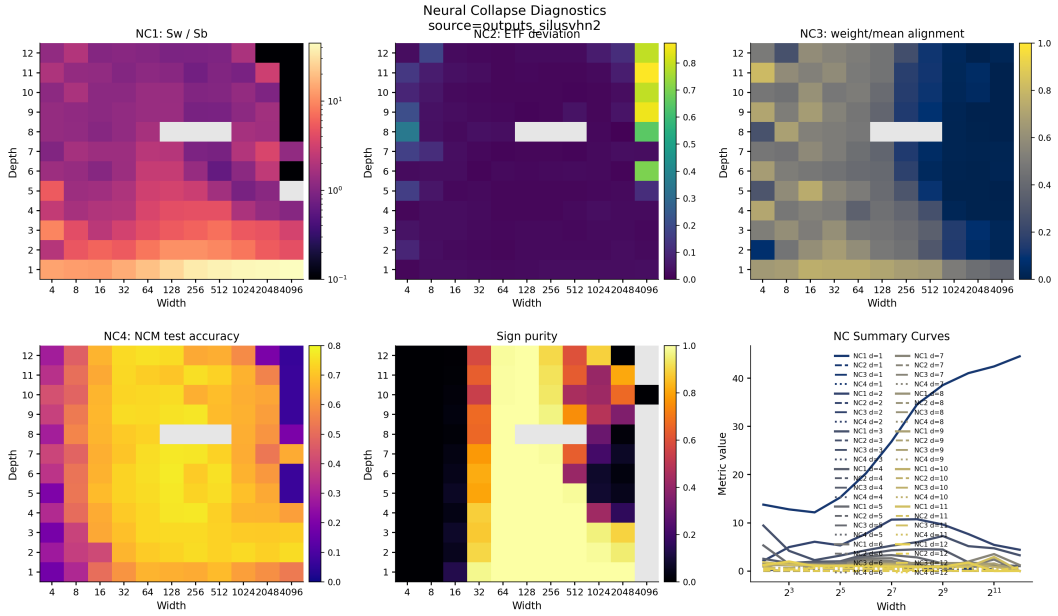


Figure 7: **Neural collapse diagnostics across the width–depth grid** (FFNN, SiLU, SVHN, 80 epochs). **Top row:** NC1 (S_W/S_B, log scale), NC2 (ETF deviation), NC3 (weight–mean cosine alignment). **Bottom row:** NC4 (nearest-class-mean test accuracy), sign purity, and NC summary curves across widths for selected depths. Sign purity is the fraction of training points whose exact last-layer code $\mathrm{sgn}(\mathbf{z}^{(L)})$ is shared only with points of the same class. It is 1 whenever every code is unique, as for nearly all training points at widths 64–512, so it measures class sorting only where codes collide. Width-4096 networks are shown only where the checkpoint was kept (depth 10). NC1 remains above 1 in 106 of 129 networks and NC3 below 0.6 in 86%, so the models have not undergone neural collapse. NCM test accuracy (NC4) stays below the full model accuracy (median 72% against 80% in the well-trained networks).

C Non-Sign-Preserving Activations

The sign-preserving property (Definition 2.1) covers the activations dominant in modern practice: ReLU, SiLU/Swish, GELU, and tanh are all sign-preserving and account for the vast majority of architectures in current use. The decomposition $h_j = S_j \cdot M_j$ with $S_j = \mathrm{sgn}(z_j)$ applies directly to all of them. We briefly discuss what happens outside this class.

Shifted-threshold activations (e.g. softplus). Softplus, $\varphi(z) = \log(1 + e^z)$, is strictly positive for all z and therefore not sign-preserving: $\mathrm{sgn}(\varphi(z)) = +1$ always. However, softplus is monotone, so it preserves the ordering of pre-activations. A natural generalization is to replace the zero threshold with the activation's symmetry axis. For softplus, $\varphi(0) = \log 2$, and the function is approximately linear for $z \gg 0$ and approximately flat for $z \ll 0$. Defining $S_j = \mathrm{sgn}(z_j)$ directly—bypassing the activation output—recovers a meaningful binary partition: it still records which side of the neuron's learned hyperplane the input falls on. The magnitude can then be defined relative to this threshold, e.g. $M_j = |\varphi(z_j) - \varphi(0)|$. We have not tested this variant experimentally, but the monotonicity of softplus ensures that the partition induced by $\mathrm{sgn}(z_j)$ is identical to the one studied throughout the paper—only the magnitude profile changes.

Symmetric activations (e.g. z^2, $\cos(z)$). Squaring or other even functions destroy sign information entirely: $(-z)^2 = z^2$. The sign–magnitude decomposition as stated does not apply. Yet a binary routing question remains meaningful—one could ask whether $|z_j|$ exceeds some data-dependent threshold τ_j (e.g. the median pre-activation magnitude), partitioning inputs into high-activation and low-activation regimes. Whether such a partition carries class-discriminative structure comparable to the sign code is an open empirical question. We note that even activations like z^2 are rarely used

in isolation in modern networks; they typically appear inside gated units (e.g. $z_1 \cdot z_2^2$ in some GLU variants) where the gating component restores a sign-preserving partition on at least one branch.

In summary, the sign-preserving assumption is not restrictive in practice: it covers all standard activations, and for the main exception (softplus), the pre-activation sign provides an equivalent partition. Extending the framework to genuinely non-monotone or even activations is possible in principle but would require a different choice of binary code, and we leave this to future work.

D Sign Code Extraction

This appendix documents the precise extraction procedure for the sign code $\mathbf{S}(x)$ used throughout the paper. We describe the procedure separately for each architecture family and specify exactly which tensors are recorded, which are discarded, and how the full-network sign code is assembled. All sign codes use the convention $\mathrm{sgn}(t) = +1$ if $t > 0$ and -1 otherwise (in particular, $\mathrm{sgn}(0) = -1$).

D.1 Feedforward networks (FFNN)

Architecture. Our FFNN models are of the form

$$x \xrightarrow{\text{flatten}} \mathbf{h}^{(0)} \xrightarrow{W^{(1)},\varphi} \mathbf{h}^{(1)} \xrightarrow{W^{(2)},\varphi} \cdots \xrightarrow{W^{(L)},\varphi} \mathbf{h}^{(L)} \xrightarrow{W^{(L+1)}} \text{logits}\,,$$

where every hidden layer applies the same sign-preserving activation φ (one of ReLU, SiLU, GELU, or tanh). The model exposes an `nn.ModuleList` of `nn.Linear` layers (`hidden_layers`) and a final `nn.Linear` output layer (`output_layer`).

Extraction. We register a forward hook on each `nn.Linear` in `hidden_layers`. The hook records the *pre-activation* $\mathbf{z}^{(\ell)} = W^{(\ell)}\mathbf{h}^{(\ell-1)} + \mathbf{b}^{(\ell)}$, i.e. the output of the linear layer *before* the activation is applied. After the forward pass completes, we apply φ to each captured pre-activation to obtain the post-activation $\mathbf{h}^{(\ell)} = \varphi(\mathbf{z}^{(\ell)})$, and compute:

$$\mathbf{S}^{(\ell)}(x) = \mathrm{sgn}\big(\mathbf{h}^{(\ell)}(x)\big) \in \{-1,+1\}^{d_\ell}\,, \tag{12}$$

$$\mathbf{M}^{(\ell)}(x) = \big|\mathbf{h}^{(\ell)}(x)\big| \in \mathbb{R}^{d_\ell}_{\geq 0}\,. \tag{13}$$

Because φ is sign-preserving, $\mathbf{S}^{(\ell)} = \mathrm{sgn}(\mathbf{z}^{(\ell)})$, so the sign code is equivalently determined by the pre-activation signs.

Full sign code. The full-network sign code is the concatenation across all L hidden layers:

$$\mathbf{S}(x) = \big(\mathbf{S}^{(1)}(x),\ \ldots,\ \mathbf{S}^{(L)}(x)\big) \in \{-1,+1\}^{D}\,, \qquad D = \textstyle\sum_{\ell=1}^{L} d_\ell\,.$$

The output layer $W^{(L+1)}$ is not activated and is therefore excluded from the sign code.

What is *not* extracted. The output logits are recorded for evaluation but do not contribute to the sign code. No sign code is computed for the input $\mathbf{h}^{(0)} = x$ (there is no activation to decompose).

D.2 Residual networks (ResNet)

Architecture. We use torchvision `BasicBlock`-based ResNets (ResNet-18, ResNet-34). The architecture comprises a convolutional stem followed by four layer groups, each containing multiple residual blocks, followed by global average pooling and a linear classifier:

$$x \xrightarrow[\text{stem}]{\text{conv1, bn1, relu, maxpool}} \mathbf{h}_{\text{stem}} \xrightarrow{\text{layer1}} \cdots \xrightarrow{\text{layer4}} \mathbf{h}_{\text{final}} \xrightarrow{\text{avgpool, fc}} \text{logits}\,.$$

Each `BasicBlock` has the following internal structure:

$$\mathbf{h}^{(\ell)} \xrightarrow{\text{conv1, bn1}} \mathbf{z}_{\text{int}} \xrightarrow{\text{relu}} \mathbf{a} \xrightarrow{\text{conv2, bn2}} \mathbf{b} \xrightarrow{+\text{ skip}} \mathbf{z}_{\text{out}} \xrightarrow{\text{relu}} \mathbf{h}^{(\ell+1)}\,,$$

where the skip connection is the identity when dimensions match, or a learned 1×1 convolution with batch normalisation (`downsample`) when a spatial or channel-dimension change occurs. Note that torchvision `BasicBlock` uses a *single* `nn.ReLU` module that is called twice during the forward pass: once after `bn1` (internal) and once after the residual addition (output).

Extraction. We register five hooks per block and one hook on the stem:

1. **Stem activation.** A forward hook on `model.relu` captures $\mathbf{h}_{\text{stem}} = \text{ReLU}(\text{bn1}(\text{conv1}(x)))$, the stem output after the first ReLU (but before max-pooling). We record $\mathbf{S}_{\text{stem}} = \text{sgn}(\mathbf{h}_{\text{stem}})$ and $\mathbf{M}_{\text{stem}} = |\mathbf{h}_{\text{stem}}|$.
2. **Block input.** A forward *pre*-hook on the block module captures the input tensor $\mathbf{h}^{(\ell)}$ entering the block. This serves as the skip connection when no downsampling is needed.
3. **Internal branch activation.** A forward hook on `block.relu` captures every ReLU output inside the block. In a `BasicBlock` the shared ReLU is called exactly twice; the *first* output is the internal branch activation $\mathbf{a} = \text{ReLU}(\text{bn1}(\text{conv1}(\mathbf{h}^{(\ell)})))$. We record $\mathbf{S}^{(\ell)}_{\text{int}} = \text{sgn}(\mathbf{a})$ and $\mathbf{M}^{(\ell)}_{\text{int}} = |\mathbf{a}|$.
4. **Branch output (pre-addition).** A forward hook on `block.bn2` captures $\mathbf{b} = \text{bn2}(\text{conv2}(\mathbf{a}))$, the output of the residual branch before addition with the skip.
5. **Skip connection.** If the block has a `downsample` module, a forward hook captures its output. Otherwise, the skip is the block input $\mathbf{h}^{(\ell)}$ recorded in step 2.
6. **Residual sum and block output.** We compute the residual sum $\mathbf{z}_{\text{out}} = \mathbf{b} + \text{skip}$ from the tensors captured in steps 4 and 5 (for the scaled ResNet-18s of the width–depth grid, Section E.2, the branch enters as $(\beta/\sqrt{B})\,\mathbf{b}$). This is the pre-activation of the block's output ReLU. The block-output activation $\mathbf{h}^{(\ell+1)} = \text{ReLU}(\mathbf{z}_{\text{out}})$ is captured via a forward hook on the block module itself. We record $\mathbf{S}^{(\ell)}_{\text{out}} = \text{sgn}(\mathbf{h}^{(\ell+1)})$ and $\mathbf{M}^{(\ell)}_{\text{out}} = |\mathbf{h}^{(\ell+1)}|$. Because ReLU is sign-preserving, $\mathbf{S}^{(\ell)}_{\text{out}} = \text{sgn}(\mathbf{z}_{\text{out}})$; we additionally store $\text{sgn}(\mathbf{z}_{\text{out}})$ and $|\mathbf{z}_{\text{out}}|$ for diagnostic use.

Full sign code. The full-network sign code concatenates the stem code and both the internal and output codes from every block:

$$\mathbf{S}(x) = \big(\, \mathbf{S}_{\text{stem}},\ \mathbf{S}^{(1)}_{\text{int}},\ \mathbf{S}^{(1)}_{\text{out}},\ \ldots,\ \mathbf{S}^{(B)}_{\text{int}},\ \mathbf{S}^{(B)}_{\text{out}} \,\big)\,,$$

where B is the total number of `BasicBlock`s across all four layer groups (e.g. $B = 8$ for ResNet-18). For convolutional layers the sign code tensors have shape $C \times H \times W$; we compress each to $C \cdot P^2$ bits by taking the sign of the average-pooled activation on a $P \times P$ grid ($P = 2$; Section D.4).

What is *not* extracted. The global average-pooling layer and the final linear classifier (`fc`) are not activated by a sign-preserving nonlinearity and do not contribute to the sign code. The second ReLU output per block (captured by the shared `block.relu` hook) is functionally identical to the block-output activation captured by the block-level hook; we use the latter and discard the former. Our CIFAR ResNets have no stem max-pooling (it is replaced by the identity), so the stem contributes only its ReLU code.

D.3 Vision Transformer (ViT)

Architecture. Our ViT follows the standard pre-norm design:

$$x \xrightarrow{\text{patch embed}} \mathbf{t} \xrightarrow{+\text{ cls, pos}} \mathbf{h}^{(0)} \xrightarrow{\text{block}_1} \cdots \xrightarrow{\text{block}_B} \mathbf{h}^{(B)} \xrightarrow{\text{norm, head}} \text{logits}\,,$$

where each transformer block has the structure

$$\mathbf{h}^{(\ell)}_{\text{attn}} = \mathbf{h}^{(\ell-1)} + \text{Attn}\big(\text{LN}_1(\mathbf{h}^{(\ell-1)})\big)\,,$$
$$\mathbf{h}^{(\ell)} = \mathbf{h}^{(\ell)}_{\text{attn}} + \text{MLP}\big(\text{LN}_2(\mathbf{h}^{(\ell)}_{\text{attn}})\big)\,.$$

The MLP sub-block is $\text{Linear} \to \text{GELU} \to \text{Dropout} \to \text{Linear} \to \text{Dropout}$.

Extraction. Unlike the FFNN and ResNet extractors, the ViT extractor performs a *manual forward pass* through the model's modules rather than relying on hooks. This gives explicit access to every intermediate tensor. For each transformer block $\ell = 1, \ldots, B$, we record three sign–magnitude pairs:

1. **Post-attention residual stream.** $\mathbf{h}^{(\ell)}_{\text{attn}} = \mathbf{h}^{(\ell-1)} + \text{Attn}(\text{LN}_1(\mathbf{h}^{(\ell-1)}))$. We record $\mathbf{S}^{(\ell)}_{\text{attn}} = \text{sgn}(\mathbf{h}^{(\ell)}_{\text{attn}})$ and $\mathbf{M}^{(\ell)}_{\text{attn}} = |\mathbf{h}^{(\ell)}_{\text{attn}}|$. This is the sign of the residual stream *after* the attention sub-block has added its contribution.

2. **MLP hidden activation.** $\mathbf{a}^{(\ell)} = \mathrm{GELU}\big(W_1^{(\ell)}\,\mathrm{LN}_2(\mathbf{h}_{\mathrm{attn}}^{(\ell)}) + \mathbf{b}_1^{(\ell)}\big)$. We record $\mathbf{S}_{\mathrm{mlp}}^{(\ell)} = \mathrm{sgn}(\mathbf{a}^{(\ell)})$ and $\mathbf{M}_{\mathrm{mlp}}^{(\ell)} = |\mathbf{a}^{(\ell)}|$. This is the internal sign code of the MLP, analogous to the internal branch code in a ResNet block. GELU is sign-preserving, so $\mathbf{S}_{\mathrm{mlp}}^{(\ell)}$ is determined by the pre-activation signs.
3. **Block output.** $\mathbf{h}^{(\ell)} = \mathbf{h}_{\mathrm{attn}}^{(\ell)} + \mathrm{MLP}(\mathrm{LN}_2(\mathbf{h}_{\mathrm{attn}}^{(\ell)}))$. We record $\mathbf{S}_{\mathrm{out}}^{(\ell)} = \mathrm{sgn}(\mathbf{h}^{(\ell)})$ and $\mathbf{M}_{\mathrm{out}}^{(\ell)} = |\mathbf{h}^{(\ell)}|$. As with ResNets, the output sign code need not equal the input sign code: sign flips introduced by the MLP branch refine the partition.

Additionally, we record the patch-embedding tokens $\mathbf{t} = \mathrm{PatchEmbed}(x)$ and their sign and magnitude. These serve as the "stem" of the ViT and are available for diagnostic use, though they do not pass through a sign-preserving activation and are therefore not included in the default sign code.

Full sign code. The full-network sign code concatenates all three code families across all B transformer blocks:

$$\mathbf{S}(x) = \big(\,\mathbf{S}_{\mathrm{attn}}^{(1)},\ \mathbf{S}_{\mathrm{mlp}}^{(1)},\ \mathbf{S}_{\mathrm{out}}^{(1)},\ \ldots,\ \mathbf{S}_{\mathrm{attn}}^{(B)},\ \mathbf{S}_{\mathrm{mlp}}^{(B)},\ \mathbf{S}_{\mathrm{out}}^{(B)}\,\big)\,.$$

Each sign code tensor has shape $(\text{batch},\ T,\ d_{\mathrm{model}})$ for the residual-stream codes and $(\text{batch},\ T,\ d_{\mathrm{mlp}})$ for the MLP hidden code, where T is the sequence length (number of patches plus one `[CLS]` token). For the last-layer code of Section 3 and Section 4 we keep the `[CLS]` token and the first patch token and flatten over them (Section D.6).

What is *not* extracted. The self-attention weights $A \in \Delta_T^T$ are continuous routing variables and do not admit a binary sign decomposition; they are not included in the sign code. The final layer norm and the classification head do not involve a sign-preserving activation and are excluded.

D.4 Spatial compression of convolutional sign codes

Convolutional sign code tensors $\mathbf{S}^{(\ell)} \in \{-1,+1\}^{C\times H\times W}$ can be very high-dimensional (e.g. $256\times 8\times 8 = 16{,}384$ bits for a single layer of ResNet-18 on 32×32 inputs). To obtain a fixed-length code that is comparable across spatial resolutions, we partition each channel's $H\times W$ feature map into a $P\times P$ grid of (approximately) equal patches, average the activation over each patch and keep the sign of the average:

$$\bar{S}_{c,p,q}^{(\ell)} = \mathrm{sgn}\Big(\tfrac{1}{|\mathrm{patch}(p,q)|}\textstyle\sum_{(h,w)\in\mathrm{patch}(p,q)} h_{c,h,w}^{(\ell)}\Big)\,.$$

This yields $C\cdot P^2$ bits per layer. We use $P=2$ throughout, which balances spatial resolution against code dimensionality. The pooled bit is a function of the activations, not of the signs alone: for ReLU it is $+1$ iff any location in the patch is active, and for smooth activations it can depend on the magnitudes. The pooled code is therefore a compressed summary of the layer rather than a subset of its spatial sign bits.

D.5 Random projection for high-dimensional codes

Even after spatial compression, the full-network sign code can have $D\sim 10^4$–10^5 dimensions. Computing exact Hamming distances between all pairs of n points in $\{-1,+1\}^D$ costs $\mathcal{O}(n^2D)$, which becomes prohibitive for large n and D. We reduce the dimensionality with a random projection: the sign code is multiplied by a matrix $R\in\{-1,+1\}^{D\times d}/\sqrt{d}$ with i.i.d. Rademacher entries, $d\ll D$, and the kNN vote ranks training points by the dot product of the real-valued projected codes $\mathbf{S}(x)R$, without re-binarising. Since $\mathbb{E}[RR^\top] = I_D$, these dot products equal those of the full codes in expectation,

$$\mathbb{E}\big[\mathbf{S}(x)RR^\top\mathbf{S}(x')^\top\big] = \mathbf{S}(x)\cdot\mathbf{S}(x') = D\big(1-2\,d_H(x,x')\big),$$

and concentrate around them as d grows (Johnson–Lindenstrauss), so the projected vote approximates the vote on Hamming distance. In all experiments labelled "projected" in the main paper, we use $d = 2{,}048$.

D.6 Summary of extracted quantities

Table 1 summarises which tensors are recorded for each architecture and how they contribute to the full sign code.

Table 1: Summary of sign code extraction across architectures. *Int.* = internal branch/MLP hidden code; *Out.* = block-output code; *Attn.* = post-attention residual stream code. "Full $\mathbf{S}$" indicates inclusion in the full-network sign code; indicates the code used for the last-layer sign-space k-NN experiment in Section 3.

Architecture	Tensor	Code type	Full S	§Section 3
FFNN	$\mathbf{h}^{(\ell)} = \varphi(\mathbf{z}^{(\ell)})$	Layer sign	✓	last only
	Output logits	—	×	×
ResNet	Stem ReLU output	Stem	✓	×
	$\mathbf{a} = \mathrm{ReLU}(\mathrm{bn1}(\mathrm{conv1}(\cdot)))$	Int.	✓	last only
	$\mathbf{h}^{(\ell+1)} = \mathrm{ReLU}(\mathbf{b} + \mathrm{skip})$	Out.	✓	last only
	Avgpool / fc	—	×	×
ViT	Patch tokens (pre-transformer)	Stem	×	×
	$\mathbf{h}^{(\ell)}_{\mathrm{attn}}$	Attn.	✓	×
	$\mathbf{a}^{(\ell)} = \mathrm{GELU}(\cdot)$	Int.	✓	×
	$\mathbf{h}^{(\ell)}$ (block output)	Out.	✓	last only
	Final norm / head	—	×	×

Which codes are used where. The informational sufficiency experiment (Section 3) uses only the *last* hidden layer's sign code: the final hidden layer for FFNNs, the pooled branch and output codes of the last `BasicBlock` for ResNets, and the output code of the last transformer block at the `[CLS]` token and the first patch token for ViTs. This is a deliberately conservative choice: a single layer's binary code—a small fraction of the full-network sign code—already accounts for the network's generalization performance. The scaling experiment (Section 4) reads its own last-layer codes, listed in Section F. The full-network sign code (all layers concatenated) is used where an analysis needs the complete routing record, such as the statistic Q of Section 6.

E Training Details

E.1 Training Details & Hyperparameters

Scope. This appendix summarizes the training protocol used for the main correlation sweeps, the intervention experiments, and the repulsion-regularized runs.

Baseline sweeps. The main width–depth heatmaps use CIFAR-10 with the full $50{,}000$-example training set. The cross-dataset scatter in Figure 2b uses three architecture families (FFNN, ResNet-18, ViT), four activations (ReLU, SiLU, GELU, tanh), three datasets (MNIST, CIFAR-10, CIFAR-100), five training-set sizes (1k, 5k, 10k, 25k, 50k), and three random seeds, for a total of 540 trained models.

Optimization.

- **Loss:** cross-entropy throughout; label smoothing is zero by default, with ViT on CIFAR-10 using 0.1.
- **Optimizers:** AdamW for FFNN and ViT; SGD with Nesterov momentum for ResNet-18.
- **Base learning rates:** FFNN uses base learning rate 3×10^{-4}; ResNet-18 uses 10^{-1}; ViT uses 1.5×10^{-4} on CIFAR-10.

- **Scaling rule:** for FFNN sweeps, the effective learning rate is

$$\mathrm{lr}_{\mathrm{eff}} = 3 \times 10^{-4} \cdot \mathrm{clip}\left(\left(\frac{1024}{w}\right)^{1/2}\left(\frac{6}{d}\right)^{1/2}, 0.1, 1.0\right),$$

  with an additional factor 0.2 for tanh. For ViT sweeps, the effective learning rate is

$$\mathrm{lr}_{\mathrm{eff}} = 1.5 \times 10^{-4} \cdot \mathrm{clip}\left(\left(\frac{1024}{w}\right)\left(\frac{6}{d}\right)^{1/2}, 0.125, 1.0\right).$$

- **Schedules:** FFNN uses a constant learning rate in the baseline sweeps. ResNet-18 and ViT use cosine decay; ResNet-18 uses 5 warmup epochs and ViT uses 10 warmup epochs.
- **Batch sizes:** 256 for FFNN, 64 for ResNet-18, and 128 for ViT.
- **Weight decay:** 10^{-4} for FFNN, 5×10^{-4} for ResNet-18, and 5×10^{-2} for ViT.
- **Mixed precision:** enabled for all large sweeps unless explicitly disabled.

Stopping rule. The main heatmap sweeps use a maximum of 120 epochs together with a practical early-stop rule based on training accuracy: 0.95 for FFNN, 0.995 for ResNet-18, and 0.98 for ViT. These thresholds were used as a compute-saving heuristic once the model had clearly fit the training set; they were not used in the repulsion runs, where late-training dynamics are part of the result.

Data and augmentation.

- **Splits:** standard train/test splits; no held-out validation split.
- **Augmentation:** ResNet-18 and ViT use RandomCrop$(32, \mathrm{padding} = 4)$ and RandomHorizontalFlip on CIFAR; horizontal flips are disabled for SVHN. FFNN uses no data augmentation.
- **Normalization:** standard per-dataset mean/std normalization.

Initialization and normalization. FFNN layers use Kaiming initialization for ReLU/SiLU/GELU and Xavier initialization for tanh. ResNet-18 uses the standard convolutional and batch-normalized residual-block parameterization. ViT uses the standard pre-norm transformer parameterization with layer norm.

Seeds and counts. The fixed-size cross-dataset sweep uses three seeds per configuration. The main width–depth heatmaps use one seed per grid point. The CIFAR-10 heatmaps in Figure 2a comprise 132 FFNN models, 108 ResNet-18 models, and 113 ViT models (353 total).

Intervention experiments. The sign-flip and magnitude-replacement experiments use corruption rates

$$0,\ 0.01,\ 0.02,\ 0.05,\ 0.1,\ 0.15,\ 0.2,\ 0.25,\ 0.3,\ 0.4,\ 0.5,\ 0.6,\ 0.7,\ 0.8,\ 0.9,\ 1.0,$$

with five independent corruption masks per setting and three seeds per network, applied to all hidden layers. A flipped unit outputs $\varphi(-z)$; a replaced magnitude is drawn as $|\eta|$, $\eta \sim \mathcal{N}(0, 1)$, with the clean signs enforced downstream.

Repulsion runs. The repulsion experiments are trained on SVHN with longer schedules (maximum 2500 epochs), AdamW, the same FFNN learning-rate rule, and no early stopping. All other baseline hyperparameters are kept fixed unless explicitly overridden by the regularizer.

Compute budget. The full project required approximately **400 H100 GPU-hours** on a 4×H100 cluster, including the main sweeps, intervention experiments, repulsion reruns, and auxiliary failed/restarted jobs. This number should be understood as a rough end-to-end estimate rather than an exact accounting.

E.2 Architecture Specifications

FFNN. Inputs are flattened to vectors. All hidden layers share the same width w, and the depth d equals the number of hidden layers. The activation is ReLU, SiLU, GELU, or tanh, followed by a linear classifier. There is no final normalization layer before the output head.

ResNet-18. We use a CIFAR-style residual network rather than the ImageNet torchvision stem: the input stem is a 3×3 convolution with stride 1 and no max-pooling. Residual blocks use batch normalization, and the residual branch is scaled by $\beta/\sqrt{B}$ with $\beta = 1$ and B the total number of residual blocks. Channel widths follow $(1, 2, 4, 8) \times (\text{base width})$, scaled uniformly by the chosen width multiplier.

Depth allocation for ResNet-18. When the total residual depth is varied, blocks are distributed across the four stages by round-robin allocation. Concretely, depth $d = 5$ gives stage layout $(2, 1, 1, 1)$, depth $d = 6$ gives $(2, 2, 1, 1)$, and depth $d = 10$ gives $(3, 3, 2, 2)$.

ViT. The ViT models use patch size 4, a learned class token, learned positional embeddings, pre-norm transformer blocks, multi-head self-attention with 8 heads, an MLP ratio of 4, and a linear classifier on the `[CLS]` token. Width controls the embedding dimension and depth controls the number of transformer blocks.

Sweep grids.

- **FFNN (Figure 2a):** widths $\{4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096\}$ and depths $\{1, \ldots, 12\}$.
- **ResNet-18 (Figure 2a):** widths $\{1, 2, 4, 8, 16, 32, 64, 128, 256\}$ and depths $\{1, \ldots, 12\}$.
- **ViT (Figure 2a):** widths $\{8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096\}$ and depths $\{1, \ldots, 12\}$, with the seven largest models $(4096, 6\text{–}12)$ omitted.
- Q **panel (Figure 5a):** FFNNs on SVHN and CIFAR-10 at widths 4–2048 and depths 1–12.

Reference parameter counts. For CIFAR-sized inputs, representative models have:

- **FFNN** w=1024, d=8**:** 10,504,202 parameters.
- **ResNet-18** w=16, d=8**:** 701,466 parameters.
- **ViT** w=256, d=6**:** 4,771,082 parameters.

E.3 Repulsion Regularizer

Definition. For a batch with labels y, we form a continuous proxy for the sign or gate pattern by concatenating selected pre-activations across layers, applying a temperature-scaled proxy, $g = \sigma(z/\tau)$ for ReLU/SiLU/GELU and $g = \tanh(z/\tau)$ for tanh, and then ℓ_2-normalizing. The repulsion loss penalizes within-class alignment:

$$\mathcal{L}_{\text{rep}} = \frac{1}{|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} (g_i^\top g_j)^2, \qquad \mathcal{P} = \{(i, j) : i \neq j,\ y_i = y_j\}.$$

The total loss is

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda(t)\, \mathcal{L}_{\text{rep}}.$$

Schedule and layers. The regularization weight is linearly ramped from zero:

$$\lambda(t) = \lambda_0 \cdot \min\Big(1, \max\big(0, \tfrac{t-t_0}{T}\big)\Big).$$

For the Figure 5 runs, $t_0 = 5$ epochs and $T = 200$. The proxy is computed from the pre-activations of all FFNN hidden layers.

Training protocol. The repulsion runs keep the baseline optimizer and data pipeline, but use a long schedule (2500 epochs) with no early stopping so that the late-stage collapse in test accuracy is visible. In the main SVHN experiments, the regularizer uses $\tau = 0.005$ and $\lambda_0 = 500$.

Algorithm 1 Repulsion-regularized training

for each minibatch (x, y) **do**
  compute logits and hidden pre-activations $\{z^{(\ell)}\}$
  form proxy features $g \leftarrow \mathrm{concat}_\ell\, \phi(z^{(\ell)}/\tau)$, where $\phi = \sigma$ or $\tanh$
  normalize each $g_i \leftarrow g_i/(\|g_i\|_2 + \varepsilon)$
  form same-class pairs $\mathcal{P} = \{(i, j) : i \neq j,\ y_i = y_j\}$
  compute $\mathcal{L}_{\mathrm{rep}} = |\mathcal{P}|^{-1} \sum_{(i,j)\in\mathcal{P}} (g_i^\top g_j)^2$
  compute $\mathcal{L} = \mathcal{L}_{\mathrm{CE}} + \lambda(t)\mathcal{L}_{\mathrm{rep}}$
  backpropagate $\mathcal{L}$ and update parameters
end for

Sensitivity. The effect is not confined to a single regularization strength: in additional SVHN reruns, both $\lambda_0 = 250$ and $\lambda_0 = 500$ preserved near-perfect training accuracy while producing the same qualitative collapse in test accuracy and purity.

Regularizer-specific hyperparameters.

- **Weight:** $\lambda_0 = 500$ in the main runs.
- **Temperature:** $\tau = 0.005$.
- **Normalization constant:** $\varepsilon = 10^{-8}$.
- **Proxy:** sigmoid for ReLU/SiLU/GELU; tanh for tanh.
- **Layers:** all hidden layers.

E.4 Sign-kNN Protocol & Sensitivity

Protocol. Sign-kNN is computed from the last-layer sign code. Codes are stored as $\{-1, +1\}$ vectors and compared by dot product, which is equivalent to Hamming distance up to an affine transformation. The main-paper sweeps use a fixed $k = 15$ with majority-vote prediction and deterministic tie-breaking by class count. Training examples form the reference bank and test examples are used as queries.

Random projection. When projected sign-kNN is reported, the code is first mapped by a random $\{\pm 1\}$ projection to dimension $d = 512$ and averaged over five independent projections. The main sign-kNN results in the paper use the full sign code rather than the projected code.

Sensitivity to k. To check that the main conclusions are not sensitive to the choice of k, we re-evaluated three overparameterized SVHN SiLU FFNN checkpoints using a 20k-example training bank and 5k test queries. The table reports *accuracy* (not error rate):

Model	model acc.	k=1	k=3	k=5	k=10	k=20
w=256, d=8	0.817	0.799	0.807	0.808	0.809	0.809
w=1024, d=8	0.826	0.777	0.770	0.780	0.780	0.778
w=1024, d=10	0.830	0.823	0.826	0.825	0.826	0.825

Across the regime where sign-kNN is already close to the network accuracy, performance varies only weakly with k; changing k from 1 to 20 alters accuracy by at most a few points. The w=1024, d=8 row sits near the transition region, where the sign gap is still visible but already small enough to preserve the same qualitative conclusion.

F Details for the Scaling Law of the Sign Code

Codes. We read one last-layer sign code per network, without spatial pooling or projection: the last hidden layer of the MLPs; the 512-unit fully connected hidden layer before the head of the CNN; the signs of the pre-activation output of the last block of ResNet-18, all 8,192 channel–position units;

and for ViTs the output code of the last transformer block at the [CLS] token and the first patch token (384 units on the ViT data ladder, 512 for the five further ViTs). For the ResNets this is not the code read in Section 3 (Section D).

Data ladders. A ladder trains one architecture on the first D images of a fixed random order of the training set, for six values of D from 2k up to 45k (ViT), 50k (CIFAR-10) or 65k (SVHN), with two seeds each. The MLPs are ReLU networks of depth 3 or 6 trained with μP and Adam at two learning-rate multipliers; the CNN is trained at two learning rates; the ResNet-18s follow the recipe of Section E on CIFAR-10 with and without augmentation, and on SVHN. The ViT ladder uses a smaller ViT than Section 3 (depth 6, width 192, three heads, AdamW, crop and flip augmentation, best-validation checkpoints). The five further ViTs are trained on all of CIFAR-10 with the recipes of Section E. Loss is measured on six ladders (two MLPs, a CNN, three ResNet-18s) and the ViT ladder, plus the five ViTs: 89 networks. Exponents are measured on 14 ladders.

Posterior and loss. The bank is the network's own training set, and the queries are the 10,000 test images. The temperature β of Equation (7) is chosen from a grid of 24 values, and the temperature of the network's logits is fitted, both by minimum NLL on a fixed half of the queries; all reported losses are measured on the other half. Figure 8 shows the ratio of the two losses for every network: 0.95–1.05 for the MLPs, 1.02–1.13 for the CNN, 0.91–1.04 for the ResNet-18s and 0.97–1.05 for the ViTs. Along each ladder, the loss of the code falls with D at 0.95–1.10 times the network's rate. Read the same way, the code of the network at initialisation and the pixels have a loss at least 15% above the network's in 87 and 81 of the 89 cases.

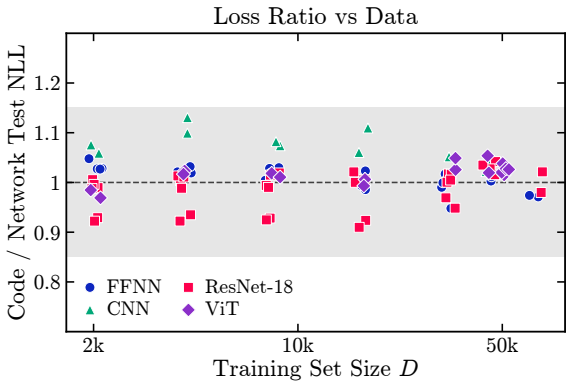


Figure 8: Calibrated test NLL of the code's posterior divided by the network's, for all 89 networks of Section 4, against training-set size (jittered horizontally). Shaded: $\pm 15\%$.

Exponents. The exponent α is minus the slope of $\ln(\text{test error})$ against $\ln D$, fitted over the training-set sizes at which the network's accuracy exceeds 0.3 (CIFAR-10) or 0.5 (SVHN), using seed means. The code's error is that of sign-kNN ($k = 15$) on the network's own last-layer code, with the network's training set as the bank. On the 14 ladders the ratio of the code's exponent to the network's is 0.89–1.06; per seed it is 0.87–1.11, and the agreement is loosest for the MLPs. On the 36 learning curves behind Figure 2b (three architectures, four activations, three datasets) the ratio is within 10% on 33 and within 15% on 34; the two outside, FFNNs with tanh on CIFAR-10 (1.26) and CIFAR-100 (1.21), are cases in which the code ends more accurate than the network.

Code against bank. To separate the code from the bank, we also read a fixed code, that of the network trained on all the data, with a bank of the first D training images, and fit its exponent α_{fixed}. The share of the network's exponent that comes from the code improving, $1 - \alpha_{\text{fixed}}/\alpha_{\text{net}}$, is 91–98% in the ResNet-18s, 90–94% in the ViT, 82–92% in the CNNs and 43–92% in the MLPs (per seed).

Scope. The result concerns the last-layer codes described above. Not every sign code of a network carries the loss: in the ViTs, the code of the last block's MLP gate at the same tokens is sparse (13–18% of units active) and, at large D, has up to 1.34 times the network's loss.

G Removing the Sign or the Magnitude Channel Fairly

The interventions of Section 5 measure how badly a network fails when its signs are flipped, or when its magnitudes are replaced at random while its signs are held. They are not a matched comparison of what the two channels carry: a flip does more than delete a unit's sign, it sends the unit to the wrong side of its hyperplane and so adds a misleading signal downstream. Here we delete each channel in the same way and with the same number of bits.

Protocol. In every hidden layer a random set B containing a share α of the units loses one channel. Their activations are replaced by the conditional expectation given what each unit keeps and given the rest of the layer, $\mathbb{E}[\mathbf{h}_B \mid \mathbf{c}_B, \mathbf{h}_{B^c}]$, estimated as a linear predictor from the training set, and the network runs forward from the replaced activations. With the sign channel a unit keeps its sign; with the magnitude channel it keeps whether $|z_j|$ lies above or below its median over the training set. Both therefore keep one bit per unit. For comparison, the magnitude channel is also given 16 equal-count bins of $|z_j|$, four bits. We use the 24 networks of Section 5: FFNNs on the full SVHN test set with six draws of B, and ResNet-18s on 5,000 CIFAR-10 test images with one draw, where the expectation is taken across channels at each spatial position. Table 2 reports the share of test images whose prediction is unchanged.

Table 2: Fair removal of one channel in every layer: share of test images on which the prediction is unchanged, when a share α of the units keeps only its sign (S, 1 bit), only a median split of $|z_j|$ (M_1, 1 bit) or 16 equal-count bins of $|z_j|$ (M_4, 4 bits). FFNNs and ResNet-18s: means over three seeds; ViT: mean over the five CIFAR-10 ViTs of our recipe, with the MLP hidden layer of every block as the site.

		$\alpha = 0.2$			$\alpha = 0.5$			$\alpha = 1$		
Network	φ	S	M_1	M_4	S	M_1	M_4	S	M_1	M_4
FFNN	ReLU	0.93	0.94	0.95	0.87	0.89	0.92	0.54	0.63	0.86
FFNN	SiLU	0.92	0.93	0.95	0.86	0.88	0.91	0.15	0.45	0.77
FFNN	GELU	0.93	0.94	0.95	0.87	0.89	0.91	0.23	0.52	0.77
FFNN	tanh	0.94	0.92	0.92	0.91	0.86	0.87	0.80	0.55	0.70
ResNet-18	ReLU	0.94	0.93	0.95	0.77	0.51	0.62	0.24	0.11	0.10
ResNet-18	SiLU	0.94	0.93	0.95	0.75	0.50	0.53	0.23	0.11	0.10
ResNet-18	GELU	0.94	0.93	0.95	0.75	0.50	0.60	0.23	0.10	0.10
ResNet-18	tanh	0.95	0.90	0.90	0.88	0.58	0.61	0.61	0.09	0.10
ViT	GELU	0.97	0.96	0.96	0.94	0.92	0.92	0.86	0.73	0.76

At low doses neither channel is needed in full. With a fifth of every layer reduced to one bit per unit, either channel keeps 90–97% of the network's decisions, whereas flipping a fifth of the signs leaves 10–14% accuracy (Figure 4). The flip's damage therefore comes from sending units to the wrong side of their hyperplanes, not from the information it removes.

In residual networks the sign is the more important channel. Once half or more of each layer loses a channel, the ResNet-18s depend on their signs, for all four activations. With half of every block's units reduced to one bit, the sign keeps 0.75–0.88 of the decisions and the magnitude bit only 0.50–0.58; even four bits of magnitude, four times the sign's budget, keep only 0.53–0.62. With every unit reduced the sign still keeps 0.23–0.61, while the magnitudes, at one or at four bits, keep 0.09–0.11, the level of a constant prediction. In these networks the sign code is not only sufficient for the decision (Section 3) but also the channel the computation needs most.

Vision transformers lean the same way. We repeat the test on the five CIFAR-10 ViTs of our recipe (GELU, dim 256, eight heads), reducing the units of the MLP hidden layer in every block, at every token. Bit for bit the sign is again the more important channel, by a small margin with half of the units reduced (0.94 of the decisions against 0.92 for the magnitude bit, and 0.92 with four magnitude bits) and clearly with every unit reduced (0.86 against 0.73, and 0.76 with four bits). Only the MLP hidden layers are reduced, while attention and the residual stream are left intact, so all values stay high.

Feedforward networks. The tanh FFNNs behave like the ResNets (0.91 against 0.86 at $\alpha = 0.5$, 0.80 against 0.55 at $\alpha = 1$). In the ReLU, SiLU and GELU FFNNs the magnitude bit carries slightly more (0.88–0.89 against 0.86–0.87 at $\alpha = 0.5$, and 0.45–0.63 against 0.15–0.54 at $\alpha = 1$), and four magnitude bits more still; these are also the networks whose accuracy drops most under the magnitude replacement of Figure 4.

Limits. The estimator is linear and the same for both channels, so it understates what either channel carries. The ResNet and ViT values rest on one draw on 5,000 test images, with a standard error of about 0.005.

H Margin, Resolution and Vote Stability: Proofs

H.1 Setting

Let (X, Y) be drawn from the test distribution, with $Y \in \{1, \dots, C\}$. Fix a trained network, its last-layer representation $\mathbf{z}^{(L)}$ and its code $\mathbf{S}^{(L)} = \operatorname{sgn} \mathbf{z}^{(L)}$, with $\operatorname{sgn}(0) = -1$ as in Definition 2.1. The labelled training set $\{(x_i, y_i)\}_{i=1}^{m}$, $m \geq k$, serves as the bank; the representation may have been learned from it. We write $d_H(x, x_i)$ for the normalised Hamming distance (Equation (5)) between the codes of x and x_i, and $\mathcal{R}(g) = \Pr[g(X) \neq Y]$ for the error of a classifier g. Sign-kNN uses a fixed rule for distance ties among bank points and a fixed rule for ties in the class vote. In the proof of Proposition 7.1(ii) and in Lemma H.1, probabilities are over the random hyperplanes, with the network, the training set and the test point fixed. Everywhere else, probabilities are over fresh test inputs, conditional on the trained network and the training set.

H.2 Proof of Proposition 7.1

Fix a test point x with label y and write $p_i = p(x, x_i)$ and $d_i = d_H(x, x_i)$. Let $j = \lfloor k/2 \rfloor + 1$, let p_{same} be the j-th smallest p_i over $y_i = y$ and p_{other} the $(k - j + 1)$-th smallest p_i over $y_i \neq y$. We assume that class y has at least j training points and the other classes together at least $k - j + 1$, so that both exist, and that $g = p_{\text{other}} - p_{\text{same}} > 0$. Set $T = p_{\text{same}} + g/2 = p_{\text{other}} - g/2$.

Part (i). Suppose $|d_i - p_i| < g/2$ for all i. At least j training points of class y have $p_i \leq p_{\text{same}}$, hence $d_i < T$. Every point of another class, except at most $k - j$ of them, has $p_i \geq p_{\text{other}}$, hence $d_i > T$. So at most $k - j$ points of other classes have $d_i < T$. The k nearest codes either all have $d_i < T$, in which case at most $k - j$ of them belong to other classes, or they include every point with $d_i < T$, in which case at least j of them belong to class y. Either way class y holds at least $j > k/2$ of the k votes and wins, under any tie rule.

Part (ii). If the n hyperplanes are independent draws from a fixed distribution, d_i is the mean of n independent indicators with mean p_i. Hoeffding's inequality [Hoeffding, 1963] gives $\Pr(|d_i - p_i| \geq g/2) \leq 2e^{-ng^2/2}$ for each i, and a union bound over the m training points shows that the condition of part (i) holds with probability at least $1 - 2me^{-ng^2/2}$. Independence is needed across hyperplanes, not across training points. For hyperplanes through the origin with isotropic Gaussian normals, $p_i = \theta_i/\pi$ [Goemans and Williamson, 1995, Charikar, 2002]; for hyperplanes with offsets, p_i is the corresponding separation probability.

Why $g > 0$ is needed. Without it the statement fails: with eight other-class training points at the query's representation and eight same-class points at its antipode, every central hash votes for the wrong class.

A sharper form. Part (i) uses only the class-y points with $p_i \leq p_{\text{same}}$, which need $d_i < T$, and the other-class points with $p_i \geq p_{\text{other}}$, which need $d_i > T$. Each needs only a one-sided deviation smaller than its own distance $|p_i - T| \geq g/2$ to the threshold. With the one-sided Hoeffding inequality and a union bound over this set I of points,

$$\Pr[\text{sign-}k\text{NN is wrong at } x] \;\leq\; \sum_{i \in I} e^{-2n(p_i - T)^2} \;\leq\; m\, e^{-ng^2/2}.$$

Points far from the threshold contribute almost nothing, so the effective number of competitors is the number of training points whose separation lies within about $1/\sqrt{n}$ of T, not m.

H.3 Random cuts recover the exact neighbours

For a fixed query and training set, let n cuts be independent draws from a fixed distribution, and let p_i be the probability that one cut separates the query from training point i. Hoeffding's inequality and a union bound give

$$\Pr_{\text{cuts}}\Big[\max_{i\le m}|d_H(X,x_i)-p_i|>\varepsilon\Big] \;\le\; 2m\exp(-2n\varepsilon^2). \tag{14}$$

Lemma H.1 (Exact neighbours from enough random cuts)**.** *Order the training set by p_i and suppose that the k-th and $(k+1)$-th smallest values differ by more than $g>0$. Then, with probability at least $1-2m\exp(-ng^2/2)$ over the cuts, the k Hamming-nearest training points are exactly the k points with the smallest p_i. For central isotropic Gaussian cuts these are the k angular nearest neighbours.*

Proof. Apply Equation (14) with $\varepsilon=g/2$. On the complementary event, every point among the k smallest p_i has $d_H\le p_{(k)}+g/2<p_{(k+1)}-g/2$, and every other point has $d_H\ge p_{(k+1)}-g/2$, so the k Hamming-nearest points are exactly these, with no ties. □

The units of a trained layer are learned and correlated, so this rate need not transfer to them; Section I measures how closely they behave like random cuts.

H.4 A reference-reader form of the condition: vote stability

The margin of Proposition 7.1 can also be stated against the vote with exact angles. Let $A_k(x)$ be the k training points nearest to x in angle in $\mathbf{z}^{(L)}$, $H_k(x)$ the k Hamming-nearest training codes, and a_k, h_k the plurality votes over them (angular kNN and sign-kNN), with the same tie rules. For a set of indices A write $N_c(A)=\sum_{i\in A}\mathbf{1}\{y_i=c\}$, and let $v_A(x)=(N_c(A_k(x)))_c$ and $v_H(x)=(N_c(H_k(x)))_c$ be the class-count vectors of the two neighbour sets. Define the number of swapped neighbours, the margin of the angular vote, and the net change in class counts,

$$t_k(x)=k-|A_k(x)\cap H_k(x)|, \qquad \tau_k(x)=\tfrac{1}{2}\|v_H(x)-v_A(x)\|_1,$$
$$V_k(x)=N_{a_k(x)}(A_k(x))-\max_{c\ne a_k(x)}N_c(A_k(x)).$$

V_k is zero for a tied angular vote. τ_k is the smallest number of label substitutions that turns one count vector into the other; changes in opposite directions cancel, so it is not a count of individual replacements that change a label. None of the three quantities uses the test label.

Proposition H.2 (Vote stability)**.** *If $V_k(x)>2\tau_k(x)$, then $h_k(x)=a_k(x)$. Consequently,*

$$\mathcal{R}(h_k)\;\le\;\Pr\big[a_k(X)\ne Y \text{ or } V_k(X)\le 2\tau_k(X)\big]\;\le\;\mathcal{R}(a_k)+\Pr[V_k\le 2\tau_k]. \tag{15}$$

Proof. Write $a=a_k(x)$ and $\Delta=v_H(x)-v_A(x)$. The two sets have equal size, so $\sum_c\Delta_c=0$ and $\|\Delta\|_1=2\tau_k(x)$. For each competitor $c\ne a$,

$$\begin{aligned}N_a(H_k(x))-N_c(H_k(x)) &= \big(N_a(A_k(x))-N_c(A_k(x))\big)+(\Delta_a-\Delta_c)\\ &\ge V_k(x)-\big(|\Delta_a|+|\Delta_c|\big)\;\ge\;V_k(x)-2\tau_k(x).\end{aligned}$$

If $V_k(x)>2\tau_k(x)$, the angular winner is the unique Hamming-vote winner, under any tie rule. Pointwise, $\{h_k(X)\ne Y\}\subseteq\{a_k(X)\ne Y\}\cup\{V_k(X)\le 2\tau_k(X)\}$, and taking probabilities gives Equation (15). □

The two sets differ by t_k deletions and t_k insertions, each changing one class count by one, so $\tau_k\le t_k$; the weaker condition $V_k>2t_k$ therefore also preserves the vote, but charges every swap, whether or not it changes the class counts. Section I.2 compares the two on trained networks.

H.5 A certificate on held-out data

Proposition H.3 (Certificate after training)**.** *Let $(X_j,Y_j)_{j=1}^N$ be independent draws from the test distribution, independent of the trained network, the training set and the choice of this diagnostic, and let $\widehat{U}=\frac{1}{N}\sum_j\mathbf{1}\{a_k(X_j)\ne Y_j \text{ or } V_k(X_j)\le 2\tau_k(X_j)\}$. For any $\delta\in(0,1)$, with probability at least $1-\delta$ over these draws, $\mathcal{R}(h_k)\le\min\big\{1,\widehat{U}+\sqrt{\log(1/\delta)/(2N)}\big\}$.*

Proof. Conditional on the trained network and training set, the summands are independent Bernoulli variables with mean u, and Equation (15) gives $\mathcal{R}(h_k) \le u$. The one-sided Hoeffding inequality gives $\Pr[u - \widehat{U} > \varepsilon] \le e^{-2N\varepsilon^2}$; set $\varepsilon = \sqrt{\log(1/\delta)/(2N)}$. □

For simultaneous use with J predetermined candidates, replace δ by δ/J. Test images that were used to design the diagnostic do not give this guarantee.

H.6 What the code must retain

For the fixed code map let $\eta_c(s) = \Pr[Y = c \mid \mathbf{S}^{(L)}(X) = s]$. The code's Bayes error is $\mathcal{R}_S^* = \mathbb{E}\big[1 - \max_c \eta_c(\mathbf{S}^{(L)}(X))\big]$.

Proposition H.4 (Code information and reader excess)**.** *For any classifier g that depends on the input only through the code,*

$$\mathcal{R}(g) = \mathcal{R}_S^* + \mathbb{E}\Big[\max_c \eta_c\big(\mathbf{S}^{(L)}(X)\big) - \eta_{g(X)}\big(\mathbf{S}^{(L)}(X)\big)\Big],$$

and both terms are nonnegative.

Proof. Conditioning on the code gives $\mathcal{R}(g) = \mathbb{E}[1 - \eta_{g(X)}(\mathbf{S}^{(L)}(X))]$; add and subtract $\mathbb{E}[\max_c \eta_c]$. □

A low code Bayes error is therefore necessary for any sign-only classifier to be accurate, and sign-kNN additionally needs a small reader excess. Appending bits to a fixed code cannot raise its Bayes error, but this does not make finite-bank sign-kNN monotone in the number of bits, and it does not compare separately trained networks of different widths.

H.7 A finite-bank condition in terms of the code geometry

The following bound makes the neighbourhood requirements explicit. It assumes that the code map is fixed independently of the labelled bank, as for a fresh bank or a sample split; it does not apply unchanged to the training set that trained the representation. For a code s let $\mathcal{B}(s, r)$ be its closed ball in normalised Hamming distance, and let P_S be the distribution of the code. Assume that for a set G of query codes with $P_S(G) \ge 1 - \beta$, each $s \in G$ has a radius $r(s)$ with

$$P_S\big(\mathcal{B}(s, r(s))\big) \ge \alpha \qquad \text{and} \qquad \eta_{a^*(s)}(s') - \eta_c(s') \ge \gamma \quad \text{for all } c \ne a^*(s), \tag{16}$$

for P_S-almost every $s' \in \mathcal{B}(s, r(s))$, where $a^*(s)$ maximises $\eta_c(s)$ and $\alpha, \gamma \in (0, 1]$. The first condition is bank coverage, the second a consistent class preference throughout the neighbourhood the reader reaches.

Proposition H.5 (Coverage and local class margin)**.** *Let the bank consist of m independent labelled examples, independent of the fixed code map, and let h_B be its sign-kNN reader. If $m\alpha \ge 2k$, then*

$$\mathbb{E}_B[\mathcal{R}(h_B)] \;\le\; \mathcal{R}_S^* + \beta + e^{-m\alpha/8} + (C-1)e^{-k\gamma^2/2}. \tag{17}$$

Proof. Fix $s \in G$. The number of bank codes in $\mathcal{B}(s, r(s))$ is binomial with mean $\mu \ge m\alpha \ge 2k$, so a multiplicative Chernoff bound gives fewer than k of them with probability at most $e^{-\mu/8} \le e^{-m\alpha/8}$. If at least k codes lie in the ball, every selected neighbour lies in it. Conditional on the bank codes, their labels are independent and the selection depends only on the codes. For each competitor $c \ne a^*(s)$, the variables $\mathbf{1}\{Y_i = a^*(s)\} - \mathbf{1}\{Y_i = c\}$ over the k selected neighbours lie in $[-1, 1]$ and have conditional mean at least γ, so their sum is non-positive with probability at most $e^{-k\gamma^2/2}$. A union bound over the $C - 1$ competitors, counting ties as failures, and the mass β outside G give $\Pr[h_B(S) \ne a^*(S)] \le \beta + e^{-m\alpha/8} + (C-1)e^{-k\gamma^2/2}$. By Proposition H.4, the reader excess is at most this disagreement probability; adding $\mathcal{R}_S^*$ proves the claim. □

Relation to standard results. The concentration steps use Hoeffding's inequality [Hoeffding, 1963]. The identity between the collision probability of random hyperplanes and the angle is the standard random-hyperplane construction [Goemans and Williamson, 1995, Charikar, 2002]. The coverage and local-margin conditions specialise the familiar neighbourhood analysis of nearest-neighbour classification [Chaudhuri and Dasgupta, 2014]. The bounds are proved in full to make their assumptions explicit; they are not a new convergence rate for learned representations.

I Evidence for the Theory of Section 7

I.1 Networks and protocol

The FFNN grid. Figure 6a and d and the comparison with random hyperplanes use a retrained copy of the CIFAR-10 SiLU FFNN grid of Figure 2a: 112 checkpoints, eleven widths (4–4096) and seven depths (1, 2, 3, 4, 6, 8, 12) at seed 0, plus seed 1 for widths up to 64. The bank is the 50,000 training images, the queries are the 10,000 test images, and $k = 15$, with ties broken by bank index and, in the vote, by the lowest class. The curves in Figure 6a and d are single networks (width 1024, seed 0), without confidence intervals.

Random hyperplanes. The hyperplanes of Figure 6a have Gaussian directions and offsets at bank quantiles matched to the activities of randomly sampled units of the layer; errors average three draws for $n \leq 64$ and use one draw for $n \geq 128$. The reference is 8,192 such hyperplanes, a finite high-resolution reader. Because these hyperplanes have offsets, their separation probability is not the angle divided by π. Across the grid, the network's own units give sign-kNN errors within 0.02 of this reader at matched activity in 65 of 70 networks of width at least 32. At width 1024, the extra error of 1,024 random hyperplanes over 8,192 is 1.49, 2.25, 1.57, 0.82, 0.50, -0.16 and -0.14 points at depths 1, 2, 3, 4, 6, 8 and 12.

Class contrast. The class contrast is the centroid-scatter ratio of the 0/1 code on training images,

$$r = \frac{S_B}{S_W}, \qquad S_B = \sum_c p_c \|\mu_c - \mu\|^2, \qquad S_W = \frac{1}{m}\sum_i \|s_i - \mu_{y_i}\|^2,$$

with μ_c the mean code of class c, p_c its share and μ the overall mean code. It is a training-set statistic, not a held-out local margin. Figure 6b uses SVHN SiLU FFNNs of width 1024 at depths 4 and 12 (one seed each; test accuracy 0.80 and 0.82) on 70,000 SVHN training images; the initialisation curve is the depth-12 network before training. At depth 12 the contrast is 0.0026 at layer 1, rises at every layer to 5.5 at layer 10 and ends at 4.0; before training it stays between 0.0010 and 0.0024. The same holds on CIFAR-10 in a separate cohort of depth-8 SiLU FFNNs at widths 256, 1024 and 2048 with three seeds each, whose head initialisation, normalisation and training precision differ from the grid: at width 1024 the mean contrast is 0.0405 and 0.0403 at layers 1 and 2 and rises at every later layer, to 0.666 at layer 8, while at initialisation it falls from 0.044 to 0.022. Across 42 trained FFNNs of depth at least 6 on CIFAR-10 and SVHN, including 23 SVHN networks trained with μP at learning-rate multipliers 1–64, $\ln r$ is linear in depth with $R^2 \geq 0.9$ in all 42 over the hidden layers before the last, and in 39 of 42 when the last hidden layer is included. The median of the pooled ratios between successive interior layers is 1.95, against 0.98 at initialisation; it is not a single growth rate shared by all layers.

Class margin in ResNet-18s. Figure 6c reads the code at the 17 gates of the twelve CIFAR-10 ResNet-18s of Section 5 (four activations, three seeds): the stem output and the two pre-activations of each residual block, over all channel–position units, with a bank of 5,000 training images and 1,000 test queries. For a query let d_{same} and d_{other} be the Hamming distances to the nearest bank code of its own class and of any other class. The median of $(d_{\text{other}} - d_{\text{same}})/(d_{\text{other}} + d_{\text{same}})$, averaged over the twelve networks, is -0.015 at the stem, turns positive at gate 7 and rises to 0.281 at gate 16; most of the rise comes in the last four gates (0.024, 0.040, 0.067, 0.150, 0.281 from gate 12 on). With the entries of every weight tensor shuffled it stays between -0.041 and -0.019 at all gates. Values below zero are expected without class structure, because the other nine classes hold most of the bank.

Neighbourhood labels. At width 1024, the median largest angular rank among the 15 Hamming neighbours is 68 at depth 1 and 36 at depth 12. The mean same-class fraction at those ranks is 0.2806 and 0.5352, and at angular rank 1 it is 0.4593 and 0.5409. These profiles average over queries and do not estimate each query's vote margin. For the same two networks the mean number of swapped neighbours is $t = 5.70$ and 4.22, and the mean net class-count change is $\tau = 2.68$ and 0.57.

Shallow and deep networks. Mean signed differences in percentage points over two groups of the grid (seed 0). "Own" is the network's own code, "8,192" the 8,192-hyperplane reader, "angular" the angular 15-NN vote.

Group	Nets	Own−network	8,192−network	Own−8,192	Angular−network
$L \le 2,\ d_L \ge 512$	8	$+4.57$	$+4.56$	$+0.01$	$+4.24$
$L \ge 4,\ d_L \ge 256$	20	-0.83	-1.35	$+0.52$	-1.23

The groups vary width as well as depth, so the table is descriptive. Close errors do not imply that the selected neighbours or predictions coincide.

I.2 Tightness of the vote-stability bound

We evaluate the upper loss $\widehat{U}_\tau$ of Equation (15) on the 112 grid networks and on the twelve SVHN FFNNs and twelve CIFAR-10 ResNet-18s of Section 5 (four activations, three seeds each), with 10,000 test queries per network and $k = 15$. For SVHN the bank is a seeded subset of 50,000 of the 73,257 training images and the queries a seeded subset of 10,000 test images. For comparison, $\widehat{U}_t$ charges every swapped neighbour ($V \le 2t$ in place of $V \le 2\tau$). The test images were used during the analysis, so these numbers describe the bound's tightness; they are not a fresh certificate in the sense of Proposition H.3. In every query, the implication of Proposition H.2 holds and the sign-kNN error lies below both upper losses.

Family	Nets	Sign error (%)	$\widehat{U}_t$ (%)	$\widehat{U}_\tau$ (%)	Within 5 pp (t / τ)	Vacuous (t / τ)
CIFAR-10 FFNN grid	112	60.80	92.70	79.71	2 / 9	37 / 1
SVHN FFNNs	12	19.22	41.10	26.09	0 / 1	0 / 0
CIFAR-10 ResNet-18	12	8.01	28.65	9.02	0 / 12	0 / 0

Errors and upper losses are network means; "within 5 pp" counts networks whose slack $\widehat{U} - \widehat{\mathcal{R}}(h_k)$ is at most five percentage points, and "vacuous" means $\widehat{U} = 1$. Charging only the net class-count change tightens the bound where binarisation leaves the vote nearly unchanged. On the ResNets the net class-count change is 4.3% of the swapped neighbours ($\sum \tau / \sum t$; mean $t = 5.79$, $\tau = 0.25$), the median slack falls from 23.61 to 1.04 points, and all twelve networks lie within 0.79–1.28 points. On the SVHN FFNNs the ratio is 24.1% and the median slack 7.36 points; on the CIFAR-10 grid it is 38.6% and 20.51 points, dominated by narrow and shallow networks. The nine grid networks within five points all have depth at least 6 and width at least 256; the single vacuous case has width 4. The bound's tightness establishes the stability of the vote, not the stronger majority margin of Proposition 7.1.

The looseness has an exact form, $\widehat{U}_\tau - \widehat{\mathcal{R}}(h_k) = \frac{1}{N}\sum_j \mathbf{1}\{V(X_j) \le 2\tau(X_j),\ h_k(X_j) = Y_j\}$, and likewise for t. The t criterion charges every swapped neighbour its worst possible effect, even when the replacement has the same class; since $V \le 15$, a query with eight or more swaps can never be certified under t. For comparison, the direct union $\Pr[a_k(X) \ne Y \text{ or } h_k(X) \ne a_k(X)]$ has network means 67.93%, 20.83% and 8.51% for the three families.

J Limits of the Argument

Section 7 explains when the last-layer code can be read: the classes must be separated in the layer (a), and the code must resolve that separation (b). This appendix collects where the explanation stops.

Width and depth are not guarantees. Proposition 7.1 needs a positive margin at the test point and a code fine enough to resolve it. Neither follows from width or depth alone, and both fail in some wide and deep networks. Six SVHN MLPs of widths 256–1024 trained at high learning rates keep sign gaps of 0.037–0.068. Their last layers are sparse: when each unit is active on only a few inputs, few units separate any two inputs, and the code resolves the layer more coarsely than its width suggests. The dense width-1024, depth-6 ReLU FFNNs of Section 5 keep gaps of 0.035, 0.038 and 0.040 over three seeds, although the angular vote is close to the network head (mean difference about -0.002); for the same-size SiLU networks the sign gap is about 0.043 and the angular gap about 0.039. Bank size, k, the activation and the training recipe can all change the reader's error at the same width and depth.

Adversarial situations. Each of the two properties can be broken on purpose.

- *The margin.* Depth provides the margin because training separates the classes, so an objective that works against this removes it at any depth. The repulsion loss $\mathcal{L}_{\text{rep}}$ of Section 6 is such an objective: it keeps the training fit, but same-class neighbours no longer share their code, and test accuracy collapses together with the code's vote Q. Without training the margin does not grow either: with shuffled weights the class margin of Figure 6c stays below zero at every gate.
- *The resolution.* Part (ii) of Proposition 7.1 holds with high probability for random hyperplanes, not for every set of n hyperplanes. A worst-case set can miss the margin entirely: if all hyperplanes are parallel, the code only orders inputs along one direction, and two inputs that differ in any other direction share their code at every n. Trained units are not random; that they read their layer like random hyperplanes is a measurement on the networks of Section I, not a theorem.
- *The test point.* The guarantee holds at each test point with $g(x) > 0$ and says nothing about points with $g(x) \leq 0$. An input moved towards another class, for instance by an adversarial perturbation, can lose its margin in the layer, and then the guarantee no longer applies; the vote may still be right, since a plurality can win, but if the layer's own neighbourhood of the input votes for the wrong class, no width helps. The argument gives no robustness guarantee.

What the argument does not show.

- *That the code keeps everything the network knows.* No sign-only reader can beat the code's Bayes error $\mathcal{R}_S^*$ (Proposition H.4). Proposition H.5 bounds the finite-bank reader by bank coverage and a local class margin, and Proposition H.3 turns the vote-stability condition into a certificate on held-out data.
- *That the network computes with its code.* Rotating one hidden layer's units within their span, $\mathbf{z} \mapsto Q^\top \varphi(Q\mathbf{z})$ for a random orthogonal Q, keeps every distance and angle of the layer. Across 234 layers (108 FFNN layers, 24 ResNet-18 blocks and 102 ViT MLP blocks), sign-kNN on the rotated code is never worse than on the native code by more than 0.02, while the network's test error rises by a median of 0.18–0.26: a code can be readable without being the one the network uses.
- *Why training separates the classes.* We measure that depth provides the margin in the networks we study, but we do not derive it from the training dynamics.
- *Other settings.* The measurements cover FFNNs on CIFAR-10 and SVHN and ResNet-18s on CIFAR-10, with $k = 15$ and the training set as the bank.

K Activation Asymmetry and the Choice of Repulsion Proxy

The repulsion experiments of Section 6 use a sigmoid gate proxy for ReLU-family activations and a tanh sign proxy for tanh networks. This appendix explains the choice through the magnitude asymmetry of each activation.

K.1 The magnitude asymmetry ratio

For neuron j with sign S_j and magnitude M_j (Definition 2.2), let $\mu_j(s) = \mathbb{E}[M_j \mid S_j = s]$ be its mean magnitude given the sign. For any neuron with $\mu_j(+1) + \mu_j(-1) > 0$, define

$$\delta_j \;:=\; \frac{\mu_j(+1) - \mu_j(-1)}{\mu_j(+1) + \mu_j(-1)} \;\in\; [-1,\, 1], \tag{18}$$

and set $\delta_j = 0$ for dead neurons with $\mu_j(+1) = \mu_j(-1) = 0$. The ratio measures how strongly a neuron's expected magnitude depends on its sign. Three regimes arise:

1. **ReLU** ($\delta_j = 1$ exactly). Inactive neurons have $\mu_j(-1) = 0$; the sign bit completely determines whether the neuron contributes.
2. **SiLU / GELU** (δ_j large and positive empirically). The negative side is attenuated but not zeroed. Whether $\mu_j(-1) \ll \mu_j(+1)$ holds in a given network depends on the pre-activation distribution, not only on the activation.

3. **Tanh** ($\delta_j \approx 0$ empirically). Because $\tanh$ is odd, $|\tanh(z)| = |\tanh(-z)|$ for all z, and under the approximately symmetric pre-activation distributions of trained networks this yields $\mu_j(+1) \approx \mu_j(-1)$. This is an empirical statement about trained networks, not a consequence of oddness alone.

Across 120 width–depth configurations of CIFAR-10 FFNNs (widths 4–4096, depths 1–12, two seeds), tanh networks have median $|\mu_j(+1) - \mu_j(-1)|$ of 0.0014, with 95 of the 120 cells at or below 0.005, while SiLU networks have a layer-mean asymmetry $\bar{\delta}$ above 0.9 at small widths and of about 0.4–0.7 at large widths and depths.

K.2 Choice of repulsion regularizer: gate proxy vs. sign proxy

The repulsion loss $\mathcal{L}_{\text{rep}}$ (Eq. 9) is designed to destroy same-class sharing of the *dominant computational pathway*. Which pathway is dominant depends on the activation's asymmetry profile, so the correct continuous proxy for the regularizer depends on the activation.

ReLU, SiLU, and GELU: sigmoid gate proxy

For these activations, δ_j is large and positive: neurons with $S_j = +1$ contribute far more strongly than neurons with $S_j = -1$. The dominant computational pathway is therefore the *active set* $\{j : g_j = 1\}$, where $g_j = \mathbf{1}[z_j > 0]$. We regularise the continuous gate proxy

$$g(x) \;=\; \sigma\big(z(x)/\tau\big) \;\in\; (0,1)^D, \tag{19}$$

which for small temperature τ approximates the binary gate. Penalising $\big(g(x_i)^\top g(x_j)\big)^2$ for same-class pairs directly drives their active-set overlap toward zero.

What gate repulsion targets. Gate repulsion and sign repulsion penalise different things, and neither implies the other. Two inputs with D neurons, each with roughly $D/2$ active, can have *orthogonal* sign codes in $\{-1,+1\}^D$ while still sharing approximately $D/4$ co-active neurons; conversely, two inputs with no co-active neuron can have strongly correlated sign codes. Sign-code orthogonality treats agreements on the active and the inactive side symmetrically, but only the active side carries substantial forward-pass signal when δ_j is large. Gate repulsion targets that side: it drives the co-active count of same-class pairs towards zero. That it is the stronger intervention for non-odd activations is an empirical finding, below.

Empirical confirmation. On SVHN (FFNN, width 1024, depth 8, SiLU), sigmoid-gate repulsion achieves the intended effect: training accuracy exceeds 97%, test accuracy collapses to approximately 40%, and leave-one-out purity at $k{=}5$ drops to 0.34. Matched sign-based repulsion at the same nominal strength either destabilises training (SiLU/GELU/ReLU, due to steeper gradients from the tanh proxy) or leaves enough active-set overlap for partial generalization to survive.

K.3 Tanh networks: sign proxy

Tanh is the important exception. Because $\delta_j \approx 0$ in trained tanh networks, the forward pass does not privilege an active set: positive-sign and negative-sign neurons contribute with comparable magnitude. A sigmoid gate proxy would impose an artificial positive-vs-nonpositive split that is not aligned with the tanh network's geometry.

For tanh models we therefore use the sign proxy

$$q(x) \;=\; \tanh\big(z(x)/\tau\big) \;\in\; (-1,1)^D, \tag{20}$$

which for small τ approximates the binary sign code $S_j = \operatorname{sgn}(z_j) \in \{-1,+1\}$. This is the correct matched intervention: when the activation treats both sides approximately symmetrically, the relevant binary structure is the sign code itself, not a privileged active set.

Consistency with the rest of the paper. The distinction between gate proxy and sign proxy arises *only* in the training regularizer. All measurements (sign-kNN, Q, purity) and all causal interventions (sign flip, magnitude replacement) are reported throughout the paper in the common binary sign-code

framework $S_j \in \{-1, +1\}$. The two frameworks are related by the bijection $S_j = 2g_j - 1$; the gate representation $g_j \in \{0, 1\}$ appears only inside the regularizer where it proved the stronger intervention for non-odd activations.

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 abstract and Introduction state the paper's three core claims in language that matches the empirical and theoretical evidence in Sections 3–7, and Section 8 narrows the scope to the studied activations, architectures, and datasets.

   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: The paper discusses its main limitations, including that width and depth do not guarantee sign-code sufficiency, that the proved part of the theory concerns random hyperplanes, the restriction to sign-preserving activations, and the limited experimental scope; see Section 8 and Section J.

   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 main text states the sign–magnitude decomposition (Definition 2.2) and the margin-and-resolution proposition (Proposition 7.1) with its assumptions; Section H gives the full proofs and the further results, and Section J states what the argument does not cover.

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 appendix specifies the sign-code extraction rules for FFNN/ResNet/ViT, the sign-kNN protocol, the repulsion training procedure, the architecture grids, and the relevant hyperparameters; see Appendix D, Appendix E.4, and Appendix E.1.

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 available at `https://github.com/NiclasGoering/sign-code` (Section 8).

   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: The main text describes each experimental design, and Appendix E.1 specifies datasets, architecture grids, preprocessing, optimizers, schedules, stopping rules, corruption grids, and regularizer hyperparameters.

   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]

   Justification: Experimental results in this work are accompanied by error bars, despite they might not be clearly visible due to highly converging values over multiple random seeds.

   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: Appendix E.1 reports the overall hardware context and an end-to-end compute estimate.

   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: The work uses standard supervised-learning experiments on public benchmark datasets together with theoretical analysis; the experimental and theoretical scope is described in Sections 3–7 and Appendix E.1, and the authors have reviewed the NeurIPS Code of Ethics.

   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 work primarily focuses on the theoretical understanding of generalization in neural networks.

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 does not release a high-risk generative model, a scraped dataset, or another asset that requires a special controlled-release safeguard; it studies standard discriminative models on established benchmark datasets.

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: [Yes]

Justification: The datasets, pretrained models, and software assets used in the experiments are all openly published version and are all well-cited.

- 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: This work does not release a new dataset, model, or code asset.

    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.