EXPAND NEURONS, NOT PARAMETERS

Anonymous authors
Paper under double-blind review

ABSTRACT

This work demonstrates how increasing the number of neurons in a network without increasing its number of non-zero parameters improves performance. We show that this gain corresponds with a decrease in interference between multiple features that would otherwise share the same neurons. On symbolic tasks, specifically Boolean code problems, splitting each neuron into sparser sub-neurons with knowledge of the clauses systematically reduces polysemanticity metrics and yields higher task accuracy. Notably, even random splits of neuron weights approximate these gains, indicating that reduced collisions, not precise assignment, are a primary driver. Consistent with the superposition hypothesis, the benefits of this framework grow with increasing interference: when polysemantic load is high, accuracy improvements are the largest. Transferring these insights to real models—classifiers over CLIP embeddings, CNNs, and deeper multilayer networks—we find that widening networks while maintaining a constant non-zero parameter count consistently increases accuracy. These results identify an interpretability-grounded mechanism to leverage width against superposition, improving performance without increasing the number of non-zero parameters. Such a direction is well matched to modern accelerators, where memory movement of non-zero parameters, rather than raw compute, is often the dominant bottleneck.

1 INTRODUCTION

Understanding the mechanisms behind neural network performance has become increasingly crucial as these models grow in complexity, scale, and ubiquity. Despite their widespread adoption, neural networks frequently remain opaque due to polysemantic neurons: individual neurons that simultaneously encode multiple features (Goh et al., 2021; Jermyn et al., 2022; Gurnee et al., 2024; Dreyer et al., 2024), frustrating interpretability. This entanglement induces interference among features that are forced to share the same neuron, degrading performance (Lecomte et al., 2023).

Two complementary lines of work highlight this phenomenon. The superposition hypothesis argues that networks represent more features than they have neurons, leading to interference when distinct features coexist within a single unit, initially through empirical findings in toy models (Olah et al., 2020; Elhage et al., 2022; Liu et al., 2025). Recent theoretical analyses formalize the parameter requirements for representing features and show that features often compete at the level of edges, not just neurons (Adler & Shavit, 2024; Hänni et al., 2024; Adler et al., 2025). Meanwhile, the lottery ticket hypothesis (Frankle & Carbin, 2018) and follow-up work (Chen et al., 2020; 2021; Zhou et al., 2019) suggest that sparse subnetworks with the right connectivity can match or exceed dense performance, again pointing to structure, not density, as the key determinant.

Together, these perspectives motivate a concrete question. If features interfere because they are forced to share neurons, what happens if we keep the total number of non-zero parameters fixed but allocate those edges across more neurons? In this work, we demonstrate a striking answer: increasing the number of neurons while keeping the number of non-zero parameters fixed consistently reduces interference and improves performance in interference-limited regimes.

In Boolean settings, where ground-truth feature structure is known, we show that widening the network without adding parameters reliably reduces feature interference. Gram-matrix block structure becomes sharper, feature capacity increases (Scherlis et al., 2022), and activations become more selective. These geometric changes closely mirror improvements in accuracy, revealing a tight interference–performance relationship. A theoretical analysis in this setting explains the effect: dis-

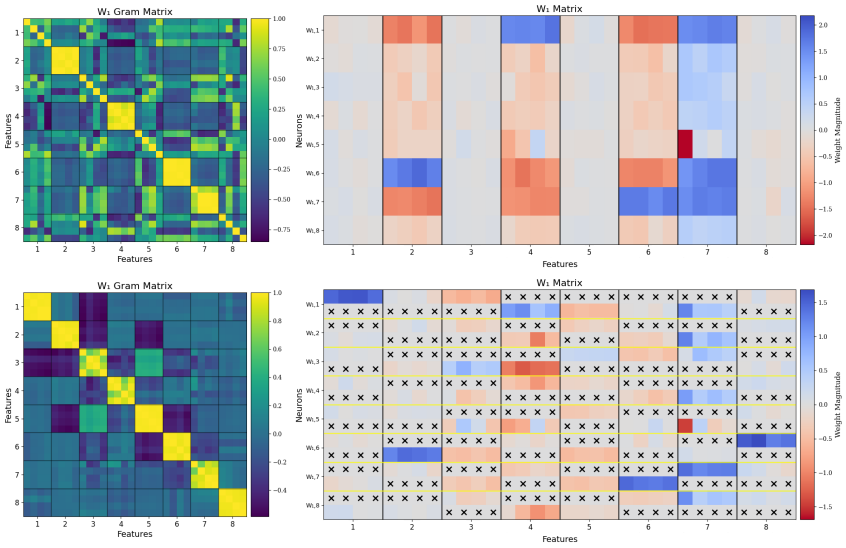

Figure 1: More neurons yields less interference between features. On a 4DNF Boolean task, clear codes are visible for clauses 2, 4, 6, and 7 for the dense model in the top row. For a network with the same number of non-zero parameters but twice as many neurons, clear codes are visible for all clauses in the top row. Black x's represent masked parameters of value 0. Full details in Section 2.4.

tributing edges across more neurons reduces expected feature collisions and preserves coverage, and this holds even for random edge partitions, an empirical pattern that appears consistently in our experiments. We then test the phenomenon beyond symbolic tasks, including Boolean models with learned embeddings, CNN backbones with widened classifier layers, and CLIP-based CIFAR-100 and ImageNet-1k classifiers. Across these settings, increasing neurons while keeping parameters fixed systematically reduces interference in high-superposition regimes and improves performance.

Taken together, our results reveal that neuron count is a fundamental axis of neural network performance, distinct from parameter count. Redistributing a fixed number of parameters across more neurons reliably decreases polysemanticity and improves performance, offering new insight into how representational bottlenecks, and their mitigation, shape neural computation.

2 Methodology

2.1 Theoretical justification for neuronal expansion

To provide a theoretical justification of how increasing width without changing the total number of non-zero parameters reduces interference, we analyze a network with r neurons trained on a Boolean DNF task (O'Donnell, 2014) of m total literals with k literals per clause. In this setting, we consider clauses and features equivalent. The network's r neurons is duplicated into αr sub-neurons while preserving the total number of non-zero weights. This is achieved by sparsifying each sub-neuron to degree $d = \frac{m}{\alpha}$. We find that this preserves clause coverage with high probability while cutting expected clause collisions by $\approx \alpha^{-(2k-1)}$. Intuitively, coverage is easy because many neurons sample the needed literals, while collisions are rare because all $2k$ inputs of two features must land within the same neuron's sparse support. The full justification can be found in Section A.5.

We also situate our setup within the feature channel coding hypothesis (Adler et al., 2025), which complements the superposition view of (Elhage et al., 2022). In this framework, abstract features are implemented by feature channels, which are sets of neurons that share a characteristic weight–sign pattern. Adler et al. (2025) give explicit constructions and combinatorial packing bounds for such codes, showing that with a fixed number of neurons there is a finite capacity for reliably representing Boolean features, and as more features are crammed into the same set of rows, code overlaps and errors must increase. Our analysis of expanding width is aligned with this picture: at fixed non-zero parameter budget, duplicating neurons increases the number of rows available to host feature-

channel codes, which reduces expected overlaps or collisions without sacrificing clause coverage. This perspective both motivates our focus on weight-level geometry, such as through feature capacity and cosine similarity, and explains why even random splitting should systematically reduce interference. Our goal in this section is not to present a complete theory of neuronal expansion, but to show that our analysis sits within a now partly formalized view of superposition. While the original superposition hypothesis was introduced from empirical and toy-model evidence (Elhage et al., 2022), subsequent work on feature-channel codes (Adler et al., 2025) derives explicit combinatorial capacity bounds for the same kind of overlapping codes. Our collision analysis can be read as a simplified instance of this framework: duplicating neurons at fixed non-zero parameter budget increases the number of rows available for such codes, thereby reducing expected overlaps.

2.2 Classifier architecture

To examine the impact of neuron count under a fixed parameter budget, we begin with a fully connected feedforward architecture with a single hidden layer. Let $\mathbf{x} \in \mathbb{R}^d$ be the input, and let h denote the number of neurons. The classifier maps inputs to output logits via the transformation $\mathbf{z} = \mathbf{W_2} \cdot \mathrm{ReLU}(\mathbf{W_1}\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$, where $\mathbf{W_1} \in \mathbb{R}^{h \times d}$ and $\mathbf{W_2} \in \mathbb{R}^{C \times h}$ are the weight matrices of the first and second layers respectively, $\mathbf{b}_1 \in \mathbb{R}^h$, $\mathbf{b}_2 \in \mathbb{R}^C$ are biases, and C is the number of output classes. In binary classification settings, we apply a final sigmoid activation to the output $\mathbf{z}$ and use binary cross-entropy loss. In multiclass classification, we treat $\mathbf{z}$ as unnormalized logits and apply standard softmax-based cross-entropy loss.

This relatively minimal architecture is intentionally underparameterized in terms of neurons to study the emergence of superposition and interference. For ImageNet-1k, we use a deeper five layer network, constructed in the same way. For our experiments with joint feature learning, we either use a simple embedding layer $\mathbf{E}$ for Boolean tasks or a CNN backbone, both of which are trainable and feed into a classifier. Additional details can be found in Sections A.7.1, A.8.

2.3 Tasks and datasets

2.3.1 Symbolic reasoning: monotone read-once DNF satisfiability

To study superposition under precise structural control, we construct a series of classification tasks based on satisfiable monotone read-once Boolean formulas in disjunctive normal form (DNF) (O'Donnell, 2014). Specifically, each formula is a disjunction of conjunctive clauses, each of which has exact four literals. Furthermore, all literals are positive and appear exactly once across the entire formula. For example: $(x_1 \wedge x_2 \wedge x_3 \wedge x_4) \vee (x_5 \wedge x_6 \wedge x_7 \wedge x_8) \vee (x_9 \wedge x_{10} \wedge x_{11} \wedge x_{12}) \cdots$

Each input is a modified binary truth assignment to the full set of variables, and the label indicates whether the assignment satisfies the formula. This formulation creates one distinct, non-overlapping feature per clause, with no interference from variable reuse or negation. By increasing the number of clauses, we increase the number of independent features to be represented, thereby intensifying superposition pressure. Conversely, by reducing the number of hidden neurons in the classifier, we decrease the network's representational capacity. This dual control allows us to induce or relieve superposition either from the feature side (via clause count) or from the neuron side (via layer width), enabling a clear experimental probe of the representational bottleneck. More details in Section A.3.

2.3.2 Visual classification tasks: FashionMNIST, CIFAR-100, and ImageNet

We test our framework on FashionMNIST (Xiao et al., 2017), CIFAR-100 (Krizhevsky et al., 2009), ImageNet-100 (constructed via a random selection of 100 ImageNet classes), and ImageNet-1k (Deng et al., 2009). For FashionMNIST, we use raw flattened grayscale images. For CIFAR-100, we evaluate both on raw flattened images and on features extracted using a frozen CLIP ViT-B/16 encoder (Radford et al., 2021), treating these embeddings as fixed inputs to the classifier. For ImageNet, we similarly use a frozen CLIP encoder for feature extraction. Using frozen embeddings decouples our analysis from representation-learning dynamics and isolates the effect of expansion under a fixed non-zero parameter budget on classification performance. In addition, we include experiments on CIFAR-100 using a CNN backbone trained jointly with the classifier (Section A.7.1), providing evidence that FPE remains effective when representations and the classifier are learned end-to-end. Further training details are provided in Sections A.3, A.7.1.

2.4 A CASE STUDY IN NEURONAL EXPANSION

We study a controlled Boolean reasoning task to illustrate how FPE reduces interference. We train the classifier from Section 2.2 on a DNF formula with 8 clauses of 4 distinct literals each, $\vee_{j=1}^{8} \left(\wedge_{i=1}^{4} x_{4(j-1)+i}\right)$, treating each clause as a feature.

We first train a compact model with 8 hidden neurons, then expand it with $\alpha = 2$ using either clause-based or random splitting (Section A.3), and continue training all models, including the dense one, under the same non-zero parameter budget (Section A.3.3). To analyze the learned representations we examine the first-layer Gram matrix $G = \mathbf{W}_1^\top \mathbf{W}_1$ (Yang & Chaudhari, 2025), which summarizes how first-layer weights span feature space: strong on-diagonal blocks indicate well-isolated clause codes, while large off-diagonal values indicate entanglement.

With 8 neurons, the dense model reaches 78.7% test accuracy, compared to 99.4% for clause-split FPE and 88.7% for random-split FPE. Figures 1 and A2 show the corresponding Gram matrices and $\mathbf{W}_1$ (vertical lines mark clause boundaries; horizontal lines group sub-neurons). The dense model's Gram matrix exhibits clear codes for some clauses but weaker, more diffuse structure for others, consistent with feature interference. Clause splitting produces a strongly block-diagonal Gram matrix and more specialized weights, recovering previously entangled clauses while keeping the parameter count fixed. Random splitting yields weaker block structure, yet still improves over the dense baseline and sometimes approaches clause-split performance.

These observations support the superposition hypothesis: the compact model is limited by neurons sharing multiple features, and increasing width under a fixed parameter budget, especially with structured clause-aware splitting, reduces such collisions and improves performance.

2.5 THE NEURONAL EXPANSION PROCEDURE

In this work, we use the edge-partitioning transformation as an experimental probe of the interference–performance relationship. We refer to this specific instantiation as Fixed-Parameter Expansion (FPE), but our goal is not to introduce a deployment-ready training recipe. Instead, we treat FPE as a controlled intervention that lets us vary neuron count and feature collisions at fixed parameter count and then measure the resulting changes in both internal geometry and task performance.

Let the baseline dense model have hidden width h, input size d, and output size C, as defined in Section 2.2. For an integer expansion factor $\alpha > 1$, we define the expanded width $h' = \alpha h$, and construct a new weight matrix $\mathbf{W_1'} \in \mathbb{R}^{h' \times d}$ with a binary sparsity mask $\mathbf{M_1} \in \{0,1\}^{h' \times d}$. For each original neuron n_i with weight vector $\mathbf{w_i}$, we duplicate $\mathbf{w_i}$ across α sub-neurons $n_{\alpha i:\alpha(i+1)}$ in $\mathbf{W_1'}$. The input dimension is partitioned into α disjoint masks $m_{(i_k)} \in \{0,1\}^d$ such that $\sum_{k=1}^{\alpha} m_{(i_k)} = \mathbf{1}_d$, and each mask is applied to one sub-neuron. This ensures no sub-neurons share input features, and the total number of non-zero parameters in $\mathbf{W_1'}$ equals that in $\mathbf{W_1}$. If biases are used, we copy the original bias to all sub-neurons to form $\mathbf{b_1'}$.

To handle the wider hidden layer, we use a "re-sparsification" strategy: We expand $\mathbf{W_2} \in \mathbb{R}^{C \times h}$ to $\mathbf{W_2'} \in \mathbb{R}^{C \times h'}$ by duplicating each original output weight vector $\mathbf{w_j}$ across α sub-features. $\mathbf{b_2'}$ is directly copied from the original. This results in $(\alpha - 1)C$ excess parameters. To preserve parameter count, the smallest-magnitude weights in $\mathbf{W_1'}$ and $\mathbf{W_2'}$ are pruned, and masks $\mathbf{M_1}$ and $\mathbf{M_2}$ are updated accordingly. The masks are not updated following initialization. This process is extended to deeper networks for our ImageNet-1k experiments, and either a simulated embedding layer $\mathbf{E}$ or a CNN backbone in prepended to this classifier for our joint representation learning and classification experiments. More in Sections A.7.1, A.8, including experimenting with mask updates.

The masks $m_{(i_k)}$ are constructed either randomly or using feature-based grouping. In Boolean tasks, clause-aware splitting assigns all inputs from a clause to the same sub-neuron. In vision tasks, we emulate group structure by observing the Gram matrix (Section A.2, 3.2). In all subsequent experiments, the dense classifier is first trained for 25 warmup epochs, during which it closely approaches convergence and learns stable mixed features. We then apply FPE to this near-converged model, creating the expanded architecture under the same parameter budget. After this intervention, both the dense and FPE models are fine-tuned for an additional 25 epochs under identical training settings (Section A3). This procedure allows us to evaluate whether increasing neuron count alone, without increasing parameter count, can reduce superposition interference and improve performance.

3 RESULTS

3.1 SYMBOLIC REASONING TASKS: BOOLEAN SATISFIABILITY

We train a classifier on a 4DNF formula while varying the number of neurons and clauses the dense network is trained on. We expand the dense models to an expansion factor $\alpha = \{2, 4\}$, and evaluate the performance of a dense model, a clause-split model, and a random-split model.

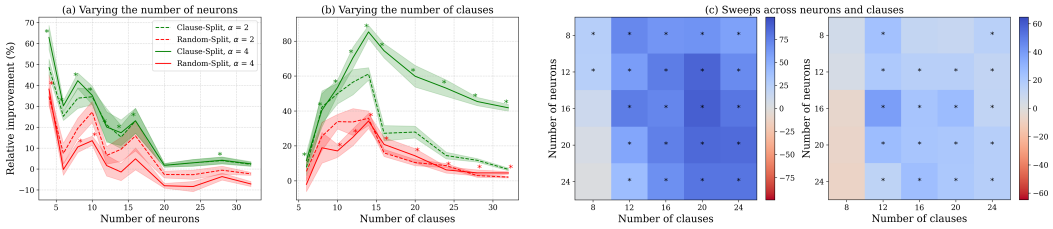

Figure 2: Trends of performance under superposition in neurons and clauses. (a) Relative improvement in test accuracy for models of varying hidden dimension on 8 clauses. (b) Relative improvement in test accuracy for models with 8 neurons. y-axis labels are shared for (a) and (b). (c) Heatmap of relative improvement percentage for clause-split split FPE models (top) and random-split FPE models (bottom). Relative improvement is calculated as $\frac{\text{FPE test accuracy} - \text{dense test accuracy}}{\text{dense test accuracy}}$. Error bars indicate one standard error of the mean. * indicates $p < 0.05$ and is shown only for $\alpha = 4$ for clarity. Results collected over five trials.

Effect of Neuron Count Increasing the number of neurons leads to diminishing returns in relative improvement for learning 8 clauses (Figure 2a). This is consistent with our expectation: with fewer neurons for the same number of features, superposition-derived interference increases, which neuron splitting via FPE effectively reduces. Clause splitting consistently outperforms random splitting, although the latter still often outperforms the dense baseline.

Effect of Clause Count As the number of clauses increases for a network with 8 neurons, performance gains from FPE initially rise, reflecting increased feature entanglement in the dense model. Again, clause-split yields higher improvements than random-split. However, beyond roughly 16 clauses, the benefit of splitting begins to plateau (Figure 2b). This suggests that a network's capacity to fully disentangle complex features is eventually saturated even with more neurons.

Scaling Behavior Figure 2c shows heatmaps of performance improvement, highlighting the joint effects of clause count and neuron count. These results support earlier observations: as the number of clauses decreases, the benefit of FPE diminishes, reflecting reduced superposition pressure in simpler settings. We also observe a plateau in performance gains for models with 8, 12, and 16 neurons, suggesting that beyond a certain point, even with splitting, the model struggles to fully disentangle the clause structure. Interestingly, we observe that randomly splitting provides a modest increase in performance across many configurations. This suggests that suboptimal splitting strategies may still provide meaningful improvements. While clause-based splitting consistently achieves the best results, randomly splitting still enhances performance in these theoretical settings, highlighting the general effectiveness of increasing neuron capacity under a fixed parameter budget.

Direct Evidence of Feature Disentanglement To empirically show that FPE reduces feature entanglement within the neural network, we measure superposition on the Boolean task using feature capacity analysis (Scherlis et al., 2022) and orthogonality.

For each feature (clause), its capacity C_i is calculated as a measure of interference. The capacity allocated to feature i is defined as: $C_i = \frac{(W_{\cdot,i} \cdot W_{\cdot,i})^2}{\sum_j (W_{\cdot,i} \cdot W_{\cdot,j})^2}$, where W are the activations of the model (which in our Boolean setting is the same as the weight matrix) and $W_{\cdot,i}$ is the activations of the ith feature. C_i is the fraction of the dimension allocated to representing feature i, with the numerator representing the squared magnitude of feature i's weight vector, and the denominator accounting for the total overlap of feature i with all other features, thus measuring how much of the representational space is uniquely dedicated to feature i versus shared with other features. High capacity is indicative

of lower polysemanticity, as neurons are more dedicated to representing single features. We then sum all individual feature capacities to measure overall capacity.

In terms of orthogonality, neuron weight vectors should also be increasingly orthogonal to one another to reduce polysemanticity. The mean cosine similarity is calculated by averaging the cosine similarity between all pairs of distinct neuron weight vectors. Lower similarity is indicative of lower polysemanticity, as features are represented more independently.

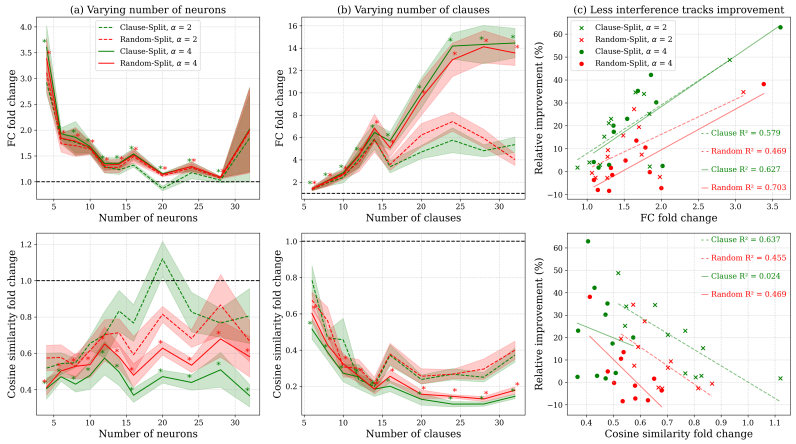

Figure 3: Changes in feature interference metrics for varying neurons and clauses. (a) Feature capacity (top) and cosine similarity (bottom) fold change for models of varying hidden dimension on 8 clauses compared to baseline. (b) Feature capacity (top) and cosine similarity (bottom) fold change for models of varying number of clauses for 8 neurons compared to baseline. y-axis labels are shared for (a) and (b) and all metrics are normalized to dense values. A fold change of 1.0 represents dense metrics and is shown by a black dashed line. (c) Least-squares regressions on relative improvement versus feature capacity fold change (top) and neuron cosine similarity fold change (bottom) when varying neurons, with the coefficients of determination indicated. Dotted lines correspond to $\alpha = 2$ and solid lines correspond to $\alpha = 4$. Relative improvement is calculated as before. Error bars indicate one standard error of the mean. * indicates $p < 0.05$ and is shown only for $\alpha = 4$ for clarity. Results collected over five trials.

Feature Capacity (FC) and Cosine Similarity Fold Change The top panels of Figure 3a and b demonstrate that FPE consistently increases feature capacity, thus reducing interference, compared to dense baseline models (black dashed line). The benefits when varying neurons are most pronounced in resource-constrained settings, where the dense model struggles with severe superposition. A complementary pattern is observed when varying the number of clauses with 8 neurons fixed. The bottom panels of Figure 3a and b examine neuron cosine similarity, where lower values indicate better feature disentanglement. FPE methods achieve superior orthogonality compared to dense models for all but one setting. Additional experiments in Section A.6.

Regressions Improvements in performance and interference track strongly while varying width. This suggests that reduced feature interference (more capacity and orthogonality) directly correlates with better accuracy at a fixed parameter count, providing insight into why FPE works (Figure 3c).

Clause Recovery and Circuits Beyond these global metrics, we also perform explicit circuit and clause level analyses: clause-split FPE models exhibit lower edge-level collision rates and substantially better recovery of the ground-truth clause partition via Gram-matrix clustering than dense baselines, providing more direct evidence of semantic disentanglement (Section A.9).

3.2 Generalization to Real-World Vision Tasks

To assess the generalizability of FPE, we evaluate its performance on four multiclass classification tasks of increasing complexity: FashionMNIST, CIFAR-100, ImageNet-100, and ImageNet-1k. We use the single hidden layer classifier (Section 2.2) for the first three tasks, and use a five layer

classifier for ImageNet-1k (Section A.8). Inspired by the Gram matrix in the Boolean DNF case, we implement a similar feature-clustering algorithm. As described in Section 2.5, we first train a dense baseline model for 25 epochs to allow for near convergence and compute the feature Gram matrix of $\mathbf{W_1}$. Finally, we cluster the rows of the Gram matrix and treat the clusters as groups of features that should remain together, as is in the case of clauses. We then balance the number of clusters that go to each sub-neuron, and fine-tune for 25 more epochs for both the baseline and the FPE model.

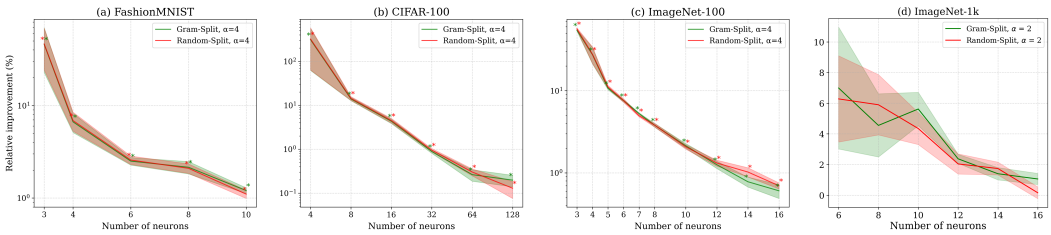

Figure 4: Fixed Parameter Expansion helps on real datasets like (a) FashionMNIST, (b) CLIP-embeddings of CIFAR-100, (c) CLIP-embeddings of ImageNet-100, and (d) CLIP-embeddings of ImageNet-1k. In all figures, the number of neurons refers to the number of neurons pre-expansion. Relative improvement is calculated as before. Error bars indicate one standard error of the mean. * indicates $p < 0.05$. Results collected over ten trials.

We observe several notable trends across FPE performance. First, feature-based splitting consistently outperforms the dense baseline across various hidden sizes, affirming the overall effectiveness of the FPE framework. Indeed, in some scenarios, we can double dense performance (Figure 4b), greatly improving accuracy per parameter. However, we find that our feature-based splitting method performs only comparably to random splitting in practice, even though both improve over the dense baseline. This is similar to what we observe in the Boolean DNF experiments (Sections 2.4, 3.1), where random splitting still provides some gains over the dense baseline.

This consistency across synthetic and real-world settings suggests that FPE confers benefits even when the splitting heuristic does not perfectly align with the underlying structure. It reinforces the idea that increasing representational disentanglement, with or without feature understanding, can mitigate superposition and improve performance, as suggested by our theoretical analysis (Section 2.1). However, it also suggests that the feature-based splitting algorithm likely did not recover the "true" underlying feature structure of the data. Developing more precise feature attribution or disentanglement methods remains an important avenue for future work. Additionally, the relative improvement of FPE is most pronounced when there are fewer neurons. This too aligns with the hypothesis that FPE is particularly beneficial in settings with high superposition, where neurons are forced to represent multiple overlapping features. As the network widens and has more capacity, the gains from expansion diminish due to decreasing superposition. Exact test accuracies are provided in Section A.7.

3.3 Joint representation and classifier learning

To evaluate whether FPE's interference reducing effects extend beyond frozen representation settings, we also consider regimes where features and the classifier are learned jointly. Specifically, we test two avenues. First, in the Boolean setting, we prepend a learnable linear embedding layer $\mathbf{E}$ before the classifier so that both the embedding and classifier weights update during training. Second, for CIFAR-100, we train a lightweight CNN backbone jointly with a classifier MLP, where the backbone maps each image to an embedding vector that is fed into the MLP. In both cases, the dense model is pre-trained for 25 epochs, then expanded via FPE, and both dense and FPE variants are fine-tuned for an additional 25 epochs using identical optimization schedules. Unlike the CLIP-based experiments, the feature embeddings here remain fully trainable throughout, allowing us to test whether FPE continues to improve performance when upstream representations are also evolving. More details in Section A.7.1.

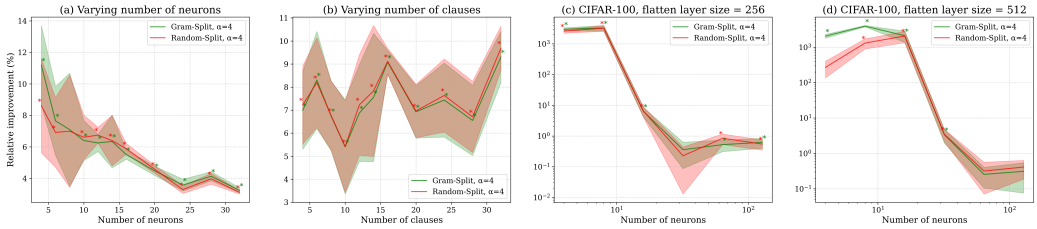

Figure 5: Jointly learning and classifying features is improved via increasing width. (a) Training a classifier on a 4DNF Boolean task with eight clauses while varying the number of neurons and with an embedding layer. (b) Training a classifier on a 4DNF Boolean task with eight neurons while varying the number of clauses and with an embedding layer. Training a CNN on CIFAR-100 with an embedding dimension of 256 (c) and 512 (d). Relative improvement is calculated as before. Error bars indicate one standard error of the mean. * indicates $p < 0.05$. Results collected over ten trials.

In the Boolean setting with a learnable embedding, FPE continues to follow the superposition story observed in earlier experiments. When we vary the number of neurons for a fixed number of clauses, the relative improvement from FPE is largest at small widths, indicating that its benefits are greatest when superposition pressure is high. However, when we instead fix the number of neurons and increase the number of clauses, the FPE gains remain much more consistent than in the fixed-feature case: allowing $\mathbf{E}$ to adapt appears to make the expanded architecture more robust as the number of underlying features grows. A similar pattern emerges on CIFAR-100 with a trainable CNN backbone and MLP classifier. For both projection sizes, FPE reliably improves accuracy across all tested classifier widths, with the largest relative gains at the smallest widths where the bottleneck is most severe. Together, these results show that FPE's interference reducing advantages persist when representations and classifiers are learned jointly.

3.4 Detailed Analysis of Fixed Parameter Expansion on CIFAR-100

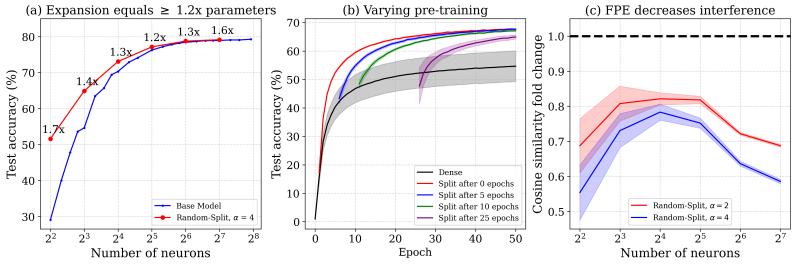

Figure 6: More extensive experiments on CIFAR-100 with random splitting. (a) An FPE model with $\alpha = 4$ achieves equivalent performance to dense networks with $\geq 1.2\times$ the number of parameters, which is indicated above each FPE model size. (b) Training curves of dense and random-split models with varying lengths of pre-training for a network with eight original neurons. (c) Randomly-split models for $\alpha = \{2, 4\}$ achieve lower cosine similarity compared to dense baselines (black dashed line) across neuron count, indicating reduced interference. In all figures, the number of neurons refers to the number of neurons pre-expansion. Relative improvement is calculated as before. Error bars indicate one standard error of the mean. Results collected over ten trials.

We conduct additional experiments using a classifier trained on CLIP-embeddings of CIFAR-100. Random-split FPE with expansion factor $\alpha = 4$ achieves accuracy comparable to dense models with substantially larger parameter budgets—for example, a 32-neuron FPE model matches a dense model with over 1.2× more non-zero weights, and similar trends hold across widths. Varying the timing of the split shows that earlier application of FPE yields the best final accuracy, likely by enabling earlier feature specialization, although even late splits still outperform dense baselines under the same total training budget. Finally, across a wide sweep of model sizes, randomly split FPE models with $\alpha = \{2, 4\}$ consistently exhibit lower mean cosine similarity between neuron weights than dense models, indicating reduced interference that persists as width increases.

Additionally, we validate our framework on deeper architectures and test its compatibility with advanced sparsity techniques such as dynamic mask retraining and structured sparsity, showing that the improvements from FPE apply to such settings as well (Section A.8). We also compare against DropConnect-style (Wan et al., 2013) weight-drop baselines and find that FPE matches or outperforms these methods at equal or larger non-zero parameter budgets (Section A.8.3). These experiments further suggest the potential of improving model performance via interference reduction, which we leave to future work, especially for larger and more complicated models.

4 Related Work

Our work connects to several areas of prior research, including the study of superposition, sparse models, and methods that manipulate representational geometry through weight structure. Below we summarize the connections while highlighting how our goal differs: rather than proposing a new pruning or sparsification technique, we study how interference governs model performance.

Superposition and Interpretability The superposition hypothesis suggests that networks encode more features than available neurons by representing multiple features within the same neuron. While this allows smaller models to have greater effective capacity, it inherently results in interference, diminishing both interpretability and performance (Olah et al., 2020; Elhage et al., 2022; Liu et al., 2025). Recent formal analyses sharpen this view by showing that modeling a given set of features requires a minimum number of parameters (Adler & Shavit, 2024; Hänni et al., 2024; Adler et al., 2025). Sparse autoencoders (SAEs) (Brinkmann et al., 2025; Cunningham et al., 2023) take these insights as a starting point and aim to analyze superposition by learning a separate sparse dictionary over activations, often improving semantic interpretability. FPE is complementary: it uses the same superposition perspective to modify the base network, reallocating a fixed number of weights across more neurons to reduce interference and improve task performance. In principle, SAEs could be applied on top of an FPE network, so we view our method as using motivations from interpretability to guide architecture design rather than competing with SAE-based analysis.

Network Compression and Growth Previous techniques related to model sparsity either train a large model densely and prune after training (Han et al., 2015; Frantar & Alistarh, 2023; Sun et al., 2023; Fang et al., 2024), or dynamically grow or rewire networks over time (Chen et al., 2015; Yuan et al., 2020; Han et al., 2021; Wu et al., 2020; Pham et al., 2024). We propose a third path grounded in insights from superposition: explicitly expanding width at a fixed non-zero parameter budget (Figure A1). Unlike random-masking studies that vary width while training from scratch (Golubeva et al., 2020), FPE is applied after an initial training phase and is designed to reduce interference by assigning features to disjoint sets of weights. DropConnect (Wan et al., 2013) likewise introduces random sparsity, but acts primarily as a regularizer and typically restores dense weights at inference. They do not enforce disjoint sub-neurons or preserve a fixed non-zero budget. By contrast, FPE produces a deterministic sparse architecture that can also be combined with other compression methods such as quantization (Frantar et al., 2022) and distillation (Boix-Adsera, 2024), which change how parameters are represented rather than where they are placed.

5 Conclusion and Discussion

We show that theoretical insights from interpretability can do more than explain trained networks: it can guide in improving architecture design. Across both symbolic and real-world settings, greater width at a constant non-zero parameter count consistently reduces interference and improves accuracy. On Boolean tasks, clause-aligned splits reduce polysemanticity and yield better performance. Strikingly, improvements are also evident for randomly split neurons, consistent with both our theoretical predictions and empirical findings. Our results in real models indicate that performance benefits may arise primarily from alleviating interference pressure, not by precisely encoding specific features. Consistent with the superposition hypothesis, the benefits of width expansion grow with polysemantic load. To our knowledge, this is the first explicit demonstration that reducing superposition is associated with improved performance at a fixed non-zero parameter count. Practically, this direction is attractive on modern accelerators, where memory movement of non-zero parameters, not raw FLOPs, is the primary bottleneck (Rajbhandari et al., 2021).

REFERENCES

Micah Adler and Nir Shavit. On the complexity of neural computation in superposition. *arXiv preprint arXiv:2409.15318*, 2024.

Micah Adler, Dan Alistarh, and Nir Shavit. Towards combinatorial interpretability of neural computation. *arXiv preprint arXiv:2504.08842*, 2025.

Enric Boix-Adsera. Towards a theory of model distillation. *arXiv preprint arXiv:2403.09053*, 2024.

Jannik Brinkmann, Chris Wendler, Christian Bartelt, and Aaron Mueller. Large language models share representations of latent grammatical concepts across typologically diverse languages. *arXiv preprint arXiv:2501.06346*, 2025.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained bert networks. *Advances in neural information processing systems*, 33:15834–15846, 2020.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Michael Carbin, and Zhangyang Wang. The lottery tickets hypothesis for supervised and self-supervised pre-training in computer vision models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 16306–16316, 2021.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*, 2023.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Maximilian Dreyer, Erblina Purelku, Johanna Vielhaben, Wojciech Samek, and Sebastian Lapuschkin. Pure: Turning polysemantic neurons into pure features by identifying relevant circuits. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8212–8217, 2024.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International conference on machine learning*, pp. 2943–2952. PMLR, 2020.

Gongfan Fang, Hongxu Yin, Saurav Muralidharan, Greg Heinrich, Jeff Pool, Jan Kautz, Pavlo Molchanov, and Xinchao Wang. Maskllm: Learnable semi-structured sparsity for large language models. *arXiv preprint arXiv:2409.17481*, 2024.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Gabriel Goh, Nick Cammarata, Chelsea Voss, Shan Carter, Michael Petrov, Ludwig Schubert, Alec Radford, and Chris Olah. Multimodal neurons in artificial neural networks. *Distill*, 6(3):e30, 2021.

Anna Golubeva, Behnam Neyshabur, and Guy Gur-Ari. Are wider nets better given the same number of parameters? *arXiv preprint arXiv:2010.14495*, 2020.

Wes Gurnee, Theo Horsley, Zifan Carl Guo, Tara Rezaei Kheirkhah, Qinyi Sun, Will Hathaway, Neel Nanda, and Dimitris Bertsimas. Universal neurons in gpt2 language models. *arXiv preprint arXiv:2401.12181*, 2024.

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

Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(11): 7436–7456, 2021.

Kaarel Hänni, Jake Mendel, Dmitry Vaintrob, and Lawrence Chan. Mathematical models of computation in superposition. *arXiv preprint arXiv:2408.05451*, 2024.

Adam S Jermyn, Nicholas Schiefer, and Evan Hubinger. Engineering monosemanticity in toy models. *arXiv preprint arXiv:2211.09169*, 2022.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Victor Lecomte, Kushal Thaman, Rylan Schaeffer, Naomi Bashkansky, Trevor Chow, and Sanmi Koyejo. What causes polysemanticity? an alternative origin story of mixed selectivity from incidental causes. *arXiv preprint arXiv:2312.03096*, 2023.

Yizhou Liu, Ziming Liu, and Jeff Gore. Superposition yields robust neural scaling, 2025. URL https://arxiv.org/abs/2505.10465.

Ryan O'Donnell. *Analysis of boolean functions*. Cambridge University Press, 2014.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

Chau Pham, Piotr Teterwak, Soren Nelson, and Bryan A Plummer. Mixturegrowth: Growing neural networks by recombining learned parameters. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2800–2809, 2024.

Jeff Pool, Abhishek Sawarkar, and Jay Rodge. Accelerating inference with sparsity using the nvidia ampere architecture and nvidia tensorrt. *NVIDIA Developer Technical Blog, https://developer. nvidia. com/blog/accelerating-inference-with-sparsityusing-ampere-and-tensorrt*, 2021.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PmLR, 2021.

Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pp. 1–14, 2021.

Adam Scherlis, Kshitij Sachan, Adam S Jermyn, Joe Benton, and Buck Shlegeris. Polysemanticity and capacity in neural networks. *arXiv preprint arXiv:2210.01892*, 2022.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In Sanjoy Dasgupta and David McAllester (eds.), *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pp. 1058–1066, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL https://proceedings.mlr.press/v28/wan13.html.

Lemeng Wu, Bo Liu, Peter Stone, and Qiang Liu. Firefly neural architecture descent: a general approach for growing neural networks. *Advances in neural information processing systems*, 33: 22373–22383, 2020.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Rubing Yang and Pratik Chaudhari. An effective gram matrix characterizes generalization in deep networks. *arXiv preprint arXiv:2504.16450*, 2025.

Xin Yuan, Pedro Savarese, and Michael Maire. Growing efficient deep networks by structured continuous sparsification. *arXiv preprint arXiv:2007.15353*, 2020.

Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. *Advances in neural information processing systems*, 32, 2019.

A Appendix

A.1 Fixed Parameter Expansion framework

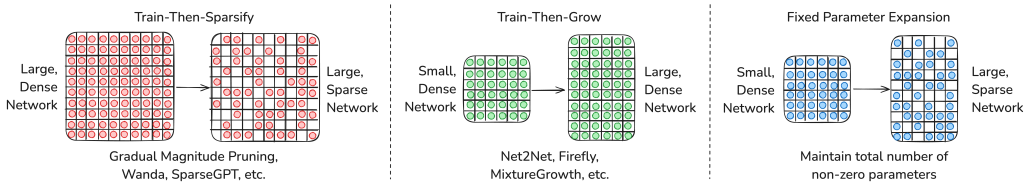

Figure A1: Paradigms of parameter efficiency in training and inference. "Train-then-sparsify" minimizes post-training inference cost, at the expense of training a large, dense model initially and of sparse fine-tuning. "Train-then-grow" amortizes some training cost via bootstrapping from a small, dense network, to yield a large, dense network that is more expensive to inference, though some techniques are constrained by final size. Here, we show theoretical justification and empirical feasibility of directly splitting neurons within a small, dense network into a large, sparse one.

A.2 Fixed Parameter Expansion pseudocode

Algorithm A1 Fixed Parameter Expansion via Neuron Splitting

Require: Weight matrices $\mathbf{W_1} \in \mathbb{R}^{h \times d}$, $\mathbf{W_2} \in \mathbb{R}^{C \times h}$, expansion factor $\alpha \in \mathbb{N}$

Ensure: Expanded, sparse matrices $\tilde{\mathbf{W}}_\mathbf{1}, \tilde{\mathbf{W}}_\mathbf{2}$ with the same total number of non-zero parameters

1: Let $h' \leftarrow \alpha h$; initialize $\tilde{\mathbf{W}}_\mathbf{1} \in \mathbb{R}^{h' \times d}$ and sparsity mask $\mathbf{M_1} \in \{0,1\}^{h' \times d}$
2: **for** $i = 1$ to h **do**
3: Duplicate neuron i into α sub-neurons $i_1, \ldots, i_\alpha$
4: Partition $\mathbf{1}_d$ into disjoint masks $m^{(i_j)} \in \{0,1\}^d$ such that: $\sum_{j=1}^{\alpha} m^{(i_j)} = \mathbf{1}_d$
5: Assign each $m^{(i_j)} \cdot \mathbf{W_1}[i,:]$ to row $i * h + j$ in $\tilde{\mathbf{W}}_\mathbf{1}$ and each $m^{(i_j)}$ to row $i * h + j$ in $\mathbf{M_1}$
6: Initialize $\tilde{\mathbf{W}}_2 \in \mathbb{R}^{C \times h'}$ and sparsity masks $\mathbf{M'_1} \in \{0,1\}^{h' \times d}$ and $\mathbf{M'_2} \in \{0,1\}^{C \times h'}$
7: Estimate $\Delta \leftarrow (\alpha - 1)h \cdot C$
8: Prune smallest-magnitude weights in $\tilde{\mathbf{W}}_\mathbf{1}$ and $\tilde{\mathbf{W}}_\mathbf{2}$ to enforce total parameter count $\leq P$
9: **return** $\tilde{\mathbf{W}}_\mathbf{1}, \tilde{\mathbf{W}}_\mathbf{2}$

A.3 Training and dataset details

A.3.1 4-SAT Boolean dataset generator

For our experiments on Boolean satisfiability, we construct a binary classification dataset over m Boolean variables partitioned into $C = \frac{m}{k}$ disjoint clauses of size k. Positive examples (half of the data) are guaranteed to satisfy at least one clause (i.e. all k literals in that clause are set to 1), by turning on all k literals in a randomly chosen clause, then adding extra "1"s until the total number of active bits lies between $\lfloor \frac{m}{4} \rfloor$ and $\lfloor \frac{m}{4} \rfloor + \lfloor \frac{m}{8} \rfloor$. Negative examples are crafted to violate every clause: we either start from a candidate satisfying assignment and flip one literal per clause, or sample arbitrary assignments and reject any that accidentally satisfy a clause. We retry until we have exactly half positive and half negative samples, with equal proportions of the negative samples generated by flipping the literal and randomly sampling. A fixed random seed ensures full reproducibility. Finally, to improve training stability and encourage generalization, we introduce small continuous variability into the binary input vectors: bits set to `True` (1) are mapped to random values in the range $[3, 3.5]$, while `False` (0) bits are mapped to values in $[0, 0.5]$. This preserves the logical structure of the satisfiability task while ensuring sufficient activation magnitude for effective learning in ReLU networks. Output labels remain binary.

Algorithm A2 Generate Boolean-DNF Classification Data

Require: n total samples, m total literals, k literals per clause, postive data active bits range (minOnes $= \lfloor m/4 \rfloor$, maxOnes $= \lfloor m/4 \rfloor + \lfloor m/8 \rfloor$), positive data proportion $p = 0.5$, seed (optional)

Ensure: $\mathbf{X} \in \{0,1\}^{n \times m}$, $\mathbf{y} \in \{0,1\}^n$

if seed $\neq \varnothing$ **then** set all RNGs to seed
$n_+ \leftarrow \lfloor p\, n \rfloor, \quad n_- \leftarrow n - n_+$
$C \leftarrow m \,/\, k$
Initialize $\mathbf{X}_+ \in \{0\}^{n_+ \times m}$, $\mathbf{X}_- \in \{0\}^{n_- \times m}$
Initialize $\mathbf{y}_+ \leftarrow \mathbf{1}_{n_+}$, $\mathbf{y}_- \leftarrow \mathbf{0}_{n_-}$
for $i = 1$ to n_+ **do**
    Draw clause index $c \sim$ Uniform$\{0, \ldots, C-1\}$
    Draw $s \sim$ UniformInt(minOnes, maxOnes)
    $L \leftarrow \{c \cdot k, \ldots, c \cdot k + k - 1\}$
    Add $(s - k)$ random indices from $\{0, \ldots, m-1\} \setminus L$ into L
    Set $\mathbf{X}_+[i, L] \leftarrow 1$
$i_- \leftarrow 1$, $u \leftarrow -1$
while $i_- \leq n_-$ **do**
    Flip a fair coin:
    if heads **then**
        Pick clause c and provisional sparsity $s \leftarrow (u \geq 0 ? s : u')$, $u' \sim$ UniformInt(minOnes, maxOnes)
        Build L as above, then remove one random index from the clause-block
    else
        $s \leftarrow (u \geq 0 ? s : u')$, sample L uniformly of size s from all m literals
    Set $\mathbf{X}_-[\, i_-, L] \leftarrow 1$
    if EvaluateDNF$(\mathbf{X}_-[i_-],\, m,\, k) = 0$ **then**
        $i_- \leftarrow i_- + 1$, $u \leftarrow -1$
    else
        Reset $\mathbf{X}_-[i_-] \leftarrow \mathbf{0}$, $u \leftarrow s$
return $[\mathbf{X}_+;\, \mathbf{X}_-]$, $[\mathbf{y}_+;\, \mathbf{y}_-]$

A.3.2 VISUAL TASKS

1. **FashionMNIST** consists of 10 clothing classes with balanced class distributions. We use the standard train/test split.
2. **CIFAR-100** includes 100 object categories with 600 images each. We use both the raw as well as the CLIP-extracted representations of the training and test sets for classification.
3. **ImageNet-100** is a 100-class subset of ImageNet-1k, which we selected randomly for computational tractability, as explained in more detail in Section 2.5. We extract CLIP embeddings for both the training and validation splits.
4. **ImageNet-1k** is a widely-used subset of the larger ImageNet dataset, containing approximately 1.2 million training images from 1,000 distinct object classes. We use CLIP-extracted embeddings for both training and validation splits.

A.3.3 TRAINING AND PARAMETERS

In our experiments, we compare three types of models: base neural network, Gram-split model, and a random-split model (Section 3.2). Training was performed on 6 RTX 2080 Ti GPU's, although these tasks are not demanding and could be run on CPU. The number of pre-training epochs was chosen so that at the time of the FPE split, the model had reached performance at least within 95% of that of a fully converged model across all datasets.

All models are trained on the Boolean-DNF data (Alg. A2), with identical hyperparameters:

- Optimizer: Adam with learning rate $\eta = 10^{-3}$
- Batch size B = 64
- Number of pre-training epochs: 25
- Number of fine-tuning epoch: 25
- Regularization: $\lambda_1 = 10^{-7}$ on $\mathbf{W_1}$, $\lambda_2 = 10^{-5}$ on all parameters.
- Trials T = 10

The only exception to this training procedure is in Figure 1, where 1000 epochs of pre-training and fine-tuning were used to fully develop the clauses for the purposes of clearer visualization, and in Figures 2, 3, where five trials were used. All other experiments, both those focused on interpretability and on performance, were conducted with the hyperparameters above.

Algorithm A3 Model Training and Selection

Require: Training data $(X_{\text{tr}}, y_{\text{tr}})$, Test data $(X_{\text{te}}, y_{\text{te}})$
Model class $\mathcal{M} \in \{\text{BaseModel}, \text{GramSplitModel}, \text{RandomSplitModel}\}$ pre-training epochs P, fine-tuning epochs F, batch size B, learning rate η, regularization penalties $\lambda_1, \lambda_2, \lambda_{\text{orth}}$, trials T
Ensure: Best model $\hat{M}$ and its test accuracy

$\text{bestAcc} \leftarrow 0$
for $t = 1, \ldots, T$ **do**
  Instantiate $M \leftarrow$ BaseModel and move to device
  $\text{opt} \leftarrow \text{Adam}(M.\text{parameters}(), \text{lr} = \eta)$
  Define binary cross-entropy loss BCE
  Build a DataLoader over $(X_{\text{tr}}, y_{\text{tr}})$ with batch size B
  for epoch $e = 1$ to P **do**
    for each batch (x_b, y_b) **do**
      $\text{opt.zero\_grad}()$
      $\hat{y} \leftarrow M(x_b)$
      $\ell \leftarrow \text{BCE}(\hat{y},\ y_b)$
      if $\lambda_1 > 0$ **then**
        $\ell \mathrel{+}= \lambda_1 \|W_1\|_1$
      if $\lambda_2 > 0$ **then**
        $\ell \mathrel{+}= \lambda_2 \sum_{p \in M.\text{parameters}} \|p\|_2^2$
      if $\lambda_{\text{orth}} > 0$ **then**
        $\ell \mathrel{+}= \lambda_{\text{orth}} \|W_1 W_1^T - I\|_F^2$
      $\ell.\text{backward}()\ \text{opt.step}()$
  Either keep fine-tuning the BaseModel or split the model via FPE
  for epoch $e = P + 1$ to $P + F$ **do**
    for each batch (x_b, y_b) **do**
      $\text{opt.zero\_grad}()$
      $\hat{y} \leftarrow M(x_b)$
      $\ell \leftarrow \text{BCE}(\hat{y},\ y_b)$
      if $\lambda_1 > 0$ **then**
        $\ell \mathrel{+}= \lambda_1 \|W_1\|_1$
      if $\lambda_2 > 0$ **then**
        $\ell \mathrel{+}= \lambda_2 \sum_{p \in M.\text{parameters}} \|p\|_2^2$
      if $\lambda_{\text{orth}} > 0$ **then**
        $\ell \mathrel{+}= \lambda_{\text{orth}} \|W_1 W_1^T - I\|_F^2$
      $\ell.\text{backward}()\ \text{opt.step}()$
  ▷ At each epoch, evaluate on test set
  $\hat{y}_{\text{te}} \leftarrow M(X_{\text{te}})$
  $\text{acc} \leftarrow \text{mean}\big(\arg\max_k \hat{y}_{\text{te},k} \ =\ y_{\text{te}}\big)$
  if $\text{acc} > \text{bestAcc}$ **then**
    $\text{bestAcc} \leftarrow \text{acc}, \quad \hat{M} \leftarrow M$
return $\hat{M},\ \text{bestAcc}$

A.4 EVEN RANDOM-SPLIT FPE REDUCES INTERFERENCE AND IMPROVES PERFORMANCE

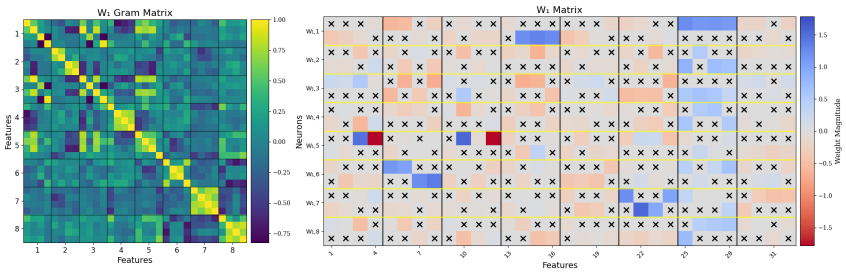

Figure A2: Random-split model gram and $\mathbf{W_1}$ matrices. Codes are more broken between neurons. Black x's represent masked parameters of value 0.

A.5 Theoretical justification for Fixed Parameter Expansion

We demonstrate theoretical properties of a randomly constructed Fixed Parameter Expansion (FPE) network. We find that with high probability, an FPE network maintains "coverage" of all clauses by at least one neuron, while significantly reducing the interference caused by a single neuron covering multiple clauses, when compared to a dense network with the same number of non-zero parameters. This holds true even when the FPE network is constructed by randomly choosing a subset of active weights for each neuron without any knowledge of the task structure.

Model Setup We analyze a DNF formula with C clauses over m literals, with each clause having k literals. We compare two architectures with equal parameter counts. One is a dense network with r neurons, each connected to all m literals. The other is a Fixed Parameter Expansion (FPE) network with expansion factor α, with αr neurons, each of which is connected to $d = \frac{m}{\alpha}$ literals by non-zero parameters, chosen at random for each neuron. Crucially, this selection is performed without any knowledge of the underlying clause structure. For simplicity, let us allow for replacement when choosing non-zero weights between all neurons. We say a neuron covers a clause if the neuron's weights for each literal in the clause has the potential to be non-zero.

Clause Coverage Probability First, for the network to learn the DNF formula, there must be at least one neuron that can compute each clause. A neuron can only compute a clause if it receives input from all the literals that comprise it; otherwise, the clause is fundamentally incomputable for that neuron. Therefore, we first show that the FPE network is highly likely to cover each clause, even though the network's sparse connections were chosen randomly.

The probability p that a single sparse neuron covers a specific k-literal clause is $p = \frac{\binom{m-k}{d-k}}{\binom{m}{d}}$. For a single clause S_i, the probability that it is not covered by one neuron is $(1-p)$. Because of our independence assumption, the probability that S_i is missed by every neuron is Pr[clause S_i missed] $M_i = (1-p)^{\alpha r}$. To find the probability that at least one of the C clauses is missed, we can use a union bound to provide an upper bound on the probability that at least one of the C clauses is missed. Pr[at least one clause missed] $= \Pr(\bigcup_{i=1}^{C} M_i) \leq \sum_{i=1}^{C} \Pr[M_i] = C \cdot (1-p)^{\alpha r}$. The probability that all clauses are covered Pr[all clauses covered] is therefore $\geq 1 - C \cdot (1-p)^{\alpha r}$. For large m and d, p is well approximated by $p = \frac{\binom{m-k}{d-k}}{\binom{m}{d}} = \frac{d!(m-k)!}{m!(d-k)!} \approx (\frac{d}{m})^k = (\frac{1}{\alpha})^k$, and for small x, $(1-x) \approx e^{-x}$. Together, Pr[all clauses covered] $\geq 1 - C \cdot (1-p)^{\alpha r} \approx 1 - C \cdot e^{-r/\alpha^{k-1}}$. To ensure Pr[at least one clause missed] $\leq C \cdot e^{-r/\alpha^{k-1}}$ is within some tolerance ϵ, $C \cdot e^{-r/\alpha^{k-1}} < \epsilon$. Solving for r yields $r > \alpha^{k-1} \ln(\frac{C}{\epsilon})$. This provides a concrete requirement on the network's dimensions to guarantee clause coverage with high probability.

Interference Analysis Now, let us consider the decrease in interference from this procedure. We analyze interference because it is a proxy for the difficulty of the optimization problem. When a single neuron covers multiple clauses, its weights receive conflicting gradient updater during training, as it tries to simultaneously represent multiple distinct concepts. This entanglement can make it difficult for optimizers to learn a clean representation for any single clause. By showing that FPE dramatically reduces the expected number of these interference events, we argue that it creates a more factored, disentangled learning problem that is fundamentally easier to solve.

For two clauses S and T, in the dense network, the probability that neuron r covers both is 1. Thus, the total number of interference instances (a neuron covering a pair of clauses) is $E_{dense} = r \cdot \binom{C}{2}$. On the other hand, in the FPE network, the probability p' that a neuron r covers both S and T is $p' = \frac{\binom{m-2k}{d-2k}}{\binom{m}{d}} \approx (\frac{1}{\alpha})^{2k}$. The expected number of clause collisions in an FPE network is therefore $E_{FPE} = \alpha r \cdot \binom{C}{2} \cdot p'$. The ratio of the expected number of collisions in the FPE network to that of dense network is therefore $\frac{E_{FPE}}{E_{dense}} = \frac{\alpha r \cdot \binom{C}{2} \cdot p'}{r \cdot \binom{C}{2}} = \alpha p' \approx \frac{1}{\alpha^{2k-1}}$. This ratio of interference shows a significant reduction for the FPE network. Intuitively, for a total of $2k$ literals to be covered by the same neuron, the first literal is free to be placed onto any neuron, while the subsequent $2k-1$ literals must be in the $\frac{1}{\alpha}$ non-zero weights for the neuron. The approximate probability of this happening

is $\frac{1}{\alpha^{2k-1}}$. Thus, our analysis shows that a sparse, wide FPE network is theoretically advantageous, even in the case where the clauses are not explicitly known. By simply applying random sparsity, it maintains coverage of all logical clauses while drastically reducing interference, all without needing to know the specific clauses in advance.

A.6 Additional feature interference reduction measurements

Table A1: Average total feature capacity. Higher values represent decreased superposition. Each network originally had eight dense neurons. Results averaged across five trials.

# Literals	Dense	Clause-split	Random-split
12	1.896 ± 0.377	2.740 ± 0.114	2.780 ± 0.133
24	2.907 ± 0.370	5.479 ± 0.400	5.268 ± 0.317
32	3.944 ± 0.409	6.977 ± 0.426	6.848 ± 0.568
40	4.819 ± 0.491	7.853 ± 0.469	7.825 ± 0.365
60	5.584 ± 0.670	10.79 ± 0.868	10.90 ± 0.747
80	5.812 ± 1.044	13.07 ± 1.093	12.00 ± 1.726
100	6.219 ± 0.535	14.21 ± 1.866	15.00 ± 0.727
128	7.699 ± 1.029	16.87 ± 0.926	13.80 ± 1.037

Table A2: Average neuron cosine similarity. Lower values represent decreased superposition. Each network originally had eight dense neurons. Results averaged across five trials.

# Literals	Dense	Clause-split	Random-split
12	0.332 ± 0.123	0.230 ± 0.021	0.243 ± 0.083
24	0.382 ± 0.127	0.173 ± 0.041	0.219 ± 0.067
32	0.291 ± 0.230	0.150 ± 0.072	0.171 ± 0.092
40	0.257 ± 0.096	0.137 ± 0.019	0.146 ± 0.049
60	0.275 ± 0.095	0.145 ± 0.030	0.142 ± 0.045
80	0.303 ± 0.101	0.108 ± 0.035	0.113 ± 0.025
100	0.312 ± 0.081	0.134 ± 0.028	0.144 ± 0.041
128	0.262 ± 0.081	0.105 ± 0.031	0.108 ± 0.027

A.7 RAW TEST ACCURACIES FOR REAL-WORLD VISION TASKS

Here, we show with the accuracy per parameter for models presented in Figures 2, 4, 5. Additionally, we provide the raw test accuracies associated with Figures 4, 5.

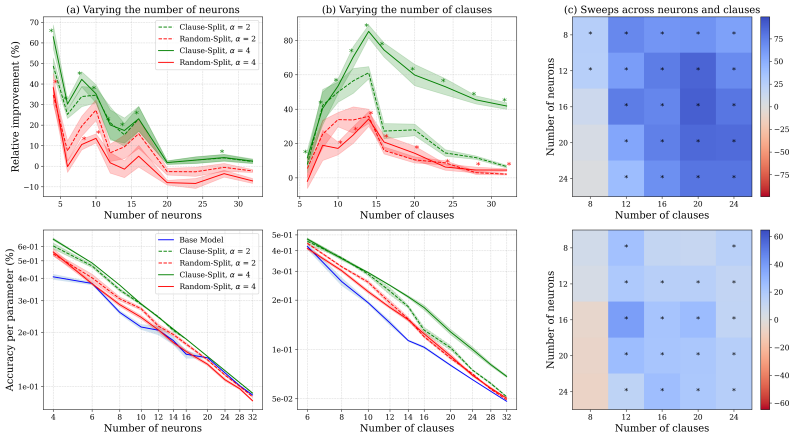

Figure A3: Trends of performance under superposition in neurons and clauses. (a) Relative improvement in test accuracy and the accuracy per parameter for models of varying hidden dimension on 8 clauses. (b) Relative improvement in test accuracy and the accuracy per parameter for models with 8 neurons. y-axis labels are shared for (a) and (b). (c) Heatmap of relative improvement percentage for clause-split split FPE models (top) and random-split FPE models (bottom). Relative improvement is calculated as $\frac{\text{FPE test accuracy} - \text{dense test accuracy}}{\text{dense test accuracy}}$. Error bars indicate one standard error of the mean. * indicates $p < 0.05$ and is shown only for $\alpha = 4$ for clarity. Results collected over five trials.

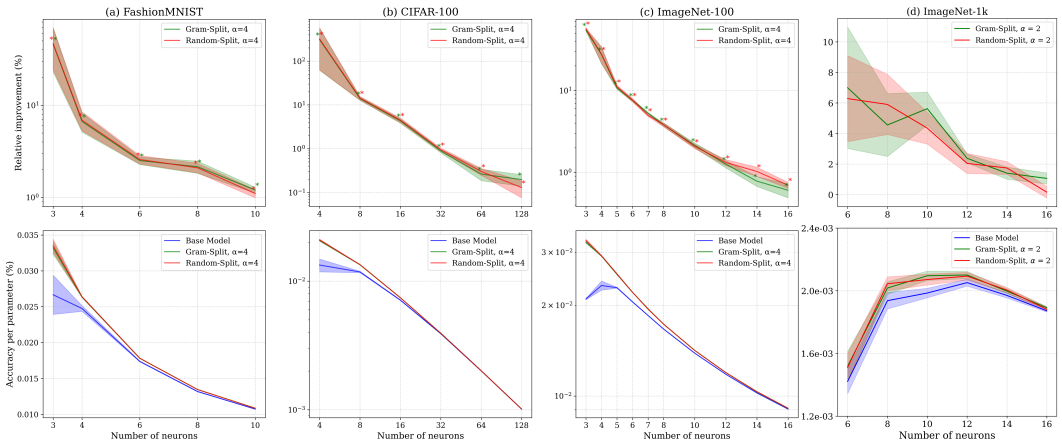

Figure A4: Fixed Parameter Expansion helps on real datasets like (a) FashionMNIST, (b) CLIP-embeddings of CIFAR-100, (c) CLIP-embeddings of ImageNet-100, and (d) CLIP-embeddings of ImageNet-1k. All baseline models were trained for 25 epochs before FPE. After the split, both the FPE model and dense model were further fine-tuned for an additional 25 epochs. The first row indicates the improvement in test accuracy of the FPE model relative to the dense baseline, for varying hidden dimensions, and is calculated as before. The second row shows the test accuracy per parameter for each configuration. In all figures, the number of neurons refers to the number of neurons pre-expansion. Relative improvement is calculated as before. Error bars indicate one standard error of the mean. * indicates $p < 0.05$. Results collected over ten trials.

Table A3: FashionMNIST. Results averaged across 10 trials.

Hidden size	Dense	Gram-Split FPE	Random-Split FPE
3	63.5 ± 6.5	79.1 ± 2.1	$\mathbf{79.8 \pm 2.2}$
4	78.6 ± 1.2	83.7 ± 0.2	$\mathbf{83.8 \pm 0.2}$
6	82.8 ± 0.3	$\mathbf{84.9 \pm 0.1}$	$\mathbf{84.9 \pm 0.1}$
8	83.8 ± 0.3	$\mathbf{85.5 \pm 0.1}$	$\mathbf{85.5 \pm 0.1}$
10	85.2 ± 0.1	$\mathbf{86.2 \pm 0.1}$	86.1 ± 0.1

Table A4: CLIP-embeddings of CIFAR-100. Results averaged across 10 trials.

Hidden size	Dense	Gram-Split FPE	Random-Split FPE
4	32.7 ± 3.8	50.8 ± 0.9	$\mathbf{51.3 \pm 0.7}$
8	57.8 ± 0.7	65.7 ± 0.2	$\mathbf{65.9 \pm 0.1}$
16	69.7 ± 0.5	72.7 ± 0.2	$\mathbf{72.8 \pm 0.1}$
32	76.4 ± 0.1	77.0 ± 0.1	$\mathbf{77.1 \pm 0.1}$
64	78.6 ± 0.1	$\mathbf{78.8 \pm 0.0}$	$\mathbf{78.8 \pm 0.1}$
128	79.2 ± 0.0	$\mathbf{79.4 \pm 0.0}$	79.3 ± 0.0

Table A5: CLIP-embeddings of ImageNet-100. Results averaged across 10 trials.

Hidden size	Dense	Gram-Split FPE	Random-Split FPE
3	38.5 ± 0.2	59.5 ± 0.5	$\mathbf{60.3 \pm 0.6}$
4	57.0 ± 2.0	71.4 ± 0.3	$\mathbf{71.5 \pm 0.3}$
5	70.0 ± 0.2	77.5 ± 0.2	$\mathbf{77.8 \pm 0.2}$
6	75.4 ± 0.2	$\mathbf{81.1 \pm 0.1}$	$\mathbf{81.1 \pm 0.2}$
7	79.2 ± 0.1	$\mathbf{83.4 \pm 0.1}$	83.2 ± 0.1
8	81.6 ± 0.2	$\mathbf{84.7 \pm 0.1}$	$\mathbf{84.7 \pm 0.1}$
10	84.9 ± 0.1	$\mathbf{86.7 \pm 0.1}$	86.6 ± 0.1
12	86.7 ± 0.1	$\mathbf{87.8 \pm 0.1}$	$\mathbf{87.8 \pm 0.1}$
14	87.7 ± 0.1	88.4 ± 0.1	$\mathbf{88.6 \pm 0.1}$
16	88.6 ± 0.1	89.1 ± 0.1	$\mathbf{89.2 \pm 0.0}$

Table A6: CLIP-embeddings of ImageNet-1k using a 5-layer MLP. Results averaged across 10 trials.

Hidden size	Dense	Random-Split FPE
6	13.0%	**13.9%**
8	23.8%	**25.1%**
10	30.6%	**31.9%**
12	38.1%	**38.8%**
14	42.9%	**43.6%**
16	46.9%	**47.0%**

A.7.1 LEARNING FEATURES JOINTLY WITH CLASSIFICATION

Recall that for our simple classifier, we use $\mathbf{z} = \mathbf{W_2} \cdot \text{ReLU}(\mathbf{W_1 x} + \mathbf{b}_1) + \mathbf{b}_2$, where $\mathbf{x} \in \mathbb{R}^d$ is the input and h denotes the number of neurons. To investigate how FPE performs when jointly learning features and classification, we use experiment with two avenues. In both cases, the entire model is pre-trained densely for 25 epochs, split via FPE, and fine-tuned for another 25 epochs, and compared to a model that was trained densely for 50 epochs. The feature embeddings are not frozen at any phase.

Boolean Experiments with Embedding Layer First, for our Boolean experiments, we can prepend a linear embedding layer $\mathbf{E}$ before the classifier to act as a feature learner. Now, $\mathbf{z} = \mathbf{W_2} \cdot \text{ReLU}(\mathbf{W_1 E x} + \mathbf{b}_1) + \mathbf{b}_2$. We can expand $\mathbf{W_1}$ and $\mathbf{W_2}$ as before. When either varying the number of neurons for eight clauses (Figure 5a), or varying the number of clauses for eight neurons (Figure 5b), FPE delivers consistent improvement in performance to the dense model.

CIFAR-100 Images with CNN Backbone Second, we use CNN backbone with an MLP block. The network operates on 3-channel images of spatial resolution 32×32, and we use it to classify CIFAR-100. The CNN backbone consists of three VGG-style (Simonyan & Zisserman, 2014) blocks with increasing channel width, followed by a 1 by 1 projection and global average pooling. This is prepended to an MLP block, that can be split via FPE as before. In more detail, the three blocks are as follows:

1. **Convolutional blocks.** Each block maps an input with C_{in} channels to an output with C_{out} channels using a stack of three 3×3 convolutions with same padding, each followed by batch normalization and a ReLU nonlinearity:

$$\begin{aligned}&\text{Conv2d}(C_{\text{in}}, C_{\text{out}}, \text{kernel} = 3, \text{stride} = 1, \text{padding} = 1, \text{bias} = \text{False})\\&\quad \to \text{BatchNorm2d}(C_{\text{out}}) \to \text{ReLU},\\&\text{Conv2d}(C_{\text{out}}, C_{\text{out}}, \text{kernel} = 3, \text{stride} = 1, \text{padding} = 1, \text{bias} = \text{False})\\&\quad \to \text{BatchNorm2d}(C_{\text{out}}) \to \text{ReLU},\\&\text{Conv2d}(C_{\text{out}}, C_{\text{out}}, \text{kernel} = 3, \text{stride} = 1, \text{padding} = 1, \text{bias} = \text{False})\\&\quad \to \text{BatchNorm2d}(C_{\text{out}}) \to \text{ReLU}.\end{aligned}$$

   This construction replaces a single wide 7×7 convolution with a sequence of three 3×3 convolutions, which reduces parameters while preserving the effective receptive field and adds extra non-linearities.

2. **Spatial downsampling.** After each block we apply 2×2 max-pooling with stride 2 to halve the spatial resolution:
   - Block 1: $3 \to 64$ channels, then MaxPool2d(2). For 32×32 inputs this gives $16 \times 16 \times 64$.
   - Block 2: $64 \to 128$ channels, then MaxPool2d(2). Output: $8 \times 8 \times 128$.
   - Block 3: $128 \to 256$ channels, then MaxPool2d(2). Output: $4 \times 4 \times 256$.

3. **Channel projection and global pooling.** The final $4 \times 4 \times 256$ feature map is projected to a `flatten_size`-dimensional representation using a 1×1 convolution:

$$\begin{aligned}&\text{Conv2d}(256, \texttt{flatten\_size}, \text{kernel} = 1, \text{bias} = \text{False})\\&\quad \to \text{BatchNorm2d}(\texttt{flatten\_size}) \to \text{ReLU}.\end{aligned}$$

   We then apply global average pooling over the spatial dimensions via AdaptiveAvgPool2d(1), producing a tensor of shape $(B, \texttt{flatten\_size}, 1, 1)$, which is flattened to a vector of length `flatten_size` per example.

In our experiments we test `flatten_size` $= 256$ and $= 512$, the latter of which is identical to the CLIP-embedding dimension. This representation $\mathbf{x}$ is then fed into the MLP classifier as before, $\mathbf{z} = \mathbf{W_2} \cdot \text{ReLU}(\mathbf{W_1 x} + \mathbf{b}_1) + \mathbf{b}_2$. In this setting, when testing both a `flatten_size` of 256 and of 512, FPE delivers improvements, especially in the interference-constrained settings (Figure 5c, d).

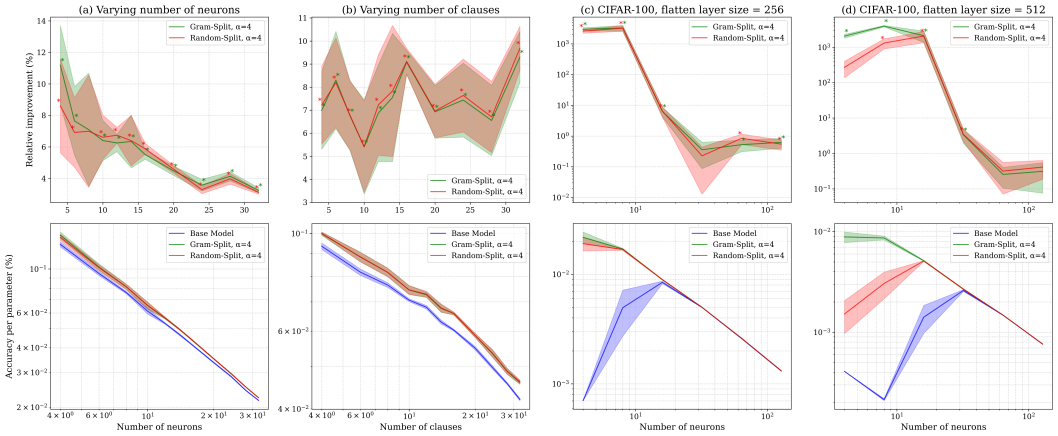

Figure A5: Jointly learning the features and classifying them is improved via increasing width. (a) Training a classifier on a 4DNF Boolean task with eight clauses while varying the number of neurons and with an embedding layer. (b) Training a classifier on a 4DNF Boolean task with eight neurons while varying the number of clauses and with an embedding layer. (c) Training a CNN on CIFAR-100 with an embedding dimension of 256. (d) Training a CNN on CIFAR-100 with an embedding dimension of 512. The first row indicates the improvement in test accuracy of the FPE model relative to the dense baseline, for varying hidden dimensions. The second row shows the test accuracy per parameter for each configuration. In all figures, the number of neurons refers to the number of neurons pre-expansion. Relative improvement is calculated as before. Error bars indicate one standard error of the mean. * indicates $p < 0.05$. Results collected over ten trials.

The raw values that created Figure 5 is provided below.

Table A7: Training a classifier on a 4DNF Boolean task with eight clauses while varying the number of neurons and with an embedding layer. Results averaged across 10 trials.

Hidden size	Dense	Gram-Split	Random-Split
4	70.3 ± 2.1	$\mathbf{78.2 \pm 2.9}$	76.3 ± 2.9
6	75.2 ± 1.8	$\mathbf{81.1 \pm 2.9}$	80.5 ± 2.8
8	80.2 ± 1.1	$\mathbf{85.7 \pm 2.8}$	$\mathbf{85.7 \pm 2.8}$
10	80.8 ± 2.7	86.2 ± 3.4	$\mathbf{86.4 \pm 3.5}$
12	84.1 ± 0.7	89.3 ± 0.9	$\mathbf{89.8 \pm 0.9}$
14	86.0 ± 0.9	91.3 ± 0.7	$\mathbf{91.4 \pm 0.6}$
16	87.1 ± 0.6	91.9 ± 0.6	$\mathbf{92.2 \pm 0.5}$
20	88.9 ± 0.6	92.8 ± 0.5	$\mathbf{92.9 \pm 0.6}$
24	90.7 ± 0.6	$\mathbf{93.9 \pm 0.4}$	93.6 ± 0.5
28	90.0 ± 0.4	$\mathbf{93.7 \pm 0.3}$	93.6 ± 0.3
32	91.6 ± 0.3	$\mathbf{94.6 \pm 0.3}$	94.4 ± 0.3

Table A8: Training a classifier on a 4DNF Boolean task with eight neurons while varying the number of clauses and with an embedding layer. Results averaged across 10 trials.

Number of clauses	Dense	Gram-Split	Random-Split
4	86.8 ± 1.5	92.7 ± 0.8	$\mathbf{92.9 \pm 0.6}$
6	81.1 ± 1.4	$\mathbf{87.8 \pm 2.2}$	87.7 ± 2.2
8	80.6 ± 1.0	$\mathbf{86.2 \pm 2.0}$	$\mathbf{86.2 \pm 1.9}$
10	79.1 ± 0.6	$\mathbf{83.4 \pm 2.0}$	$\mathbf{83.4 \pm 1.9}$
12	80.6 ± 0.8	86.0 ± 1.2	$\mathbf{86.2 \pm 1.3}$
14	78.6 ± 0.8	84.4 ± 1.7	$\mathbf{84.6 \pm 1.8}$
16	79.2 ± 0.4	$\mathbf{86.4 \pm 0.6}$	$\mathbf{86.4 \pm 0.7}$
20	79.1 ± 0.7	84.5 ± 0.9	$\mathbf{84.6 \pm 0.9}$
24	77.9 ± 0.7	83.7 ± 1.6	$\mathbf{83.9 \pm 1.6}$
28	77.4 ± 0.4	82.5 ± 1.3	$\mathbf{82.6 \pm 1.3}$
32	76.6 ± 0.5	83.7 ± 0.8	$\mathbf{84.0 \pm 0.8}$

Table A9: Training a CNN on CIFAR-100 with an embedding dimension of 256. Results averaged across 10 trials.

Number of neurons	Dense	Gram-Split	Random-Split
4	1.0 ± 0.0	$\mathbf{30.9 \pm 3.6}$	27.0 ± 3.8
8	14.1 ± 6.3	$\mathbf{48.4 \pm 0.7}$	47.9 ± 0.9
16	48.0 ± 1.2	50.8 ± 0.4	$\mathbf{50.9 \pm 0.4}$
32	56.7 ± 0.2	$\mathbf{56.9 \pm 0.2}$	$\mathbf{56.9 \pm 0.2}$
64	58.9 ± 0.1	59.2 ± 0.1	$\mathbf{59.4 \pm 0.2}$
128	59.7 ± 0.1	$\mathbf{60.0 \pm 0.1}$	$\mathbf{60.0 \pm 0.1}$

Table A10: Training a CNN on CIFAR-100 with an embedding dimension of 512. Results averaged across 10 trials.

Number of neurons	Dense	Gram-Split	Random-Split
4	1.0 ± 0.0	$\mathbf{21.7 \pm 2.6}$	3.7 ± 1.3
8	1.0 ± 0.0	$\mathbf{42.3 \pm 1.9}$	14.9 ± 4.4
16	13.8 ± 4.3	$\mathbf{50.4 \pm 0.6}$	50.0 ± 0.5
32	50.9 ± 1.0	52.5 ± 0.5	$\mathbf{52.6 \pm 0.5}$
64	57.9 ± 0.2	58.0 ± 0.2	$\mathbf{58.1 \pm 0.2}$
128	59.7 ± 0.1	59.8 ± 0.1	$\mathbf{59.9 \pm 0.1}$

A.8 Validating Fixed Parameter Expansion at scale and in practical scenarios

A.8.1 Scaling to Deeper Architectures and High-Dimensional Outputs

We first apply FPE to deeper (5, 7, and 9-layer) MLPs on CLIP embeddings of CIFAR-100, incorporating LayerNorm before each activation function for training stability. To scale FPE while maintaining a fixed parameter count, we use an alternating expansion strategy.

Concretely, for hidden layers i, $i+1$, and $i+2$, when we expand i from $h \to \alpha h$ neurons, the subsequent layer $i+1$ must also be adjusted to accept a wider input, from $h \to \alpha h$ as well. We then apply the `re-sparsify` method as described before (Section 2.5), and do not expand then neurons in layer $i+1$. Thus, for hidden layer $i+2$, we do not need to worry about expanding the input, and can simply expand the number of neurons again. We repeat this process with pairs of hidden layers, stopping before and not expanding the final classification head.

This alternating approach allows the network to benefit from the increased neuron capacity of FPE while maintaining stability and avoiding the excessive sparsity that could arise from expanding consecutive layers. As the results below show, FPE provides significant accuracy improvements, especially in narrower, more constrained models, confirming its compatibility with deeper, standard architectures (Table A11).

Table A11: Relative improvement in test accuracy from applying FPE to deeper MLPs with varying baseline hidden dimensions. Networks trained on CLIP embeddings of CIFAR-100. $\alpha = 2$. Results averaged across 30 trials.

Depth	Hidden 4	Hidden 8	Hidden 12	Hidden 16
5	28%	2.6%	0.4%	0.2%
7	34%	11%	1.1%	0.8%
9	104%	8.7%	2.0%	1.4%

By not expanding the final classification layer, FPE can also be applied to tasks with a large number of classes. We validate this by applying FPE to a 5-layer MLP trained on the full ImageNet-1k dataset (Table A6), where it consistently improves performance over the dense baseline.

A.8.2 Compatibility with Advanced and Structured Sparsity

To further assess FPE's practical utility, we test its compatibility with two key techniques: dynamic sparse training and hardware-friendly structured sparsity.

Inspired by RigL (Evci et al., 2020), we investigate if FPE can be enhanced with a dynamic mask. After the initial FPE split, we periodically update the sparsity mask by randomly unmasking a fraction of weights and, to maintain a fixed parameter count, re-pruning an equal number of the lowest-magnitude weights. This learnable approach yields consistent gains over our one-shot FPE. This approach improves upon random one-shot FPE by 4.5% for 4 originally dense neurons, 1.9% for 8 originally dense neurons, and 0.5% for 16 originally dense neurons, confirming this is a promising direction for future work. This corresponds to relative improvements of 53%, 14%, and 3.0% over the respective dense models. These were collected over ten trials for the optimal rewiring hyperparameters on a single layer MLP trained on CIFAR-100 embeddings. This demonstrates that FPE provides a strong foundation that can be enhanced with more sophisticated, learnable sparse training techniques.

To explore FPE's potential for efficient inference, we constrain its masks to a 2:4 structured sparsity pattern, which is supported by modern hardware accelerators (Pool et al., 2021). For an expansion factor of $\alpha = 2$, each group of four consecutive weights in the original dense neuron is randomly assigned such that two weights connect to one sub-neuron and the other two connect to the second. As shown in Table A12, this structured approach achieves performance that is highly competitive with unstructured random sparsity. This is a promising result, as it suggests that FPE can be adapted to leverage significant hardware speedups without a meaningful performance trade-off.

Table A12: 2:4 structured sparsity FPE performs comparably to random sparsity FPE on CIFAR-100. RI denotes relative improvement. $h = 8, \alpha = 2$. Results averaged across 10 trials.

Dense Accuracy	2:4 Sparsity FPE Accuracy	Random Sparsity FPE Accuracy	2:4 Sparsity FPE RI	Random Sparsity FPE RI
53.89%	61.67%	61.92%	14.42%	14.89%

A.8.3 Comparison of Fixed Parameter Expansion to DropConnect

To compare our technique to DropConnect (Wan et al., 2013), we compare FPE to two DropConnect style baselines that match the FPE setting as closely as possible while using standard DropConnect training and inference rules.

- a **non-disjoint FPE** baseline, which keeps the "one-shot" sparse architecture but removes the disjointness constraint on incoming weights to simulate a single DropConnect instantiation
- a **DropConnect** baseline in the standard sense, where a new weight mask is sampled at each forward pass during training and a dense, scaled weight matrix is used at test time.

Both baselines use the same warmup schedule and expansion factor as FPE and are designed so that the *expected* number of active parameters per forward pass during training matches the dense and FPE models. The key differences are (i) whether the mask is sampled once or resampled every step, and (ii) whether supports across duplicated neurons are disjoint.

Non-disjoint FPE (one-shot random mask). This baseline is architecturally closest to FPE and serves to isolate the effect of disjointness.

1. **Warmup (dense).** We first train the original dense classifier (with hidden width r) for the same 25 pre-training epochs as in our FPE experiments (Section A.3).
2. **Neuron duplication.** After warmup, we duplicate each hidden neuron α times, as in FPE. Let $\mathbf{W_1} \in \mathbb{R}^{h\times d}$ and $\mathbf{W_2} \in \mathbb{R}^{C\times h}$ denote the input and output weight matrices of the dense model. We construct duplicated matrices
$$\mathbf{W_1}' \in \mathbb{R}^{\alpha h\times d}, \quad \mathbf{W_2}' \in \mathbb{R}^{C\times \alpha h},$$
by copying each row of W_1 and each column of W_2 into α identical copies (i.e., we increase the number of neurons but do not yet introduce sparsity).
3. **Mask initialization** We then simulate how FPE would create a mask for each neuron in $\mathbf{W_1}$ and $\mathbf{W_2}$ via re-sparsification (Section 2.5), yielding $\mathbf{M_{1,FPE}}$ and $\mathbf{M_{2,FPE}}$. We keep $\mathbf{M_{2,FPE}}$ as $\mathbf{M_2}$ for DropConnect's $\mathbf{W_2}$, but $\mathbf{M_{1,FPE}}$ is not used: it is purely to set how many weights per neuron are non-zero in both DropConnect setups.
4. **One-shot random masking.** For each neuron n_i in $\mathbf{W_1}$, find how many non-zero weights there are in $\mathbf{M_{1,FPE}}$ for n_i. To construct $\mathbf{M_1}$, simply sample that many weights randomly once. This yields a sparse, widened architecture in which each duplicated neuron has the same degree as under FPE, but the supports of different neurons can overlap and some input dimensions may be dropped entirely. The mask $\mathbf{M_1}$ is sampled once after warmup and then kept fixed for the remainder of training and at test time.
5. **Training and inference.** We continue training the network with $\mathbf{W_1}' \odot \mathbf{M_1}$ as a fixed sparse weight matrix (only surviving weights are updated), and use the same fixed mask at inference. Thus this baseline is a one-shot, non-disjoint alternative to FPE that maintains a fixed number of non-zero parameters at test time but does not minimize feature collisions across neurons.

Empirically, this non-disjoint FPE baseline underperforms both true FPE and, at large expansion factors, even the dense model (Figure A6, Table A13). This indicates that it is not widening alone that matters, but specifically that FPE's disjoint allocation of incoming weights reduces collisions and improves performance.

DropConnect (weight dropout with dense test-time weights). To more closely follow the standard DropConnect setting, we also implement a baseline with *resampled* masks during training and dense weights at test time.

1. **Warmup, duplication, and mask initialization.** As above, we first warm up a dense classifier, duplicate each hidden neuron α times, and initialize a sparsity mask to obtain $\mathbf{W_1}'$, $\mathbf{W_2}'$, $\mathbf{M_1}$, $\mathbf{M_2}$.

2. **DropConnect training.** During the continued training phase, we apply DropConnect to the incoming weights $\tilde{W}_1$ by sampling a new binary mask at each forward pass in the same way as before. For each neuron n_i in $\mathbf{W_1}$, find how many non-zero weights there are in $\mathbf{M_{1,FPE}}$ for n_i. To construct $\mathbf{M_1}$, simply sample that many weights randomly once. This yields a sparse, widened architecture in which each duplicated neuron has the same degree as under FPE. After $\mathbf{M_1}$ is resampled for each forward pass during training, $\mathbf{W_1}' \odot \mathbf{M_1}$ is used as the sparse weights. For simplicity, we apply DropConnect only to $\mathbf{W_1}'$; applying it to $\mathbf{W_2}'$ does not change the qualitative conclusions.
3. **Inference with weight averaging.** At test time we follow the standard DropConnect inference rule (Wan et al., 2013): we do not sample a mask, but instead evaluate the network with dense, deterministically scaled weights

$$\mathbf{W_{1,TEST}} = p\mathbf{W_1}'$$

where p is the number of non-zero weights in $\mathbf{M_1}$ divided by the total number of weights in $\mathbf{M_1}$, as per Wan et al. (2013). In practice, this is slightly less than $\frac{1}{\alpha}$ due to the resparsification procedure. Thus, unlike FPE and the non-disjoint FPE baseline, the DropConnect model is dense at inference and does not obey a fixed non-zero parameter budget; it serves as a regularization baseline rather than a fixed-parameter architectural comparator.

Intuitively, the non-disjoint FPE baseline tests whether widening with a single random sparse mask (but without disjointness) is sufficient to obtain FPE's benefits, while the DropConnect baseline tests whether repeated random masking of weights during training can account for the effect. In our CIFAR-100 CLIP experiments (Figure A6, A14), both baselines underperform FPE: the non-disjoint FPE model performs worse than FPE and often worse than the dense model at large expansion factors, and the DropConnect model, despite having access to more active weights at test time (dense $p\mathbf{W_1}'$), still remains below both the dense model and FPE. These results support our claim that FPE's gains arise specifically from its structured, disjoint allocation of incoming weights that reduces feature collisions, rather than from dropout-style random masking alone.

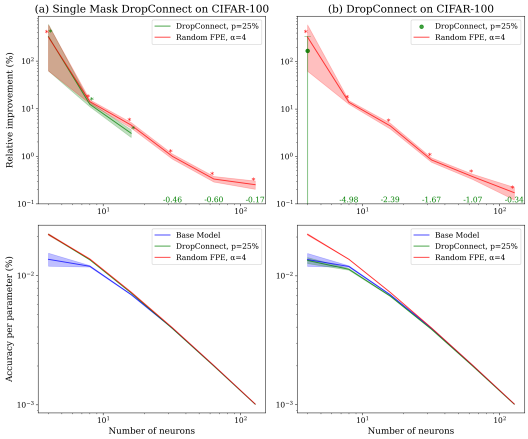

Figure A6: Comparison of FPE to DropConnect-style controls. (a) Single DropConnect mask instantiation on CLIP-embeddings of CIFAR-100, analogous to non-disjoint FPE. (b) DropConnect with random mask sampling during training and expected value weight scaling during evaluation. Relative improvement is calculated as before. Error bars indicate one standard error of the mean. * indicates $p < 0.05$. Results collected over ten trials.

Table A13: Single Mask DropConnect

Number of clauses	Dense	Single Mask DC	Random-Split FPE (ours)
4	32.7 ± 3.8	$\mathbf{51.0 \pm 0.6}$	50.9 ± 0.7
8	57.8 ± 0.7	64.8 ± 0.1	$\mathbf{65.8 \pm 0.2}$
16	69.7 ± 0.5	71.8 ± 0.1	$\mathbf{72.8 \pm 0.1}$
32	76.4 ± 0.1	76.0 ± 0.1	$\mathbf{77.1 \pm 0.1}$
64	78.6 ± 0.1	78.1 ± 0.1	$\mathbf{78.8 \pm 0.1}$
128	79.2 ± 0.0	79.1 ± 0.0	$\mathbf{79.4 \pm 0.0}$

Table A14: DropConnect

Number of clauses	Dense	DropConnect	Random-Split FPE (ours)
4	32.7 ± 3.8	32.2 ± 1.7	$\mathbf{51.2 \pm 0.7}$
8	57.8 ± 0.7	54.9 ± 0.8	$\mathbf{65.7 \pm 0.2}$
16	69.7 ± 0.5	68.0 ± 0.5	$\mathbf{72.8 \pm 0.1}$
32	76.4 ± 0.1	75.1 ± 0.1	$\mathbf{77.0 \pm 0.1}$
64	78.6 ± 0.1	77.7 ± 0.1	$\mathbf{78.9 \pm 0.1}$
128	79.2 ± 0.0	78.9 ± 0.1	$\mathbf{79.3 \pm 0.0}$

A.9 Circuit Analysis and Semantics of Disentangled Features

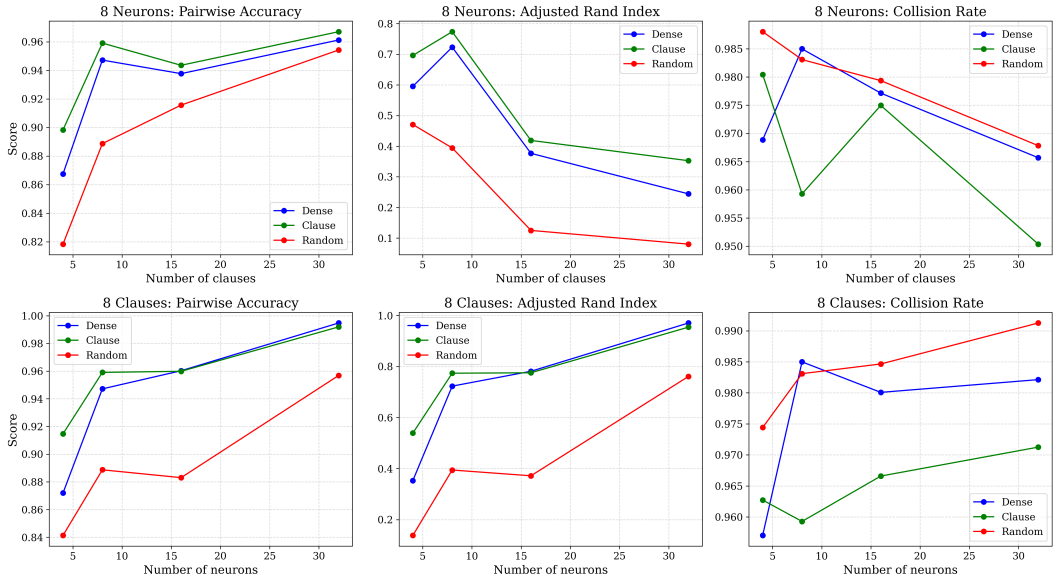

Figure A7: Sweep of model configurations evaluating three metrics: Pairwise Accuracy, Adjusted Rand Index (ARI), and Collision Rate. Top row: models with 8 neurons while varying the number of clauses. Bottom row: models with 8 clauses while varying the number of neurons. Each plot compares Dense, Clause-based, and Random connectivity patterns.

FPE is used in our work primarily as a tool for experimentally probing how width and fixed-parameter sparsity interact to reduce interference, not as a dedicated interpretability method. Our core thesis concerns the relationship between neuron count, collisions, and performance under a fixed non-zero parameter budget, rather than semantic or mechanistic interpretability. However, we recognize its connection to interpretability, and here explore preliminary mechanistic evidence to reveal reduced entanglement in clause-split architectures by conducting two additional empirical analyses: (1) an edge-level collision metric that directly measures circuit overlap, and (2) a semantic clustering analysis of hidden representations via Gram-matrix structure. Together, these analyses, though not intended as a full interpretability study, provide initial mechanistic and representational evidence that neuron expansion reduces interference and increases clause-specific structure.

Experimental Setup for the Model Sweep

For the architectural sweeps shown in Fig. A7, all reported values are averaged over 10 independently trained models per configuration. Dense models were first trained for 25 epochs, after which their neurons were split. The resulting split models were then trained for an additional 25 epochs. This ensures that dense and split variants receive comparable optimization budgets.

For the sweeps, we explored two complementary axes of variation: (1) we fixed the number of clauses at 8 and varied the number of neurons, and (2) we fixed the number of neurons at 8 and varied the number of clauses. In all experiments, each clause contained 4 literals, ensuring consistent clause size across model families.

Mechanistic Edge-Collision Analysis

We add an additional mechanistic edge-collision metric through circuit analysis, designed to trace how individual hidden-layer edges are shared across clause computations. For each test sample, we identify the clause that is satisfied and trace the hidden-layer edges that contribute to its activations. The collision rate counts the proportion of edges used by at least 2 distinct clauses. Lower collision rates correspond to cleaner circuit separation and reduced interference. As shown in Fig. A7 (top–right and bottom–right panels), clause-split models generally achieve lower collision rates.

This suggests that neuron expansion yields circuits with less overlap and more clause-specific routing.

Clause-Aligned Representations via Gram-Matrix Clustering

To additionally assess whether neuron expansion induces clause-aligned representations, we also cluster the hidden-layer Gram matrix using k-means and compare how well the resulting groups recover the ground-truth clause structure. We report two quantitative metrics:

- Pairwise Agreement: the fraction of literal pairs for which the model's cluster membership agrees with the true clause grouping.
- Adjusted Rand Index (ARI): a standard clustering-agreement score (ARI = 1 indicates perfect recovery).

Across all tested configurations, clause-split models show higher pairwise agreement and ARI than the unexpanded baseline, suggesting that expansion pushes features toward clause-aligned, semantically coherent groups. These improvements are visible in Fig. A8, where the clustered Gram matrices form clearer block structures reflecting the underlying clause partition. Dense models also improve as neurons increase, but likely for a different reason: with enough neurons, even dense models have the capacity to represent all clause features fully, causing their performance to converge toward the clause-split models. However, at low and medium neuron counts, where there is greater superposition pressure, the clause-split architecture more consistently recovers clause-aligned structure. The Gram-matrix visualizations confirm this: clause-split models form clean block-diagonal clusters early, whereas dense models only approximate this structure once further parameterized.

Together, the collision analysis and the semantic clustering provide complementary initial evidence that neuron expansion reduces interference, improves clause alignment, and yields more interpretable hidden features. We view these mechanistic analyses as preliminary but promising evidence that neuron expansion reduces circuit-level interference. While the separation between dense and clause-split models is not extreme in all configurations, the trend is consistent across runs: clause-split models exhibit lower collision rates and higher clause-recovery scores. We include these analyses to illustrate the mechanism suggested by our theoretical model, and we view more comprehensive mechanistic studies, such as scaling these analyses to deeper or real-world networks, as an important direction for future work.

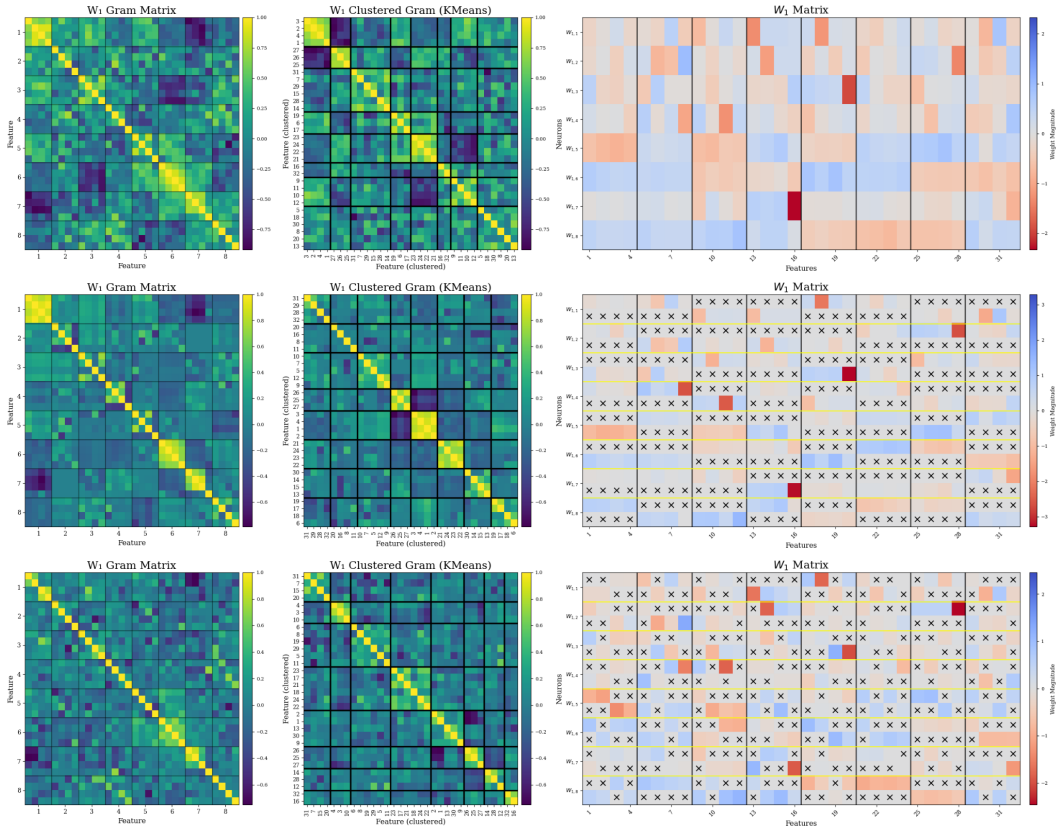

Figure A8: Gram matrix, clustered gram matrix (by k-means), and W_1 matrix for a boolean network of size: 32 literals, 4 literals per clause, and 8 neurons. Top row: dense model, middle row: clause-split model, bottom row: randomly-split model.