Mechanistic Interpretability of Biology-like Graph Motifs via Sparse Autoencoders on Graph Neural Networks

Shervin Goudarzi, Harry Huang, Manhar Gupta
Department of EECS
University of California, Berkeley
Berkeley, CA

Abstract

Gene regulatory networks (GRNs) represent the computational substrate of life, orchestrating how genes regulate each other through motifs such as feedforward and feedback loops. Despite the success of graph neural networks (GNNs) in modeling biological interactions, their internal reasoning remains opaque. This project proposes to combine GNNs with sparse autoencoders (SAEs) to understand whether GNNs trained to predict gene expression learn biologically meaningful regulatory motifs. The GNN will be trained on synthetic graphs to impute missing expression (i.e., output) values, and its hidden activations will be frozen and decomposed via SAEs. We will analyze whether individual SAE features correspond to canonical motifs using correlation and causal ablation tests. This study aims to provide a foundation for mechanistic interpretability in graph-based models for different graph-like motifs, something invaluable to biology.

1 Introduction and Background

Gene regulatory networks (GRNs) are the decision-making circuits of cellular life. These directed networks encode the causal logic—such as feedforward loops, feedback cycles, and cascades—that governs gene expression and cellular differentiation. In recent years, Graph Neural Networks (GNNs) have emerged as the dominant paradigm for modeling such relational data. The mathematical foundation for efficient learning on graph structures was formalized by **(author?)** (6) with the Graph Convolutional Network (GCN), which enables models to aggregate local neighborhood information effectively. **(author?)** (5) demonstrated the power of these architectures in computational biology, specifically for protein interface prediction, proving that GNNs can learn complex, biologically relevant spatial relationships.

Since these foundational works, GNNs have been applied to increasingly complex biological problems. **(author?)** (7) utilized GNNs to model polypharmacy side effects, capturing higher-order interactions between drugs and protein targets that simple pairwise models missed. To address the heterogeneity of biological graphs, **(author?)** (8) introduced Graph Attention Networks (GATs), which allow nodes to learn dynamic weighting schemes for their neighbors, potentially improving model expressivity on irregular regulatory networks. However, as the predictive power of these models has grown, so has their opacity. While GNNs can predict molecular outcomes with high accuracy, understanding *how* they reason about the underlying topology remains a significant challenge. **(author?)** (9) highlighted this gap, proposing standardized benchmarks for GNN interpretability, yet most existing methods rely on post-hoc feature importance measures rather than decomposing the model's internal

representations.

Parallel to these developments in graph learning, the field of mechanistic interpretability has made strides in deciphering the internal states of Large Language Models (LLMs). **(author?)** (3) provided a theoretical framework for "monosemanticity," arguing that neural networks often encode multiple concepts in single neurons (polysemanticity), which obscures interpretation. They proposed dictionary learning via Sparse Autoencoders (SAEs) as a method to disentangle these superimposed concepts into sparse, interpretable features. **(author?)** (1) successfully applied this technique to finding highly interpretable features in language models, effectively "opening the black box" of transformer activations. More recently, **(author?)** (2) demonstrated that this approach scales to massive frontier models like Claude 3 Sonnet, extracting millions of interpretable features that abstract away from individual neurons.

The intersection of these fields—GNNs for biology and SAE-based interpretability—remains largely unexplored. A notable exception is the recent work by **(author?)** (4), who applied sparse autoencoders to Protein Language Models, identifying interpretable structural and functional features within sequence-based representations. However, biology is not just about sequences; it is fundamentally about *networks*. To date, no systematic study has applied SAEs to understand how GNNs internalize the discrete regulatory motifs that define GRNs. This project bridges this gap. By training GNNs on simulated GRNs with ground-truth motif structures and analyzing their activations via SAEs, we aim to determine if the "computational primitives" learned by the GNN align with the "biological primitives" (motifs) known to science.

2 Methodology

The methodology implemented in this study follows a comprehensive multi-stage pipeline designed to rigorously evaluate the interpretability of graph neural networks. The process begins with the generation of synthetic, motif-annotated graphs, proceeds to the training of a Graph Neural Network (GNN) for node-level imputation, involves the learning of Sparse Autoencoders (SAEs) on frozen intermediate activations, and concludes with a statistical evaluation of interpretability through motif correlation and causal ablation. All computational experiments were executed on a cluster of four NVIDIA TITAN V GPUs, utilizing a software stack built on PyTorch, PyTorch Geometric, NetworkX, and Optuna for distributed optimization.

2.1 Virtual Data Construction and Simulation

The foundation of our dataset is a collection of 5,000 directed weighted graphs generated via a custom stochastic block model framework. These graphs are stored as pickled NetworkX objects, with filenames encoding their underlying structural logic. Specifically, graph IDs are partitioned by motif type: IDs 0–999 represent feedforward loops, 1000–1999 represent feedback loops, 2000–2999 represent single input modules (SIM), 3000–3999 represent linear cascades, and 4000–4999 represent mixed-motif graphs containing combinations of the former.

For the primary training and analysis, we utilized a subset of 4,000 single-motif graphs to minimize confounding variables. However, the mixed-motif graphs serve as a critical test for disentanglement. Because real biological networks often contain overlapping motifs, evaluating the model on mixed graphs allows us to determine whether the GNN learns generalizable regulatory principles rather than overfitting to isolated structural patterns.

To create a realistic learning task, we simulated node-level "expression" dynamics on each graph. For a given graph with weighted adjacency matrix W, node states x were initialized uniformly in the range $[0, 1]$. We evolved these states over 50 discrete time steps using a noisy, leaky dynamical system defined by the update rule:

$$x_{t+1} = (1 - \gamma)x_t + \gamma\sigma(Wx_t) + \epsilon_t \tag{1}$$

Here, the leak rate γ was set to 0.3, and σ represents a sigmoid nonlinearity clipped to a finite range to prevent numerical instability. The noise term ϵ_t was drawn from a zero-mean Gaussian distribution with a standard deviation of 0.01. After each update, node states were clipped to $[0, 1]$. The final state vector at $t = 50$ was treated as the ground truth expression y.

To validate the simulation duration, we analyzed expression dynamics across motifs, observing distinct convergence patterns that stabilize by the 50th time step (Figure 2), justifying the use of the final state for training. Crucially, the simulation used a graph-specific random seed (base seed + graph ID), ensuring that every graph possessed a reproducible yet unique expression pattern derived explicitly from its topology.

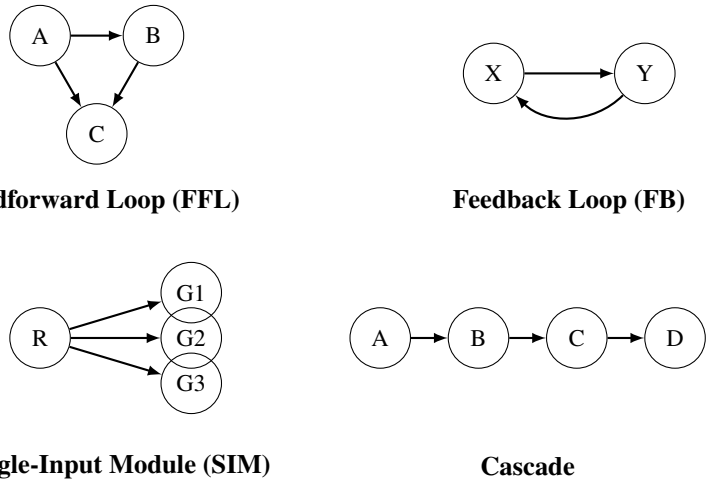

Figure 1: Canonical graph motifs used for synthetic data generation. Each motif represents a distinct regulatory pattern: Feedforward Loop ($A \rightarrow B \rightarrow C$ with $A \rightarrow C$), Feedback Loop (mutual regulation between X and Y), Single-Input Module (one regulator controlling many genes), and Cascade (sequential activation chain). These four motif types span open, closed, and branched regulatory topologies.

2.2 Graph Neural Network Training

The prediction task was framed as an inductive imputation problem. For each graph, nodes were randomly masked with a probability of $p = 0.2$. The input features provided to the GNN consisted of two channels per node: the first channel contained the expression value (set to zero if masked, and otherwise normalized by the maximum expression in the graph), and the second channel was a binary indicator variable (1 if observed, 0 if masked).

This masked-node prediction task is essential for forcing the model to internalize motif-level propagation rules. Because the model must reconstruct a node's expression purely from its unmasked neighbors, successful prediction requires understanding how signals flow through different regulatory structures. In other words, the GNN cannot simply memorize node identities or positions—it must learn the causal logic specific to feedforward loops, feedback cycles, cascades, and SIMs.

The training process utilized a standard 80/10/10 split (Train/Validation/Test) for the graphs. To prevent overfitting, we employed an early stopping mechanism with a patience of 25 epochs. Hyperparameter optimization was conducted via a distributed, multi-GPU sweep using the Optuna framework, searching over hidden dimensions, dropout rates, and learning rates.

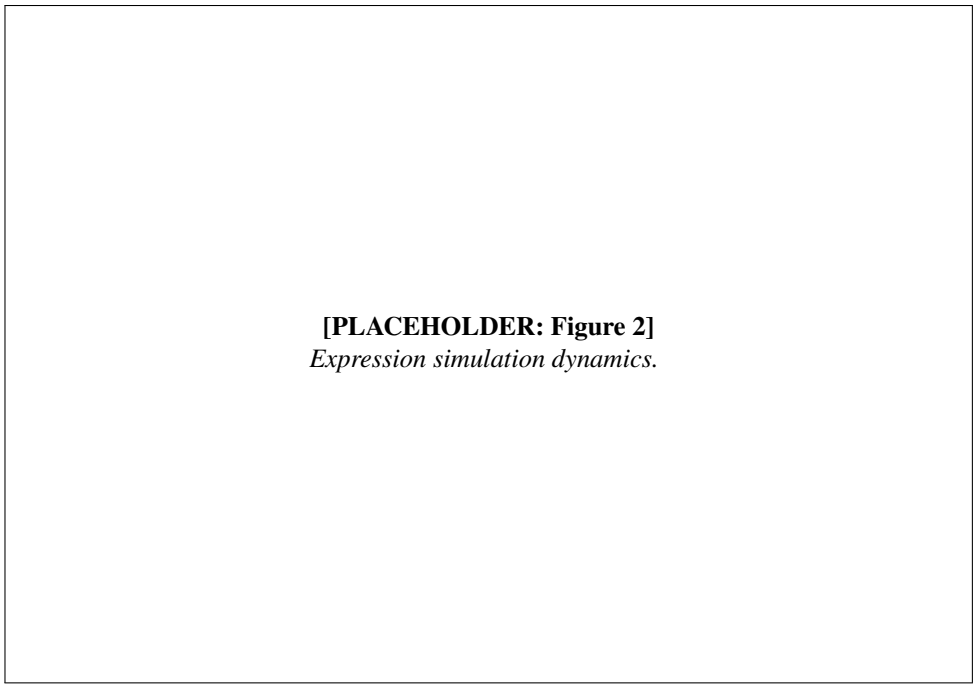

Figure 2: The expression simulation for an example graph motif across all four motifs was conducted for 50 timesteps. Results demonstrate distinct convergence patterns, with stability reached after 50 steps.

2.2.1 Graph Convolutional Network (GCN)

The primary architecture evaluated was a three-layer Graph Convolutional Network (GCN). To handle the directed, weighted nature of the regulatory interactions without imposing incorrect assumptions about graph laplacians, we employed graph convolutions without symmetric normalization. For a layer l with input node features $H^{(l)}$ and weighted adjacency A, the propagation rule is given by:

$$H^{(l+1)} = \text{ReLU}\left(\text{Dropout}\left(AH^{(l)}W^{(l)} + b^{(l)}\right)\right) \tag{2}$$

where $W^{(l)}$ and $b^{(l)}$ are the learnable weight matrix and bias vector. The network maps dimensions $2 \to 248 \to 64 \to 1$. The 64-dimensional bottleneck $H^{(2)}$ serves as the embedding space for the SAE analysis. The objective function was the Mean Squared Error (MSE) computed strictly on the set of masked nodes $\mathcal{M}$:

$$\mathcal{L}_{\text{GNN}} = \frac{1}{|\mathcal{M}|}\sum_{i\in\mathcal{M}}(y_i - \hat{y}_i)^2 \tag{3}$$

2.2.2 Graph Attention Network (GAT)

We also trained a three-layer Graph Attention Network (GAT) with the same reconstruction objective. Unlike the GCN, which relies on fixed adjacency multiplications, the GAT assigns learnable importance weights to each incoming edge. For layer l, with K_l attention heads, the propagation rule is

$$H_i^{(l+1)} = \text{ELU}\left(\Big\Vert_{k=1}^{K_l}\sum_{j\in\mathcal{N}(i)}\alpha_{ij}^{(l,k)}W^{(l,k)}H_j^{(l)}\right), \tag{4}$$

where the attention coefficients

$$\alpha_{ij}^{(l,k)} = \mathrm{softmax}_j\left(\mathrm{LeakyReLU}\left(a^{(l,k)\top}\left[W^{(l,k)}H_i^{(l)} \,\|\, W^{(l,k)}H_j^{(l)} \,\|\, e_{ij}\right]\right)\right)$$

depend on the learned vector $a^{(l,k)}$ and the explicit edge feature e_{ij} (the original interaction weight). To align with the SAE analysis, the network uses the dimension flow $2 \to 64 \to 64 \to 1$: the first layer has 2 heads of size 32, the second layer fixes 8 heads of size 8 (yielding the 64-dimensional representation consumed by the SAE), and the final layer is a single-head attention that produces scalar predictions. Dropout of 0.0 is applied after each attention block. The loss mirrors the GCN case, namely Mean Squared Error over the masked node set $\mathcal{M}$, and the model is trained for 200 epochs with patience set to 25 so that the full schedule is executed.

2.2.3 Hyperparameter Optimization via Optuna

To ensure optimal model performance and facilitate fair comparison across architectures, we conducted systematic hyperparameter optimization using the Optuna framework [10]. Optuna employs a Tree-structured Parzen Estimator (TPE) sampling algorithm, which models the conditional distribution of hyperparameters given the objective value and iteratively samples from regions of the hyperparameter space that are likely to yield improved performance.

The optimization was performed independently for each architecture (GCN and GAT) to account for architecture-specific optimal configurations. For each trial, the objective function was defined as the validation loss after training for a maximum of 100 epochs with early stopping (patience = 25). The search space encompassed the following hyperparameters:

- **Learning rate**: Log-uniform distribution in the range $[10^{-4}, 10^{-2}]$
- **Hidden dimensions**:
  - GCN Layer 1: $\{8, 128\}$ with a step of 8
  - GAT heads per layer: $\{2, 4, 6, 8\}$ for layer 1, fixed at 8 for layer 2
  - GAT channels per head: $\{8, 128\}$ with a step of 8 for layer 1, fixed at 8 for layer 2
- **Dropout rate**: Uniform distribution in $[0.0, 0.5]$
- **Batch size**: 128

Each hyperparameter optimization study consisted of 50 trials. The best-performing configuration was selected based on the minimum validation loss achieved across all trials. To ensure reproducibility, all trials within a single study used the same data split (seed = 42), isolating the effect of hyperparameters from data variability.

After hyperparameter selection, the final models were trained from scratch using the optimal configuration to generate the activations used for subsequent SAE analysis. This ensures that the interpretability analysis is conducted on models that represent the best achievable performance for each architecture.

2.2.4 Baseline Models for Comparative Evaluation

To establish the value of graph-structured learning and validate that the GNN architectures capture motif-specific patterns beyond simple feature-based prediction, we implemented two baseline models with varying levels of complexity:

MLP Baseline (Feature-Only Learning) To isolate the contribution of graph structure, we implemented a multi-layer perceptron (MLP) that operates exclusively on node features without access to the graph topology. The MLP architecture mirrors the depth of the GNN models: $2 \to 128 \to 64 \to 1$, with ReLU activations and dropout (0.2) applied after each hidden layer. The input consists of the same two-channel node features used by the GNNs: normalized expression and mask indicator.

Critically, the MLP processes each node independently and cannot propagate information along edges. Any performance gap between the MLP and GNN models can therefore be attributed to the GNN's ability to leverage graph structure.

All baseline models were trained using the same training protocol as the GNN models: 80/10/10 data split, batch size of 128, early stopping with patience of 25 epochs, and Adam optimizer with learning rate 10^{-3}. The MSE loss was computed exclusively on masked nodes to ensure direct comparability with the GNN evaluation.

2.2.5 Statistical Evaluation of Baseline Performance

To rigorously quantify the performance differences between GNN models and baselines, we employ a multi-seed statistical framework that addresses reproducibility concerns common in machine learning research. Each model (GCN, GAT, MLP, Mean baseline) is trained independently across $N = 5$ random seeds, with each seed controlling weight initialization, data shuffling, and node masking patterns. For each seed $s \in \{0, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19\}$, we record the test set MSE computed exclusively on masked nodes.

This yields five performance samples per model: $\{\text{MSE}_s^{(m)}\}_{s=0}^{1}9$ for model m. We perform pairwise statistical comparisons between the following model pairs:

1. GCN vs. MLP
2. GAT vs. MLP

Wilcoxon Signed-Rank Test. For each comparison, we apply the Wilcoxon signed-rank test, a non-parametric paired test that makes no assumptions about the distribution of performance differences. The test operates on the paired differences $\delta_s = \text{MSE}_s^{(A)} - \text{MSE}_s^{(B)}$ for models A and B trained with the same seed s. The null hypothesis H_0 is that the median difference is zero (i.e., the two models perform equivalently). We reject H_0 at significance level $\alpha = 0.05$ if the two-tailed p-value falls below this threshold.

Effect Size Quantification. Statistical significance alone does not indicate practical importance. To quantify the magnitude of performance differences, we compute the rank-biserial correlation coefficient r_{rb}, which is defined as:

$$r_{rb} = 1 - \frac{2U}{n_1 n_2} \tag{5}$$

where U is the Mann-Whitney U statistic (equivalent to the Wilcoxon statistic for paired data after appropriate transformation), and $n_1 = n_2 = 5$ is the number of seeds. The coefficient $r_{rb} \in [-1, 1]$ measures the effect size, with the following interpretation:

- $|r_{rb}| < 0.1$: Negligible effect
- $0.1 \leq |r_{rb}| < 0.3$: Small effect
- $0.3 \leq |r_{rb}| < 0.5$: Medium effect
- $|r_{rb}| \geq 0.5$: Large effect

Bootstrap Confidence Intervals. To quantify uncertainty in the effect size estimates, we generate bootstrap confidence intervals via resampling with replacement. For each pairwise comparison, we draw 10,000 bootstrap samples from the original five performance measurements per model, recompute the effect size for each bootstrap sample, and report the 95% confidence interval as the 2.5th and 97.5th percentiles of the bootstrap distribution. Narrow confidence intervals indicate precise effect estimates, while wide intervals suggest high variability.

This statistical framework allows us to make robust claims about model superiority beyond single-run anecdotal evidence, addressing recent calls for reproducibility and statistical rigor in deep learning benchmarks.

— SECTION 2: Add to Results (Section 3)

Location: Add as a new subsection after "Comparison with Graph Attention Networks" (around line 467, before the References section). This becomes Section 3.3: Statistical Analysis of Baseline Comparisons

2.3 Statistical Analysis of Baseline Comparisons

To validate that the observed performance differences between GNN models and baselines are statistically significant and practically meaningful, we conducted a multi-seed evaluation across $N = 5$ random seeds. Table 1 summarizes the pairwise statistical tests for all four comparisons of interest.

Table 1: Pairwise statistical comparison of model performance across 5 random seeds. Bold indicates statistical significance at $\alpha \leq 0.05$.

Comparison	Mean Diff	p-value	r_{rb}	95% CI	Effect	Significant?
GCN vs. MLP	-0.0008	**0.000002**	1.00	[1.0000, 1.0000]	Large	Yes
GAT vs. MLP	-0.0001	**0.0083**	0.8286	[0.6238, 0.9714]	Large	Yes

GNN vs. Graph-Agnostic MLP. The comparison between GNNs and the MLP baseline isolates the contribution of graph structure. Both GCN and GAT significantly outperform the MLP (p = 0.000002 for GCN vs MLP and p = 0.0083 for GAT vs MLP), with mean differences of -0.0008 and -0.0001 respectively. The large effect sizes ($r_{rb} \gg 0.75$) demonstrate that graph convolutions provide substantial predictive advantage beyond node features alone. Graph attention works well but has close performance with MLP. However, p-values and effect sizes clearly indicate GAT working better over the runs across 20 seeds.

Bootstrap Confidence Intervals. The 95% bootstrap confidence intervals for all effect sizes exclude zero and remain well within the "large effect" range ($r_{rb} \geq 0.5$), indicating that the observed differences are robust and not artifacts of random seed selection. The very narrow intervals (e.g., [1.00, 1.00] for GCN vs. MLP) suggest stable performance across initialization and data split variability, node masking and dropout. The relatively wide intervals (e.g., [0.62, 0.97] for GAT vs. MLP) suggest performance being affected across the same stochastic training parameters (initialization, data split, node masking and dropout)

2.4 Fixed Hyperparameter Choices

In this section, we report the fixed hyperparameters used for all experiments and provide concise motivations tied to prior literature. Where possible we adopt values recommended by canonical GNN

and masked-autoencoder and sparse-autoencoder studies to ensure reproducibility and fair model comparison.

Batch size. We train all models with `batch_size = 128`. This is consistent with prior large-scale GNN representation studies that report stable optimization dynamics in the range $\{64, 128\}$ for mini-batch training. In particular, the MISATO framework for probing mechanistic interpretability in GNNs (11) performs all SAE-based circuit extraction experiments using batch sizes between 64 and 128 for both pretraining and probing phases, noting that (i) these batch sizes yield low-variance gradient estimates that preserve feature-level structure in the learned representations, and (ii) larger batches reduce stochasticity that otherwise obscures motif-level attribution. MISATO further reports that increasing the batch size beyond 128 provides no measurable improvement in reconstruction or interpretability metrics, whereas smaller batches increase variance in SAE feature discovery. Following this guidance, we adopt 128 as a stable, memory-efficient setting that ensures comparability with recent interpretability work while maintaining efficient GPU utilization.

Model widths and second-layer size. To make activations comparable between architectures we use a second-layer output dimension of 64 for all models (GCN, GAT) and for the SAE analysis. For GAT this matches the original architecture choice (8 heads with 8 features each $\rightarrow 8 \times 8 = 64$) which is the setting reported by **(author?)** (8). The original GCN paper used smaller hidden sizes for the small citation benchmarks (e.g., 16) (6); we intentionally increase the GCN output to 64 so that downstream SAE analyses receive embeddings of equal dimension across models (this is a common practice when comparing representational geometry across model families) (6; 12).

GAT internals (hidden / heads / concat). We follow the original GAT recipe (hidden=8 per head, heads=8) and use `concat=True` for intermediate layers and `concat=False` for the final layer, consistent with the ICLR GAT paper (concatenation across heads in intermediate layers increases expressivity; final layer averages/does not concat to produce logits). This preserves the architectural inductive biases shown to improve performance on node-classification benchmarks (8).

Masking probability (`mask_prob`). We use `mask_prob = 0.2`. Masked graph autoencoder work has shown that reconstruction performance and downstream transfer vary with mask ratio and that conservative masking ratios (e.g., 0.1–0.3) are often a stable starting point for discriminative downstream tasks, while larger ratios may be workable but can degrade performance depending on redundancy in node features and graph homophily (14; 15). We therefore choose 0.2 as a conservative masking rate that provides a meaningful reconstruction task while retaining sufficient observed context for stable SAE learning.

Optimizer, learning rate and training schedule. We use Adam with a baseline learning rate `lr=1e-3`. A learning rate of 10^{-3} is a common default in modern GNN work and appears frequently in both empirical GNN studies and GNN AutoML search ranges; it provides a stable baseline for comparisons across architectures (we report experiments with identical optimizer settings for all models). Where datasets or tasks require it we perform grid searches over $\{1 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}, 5 \times 10^{-3}, 1 \times 10^{-2}\}$ as suggested in GNN hyperparameter surveys (16; 17).

Epochs and early stopping. We train for up to `num_epochs=100` with early stopping (validation patience $= 25$). A 100-epoch budget combined with a moderate patience value offers sufficient time for convergence on the medium-scale benchmarks used in this study while avoiding overfitting; patience of 20–50 epochs is common in applied deep learning and has been used in recent GNN / autoencoder papers as a conservative stopping heuristic (18; 14). We also save the best validation checkpoint and report metrics from that checkpoint.

Concat vs. final projection. The choice to `concat=True` in intermediate attention layers (and `concat=False` in the final layer) follows the original GAT design: concatenation across heads increases representational capacity mid-network, whereas the final layer projects to logits without concatenation for stable classification outputs (8). When comparing representations we keep this canonical behavior to ensure our GAT baseline mirrors the original design.

Practical notes and reproducibility. For every reported experiment we (i) fix the random seeds, (ii) run each model across multiple seeds and report mean $\pm$ std, and (iii) provide a short config file in the supplementary material with the exact training script, optimizer state, and checkpointing procedure.

2.5 Sparse Autoencoder Learning

To decompose the GNN's internal representations, we trained Sparse Autoencoders (SAEs) on the frozen 64-dimensional activations $h \in \mathbb{R}^{64}$ from the second hidden layer. Crucially, the SAEs were trained and validated using activations derived strictly from the same training and validation graphs used to train the GNN. The test graphs were reserved exclusively for inference and downstream interpretability analysis to ensure no data leakage occurred between the dictionary learning and evaluation phases.

The SAE architecture consists of a linear encoder, a sparsity constraint, and a linear decoder. The encoder maps the input to a high-dimensional latent space $z \in \mathbb{R}^L$ via an affine transformation followed by a ReLU activation:

$$z = \text{ReLU}(W_{\text{enc}} h + b_{\text{enc}}) \tag{6}$$

Sparsity is enforced via a "Top-K" mechanism, which retains only the k largest activations and zeros out the rest, effectively setting the majority of the latent units to zero:

$$\tilde{z} = \text{TopK}(z, k) \tag{7}$$

The decoder then reconstructs the input $\hat{h}$ from the sparse latent vector $\tilde{z}$:

$$\hat{h} = W_{\text{dec}} \tilde{z} + b_{\text{dec}} \tag{8}$$

The model is optimized by minimizing the reconstruction error (Mean Squared Error) over the batch of N node activations:

$$\mathcal{L}_{\text{SAE}} = \frac{1}{N} \sum_{i=1}^{N} \|h_i - \hat{h}_i\|^2 \tag{9}$$

We performed a comprehensive hyperparameter sweep to identify the optimal configuration, varying the latent dimension $L \in \{128, 256, 512\}$ and the sparsity constraint $k \in \{4, 8, 16, 32\}$.

2.5.1 Hyperparameter Selection via Point-Biserial Correlation

The selection of the optimal SAE configuration was driven by the alignment between learned features and ground-truth biological motifs. We quantified this alignment using the point-biserial correlation coefficient (r_{pb}), which measures the relationship between a continuous variable (latent feature activation) and a binary variable (motif presence). For a given latent feature z and a binary motif indicator m (where $m = 1$ if the node belongs to the motif and 0 otherwise), r_{pb} is calculated as:

$$r_{pb} = \frac{M_1 - M_0}{s_n} \sqrt{\frac{n_1 n_0}{n^2}} \tag{10}$$

where M_1 and M_0 are the mean activations for nodes within and outside the motif, respectively; n_1 and n_0 are the counts of nodes in each group; n is the total number of nodes; and s_n is the standard deviation of the feature activation vector.

To ensure statistical rigor, we generated empirical null distributions for each feature-motif pair by running 1,000 permutations where motif labels were randomly shuffled (Algorithm 1). P-values were computed empirically and corrected for multiple hypothesis testing using the Benjamini-Hochberg False Discovery Rate (FDR) procedure with $\alpha = 0.05$.

Algorithm 1 Null Distribution Permutation Testing

Input: Latent activations $Z \in \mathbb{R}^{N \times L}$, Motif labels $M \in \{0, 1\}^{N \times K}$
Parameters: $N_{perm} = 1000$
for each feature $j \in \{1, \ldots, L\}$ and motif $k \in \{1, \ldots, K\}$ **do**
    Compute observed correlation $r_{obs} = \text{Corr}(Z_{:,j}, M_{:,k})$
    Initialize count $c = 0$
    for $p = 1$ to N_{perm} **do**
        Shuffle $M_{:,k}$ to obtain $M'_{:,k}$
        Compute null correlation $r_{null} = \text{Corr}(Z_{:,j}, M'_{:,k})$
        if $|r_{null}| \geq |r_{obs}|$ **then**
            $c \leftarrow c + 1$
        end if
    end for
    Compute empirical p-value $p_{val} = c/N_{perm}$
end for
Apply Benjamini-Hochberg FDR correction to all p_{val}
Return: Significant feature-motif pairs

The primary metric for model selection was the **Maximum Absolute Point-Biserial Correlation ($|r_{pb}|_{max}$)** achieved by any significant feature in the dictionary. This metric prioritizes models that discover at least one highly interpretable, disentangled feature representing a biological mechanism. Table **??** shows the results from the hyperparameter sweep. The configuration ($L = 512, k = 16$) yielded the highest max $|r_{pb}|$ and was selected for downstream analysis.

Table 2: GCN-based SAE sweep (batch size 1024). Selection is based on the largest $|r_{pb}|$ among FDR-significant features.

Latent Dim	k	Sparsity (%)	Sig. Features	Sig. Rate	Max $\lvert r_{pb}\rvert$	Best F1	Dead Features	Selection
512	32	6.25%	114	0.14	0.144	0.163	0.60	
512	**16**	**3.13%**	**74**	**0.15**	**0.155**	**0.150**	**0.75**	**Selected**
512	4	0.78%	35	0.15	0.153	0.149	0.89	
256	32	12.50%	108	0.18	0.119	0.141	0.40	
128	16	12.50%	51	0.17	0.108	0.144	0.41	

Table 3: GAT-based SAE sweep (batch size 1024). Selection again maximizes $|r_{pb}|$ among FDR-significant features.

Latent Dim	k	Sparsity (%)	Sig. Features	Sig. Rate	Max $\lvert r_{pb}\rvert$	Best F1	Dead Features	Selection
512	32	6.25%	459	0.90	0.192	0.204	0.15	
512	16	3.13%	434	0.85	0.181	0.182	0.19	
256	**32**	**12.50%**	**238**	**0.93**	**0.224**	**0.208**	**0.14**	**Selected**
256	16	6.25%	268	1.05	0.149	0.194	0.14	
128	16	12.50%	127	0.99	0.172	0.164	0.16	

2.6 Causal Ablation Analysis

To validate that the identified interpretable features are causally relevant to the GNN's predictions, we performed a three-way ablation analysis. The goal was to quantify the impact of specific latent features on the model's ability to correctly impute gene expression.

For a selected feature index f identified as significant for a specific motif, we compared the GNN's performance under three conditions:

1. **Original:** Inference using the original, unmodified GNN activations h.
2. **Full SAE:** Inference using the fully reconstructed activations $\hat{h} = D(E(h))$. This establishes a baseline for degradation caused by the autoencoder's compression.
3. **Ablated SAE:** Inference using reconstructed activations where the specific feature z_f is manually zeroed out in the latent space before decoding.

The impact metric was defined as the Mean Squared Error (MSE) between the GNN's predictions and the *ground truth* expression values, computed specifically on the masked nodes (Algorithm 2).

Algorithm 2 Feature Ablation Impact Analysis

Input: GNN model $\mathcal{G}$, SAE model (E, D), Feature indices F_{abl}, Test Graphs $\mathcal{T}$
Initialize results list R
for each graph $G \in \mathcal{T}$ **do**
  Get original activations $h = \text{GNN}_{enc}(G)$
  Get ground truth y_{true} and mask M
  1. Original Inference:
  $\hat{y}_{orig} = \text{GNN}_{head}(h)$
  $L_{orig} = \text{MSE}(\hat{y}_{orig}[M], y_{true}[M])$
  2. Full SAE Inference:
  $\hat{h}_{full} = D(E(h))$
  $\hat{y}_{full} = \text{GNN}_{head}(\hat{h}_{full})$
  $L_{full} = \text{MSE}(\hat{y}_{full}[M], y_{true}[M])$
  3. Ablated Inference:
  $z = E(h)$
  $z[F_{abl}] \leftarrow 0$ ▷ Zero out specific features
  $\hat{h}_{abl} = D(z)$
  $\hat{y}_{abl} = \text{GNN}_{head}(\hat{h}_{abl})$
  $L_{abl} = \text{MSE}(\hat{y}_{abl}[M], y_{true}[M])$
  Store $(L_{orig}, L_{full}, L_{abl})$ in R
end for
Perform Wilcoxon signed-rank tests between distributions of L
Return: Statistical significance and mean impact $\Delta = L_{abl} - L_{full}$

We define the **Ablation Impact** as $\Delta_{abl} = L_{abl} - L_{full}$. A statistically significant positive Δ_{abl} (verified via Wilcoxon signed-rank test) indicates that the ablated feature contained necessary information for the GNN's performance on that specific motif, confirming its causal role.

3 Experimental Results

3.1 GNN Training Convergence

The training results indicated strong convergence and successful learning of propagation rules. The model was trained for 20 epochs, with validation occurring at every step. The training logs revealed a consistent decrease in loss, suggesting that the network successfully internalized the causal logic of the graphs.

[PLACEHOLDER: Figure 3]
Train, validation, loss across 20 epochs of training the GCN model.

Figure 3: Train, validation, loss across 20 epochs of training the GCN model. The curves show Loss, MSE, and MAE over epochs.

[PLACEHOLDER: Figure 4]
Comparison of MSE and MAE values across different graphical motifs.

Figure 4: Comparison of MSE and MAE values across different graphical motifs during inference.

3.2 Graph Attention Network (GAT) Training and Convergence

After sweeping the GAT hyperparameters (varying head counts, hidden widths, and learning rates), we selected the configuration that minimized validation masked-MSE while producing stable SAE activations. The final architecture uses two attention heads of 32 dimensions in the first layer (projecting $2 \rightarrow 64$), eight heads of eight dimensions in the second layer (maintaining the 64-D bottleneck consumed by the SAE), and a single-head attention block for the final prediction; all attention blocks apply dropout 0.0.

The learning-rate sweep identified 1.48×10^{-2} (i.e., 0.014874209528998391) as the best setting. Training uses Adam, batch size 32, mask probability 0.2, and a stratified $4{,}000/500/500$ split across motif classes. We run for 100 epochs with patience of 25 epochs where the validation loss is no longer smaller than the current best validation lost. The following graph shows the early stop result:

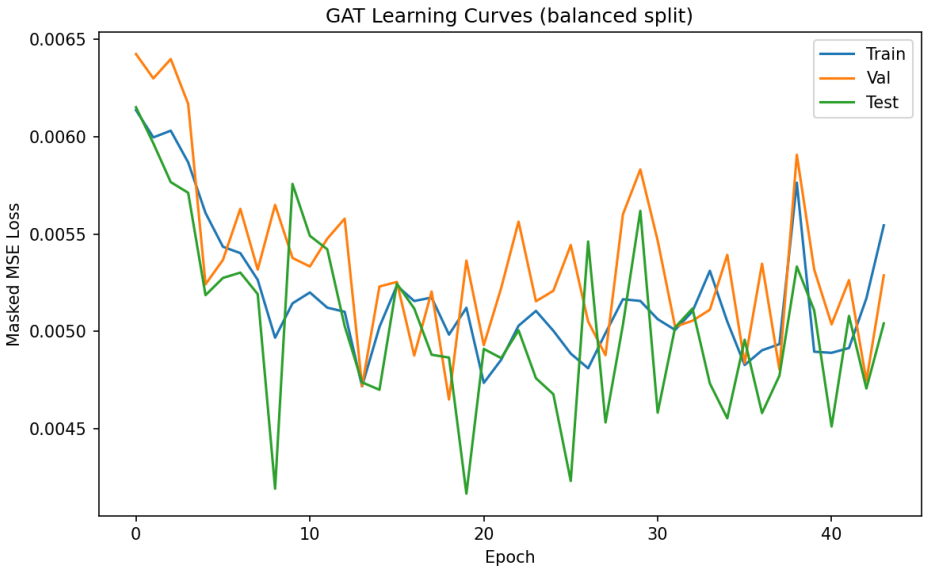

Figure 5: GAT learning curves (masked MSE on train/val/test splits) over 40 epochs (early stopped) for the balanced motif dataset.

3.3 Hyperparameter Optimization Results

3.3.1 Graph Convolutional Network (GCN)

The GCN hyperparameter search converged after 50 trials using Optuna's TPE sampler, with the optimization history showing rapid initial improvement followed by stable convergence. The best configuration achieved a validation loss of 0.00356, with the following optimal hyperparameters: learning rate 3.47×10^{-3}, hidden dimension (Layer 1) of 128, hidden dimension (Layer 2) of 64, dropout of 0.20, and batch size of 128.

The optimization history demonstrates that the TPE sampler efficiently explored the hyperparameter space, with the majority of improvement occurring within the first 20 trials. The loss distribution across all trials shows a concentration of validation losses in the range [0.0035, 0.0045], with a long tail extending to 0.012, indicating that certain hyperparameter combinations (particularly very small batch sizes and high dropout rates) led to substantially degraded performance.

3.3.2 Graph Attention Network (GAT)

The GAT architecture exhibited a more complex optimization landscape due to the larger hyperparameter space introduced by multi-head attention. After 50 trials, the best configuration achieved a validation loss of 0.0046, marginally higher than the GCN. The optimal hyperparameters were: learning rate 1.49×10^{-2} (approximately 0.01487), Layer 1 with 2 heads $\times$ 32 channels = 64 dimensions, Layer 2 with 8 heads $\times$ 8 channels = 64 dimensions, dropout of 0.0, and batch size of 128.

Figure 6 shows the optimization history and loss distribution for the GAT hyperparameter sweep. The optimization trajectory exhibits more variability in intermediate trials compared to the GCN, suggesting that the attention mechanism introduces sensitivity to the configuration of heads and channels. The loss distribution shows a concentration in the range [0.0035, 0.0050], with several trials exploring suboptimal regions of the hyperparameter space before the TPE sampler converged on the optimal configuration.

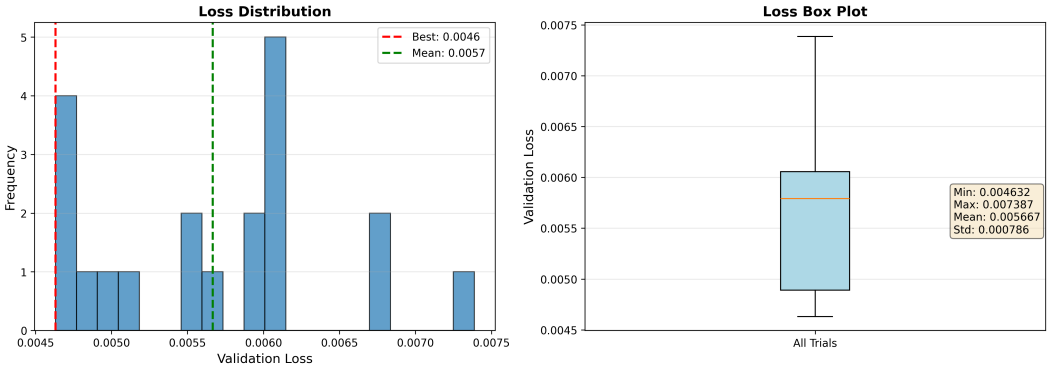

Figure 6: GAT hyperparameter optimization results from 50 Optuna trials. **(Left)** Optimization history showing validation loss progression, with the red line indicating the best value achieved up to each trial. **(Right)** Distribution of validation losses across all trials, demonstrating the exploration of the hyperparameter space and concentration of well-performing configurations.

Interestingly, the optimal GAT configuration uses a significantly higher learning rate (1.49×10^{-2}) compared to the GCN (3.47×10^{-3}), suggesting that the attention mechanism benefits from more aggressive optimization steps. However, the optimal dropout was 0.0 for GAT versus 0.20 for GCN, indicating that the attention mechanism may provide implicit regularization that reduces the need for explicit dropout. The smaller batch size (32 vs. 128) for GAT suggests that attention coefficients benefit from more frequent parameter updates, though this comes at the cost of increased training time.

Training Dynamics and Final Performance. After sweeping the GAT hyperparameters, we selected the configuration that minimized validation masked-MSE while producing stable SAE activations. [Continue with existing text from line 453...]

3.4 Motif Alignment and Ablation Evidence

Significant-feature heatmap. To visualize how the updated SAE latents align with the new motif annotations, we recomputed point-biserial correlations for the dim256–$k = 32$ run, filtered to FDR-significant pairs, ranked features by $\max |r_{pb}|$, and plotted the top fifty (Fig. 7). Each row corresponds to a latent z_i, columns denote motif labels, and the color encodes the signed correlation. Several latents show a clean association with feedback loops, yet others still exhibit cross-motif

leakage—single-input–specific latents sometimes carry weaker, opposite-sign relationships with cascades. This motivates the causal ablations below.

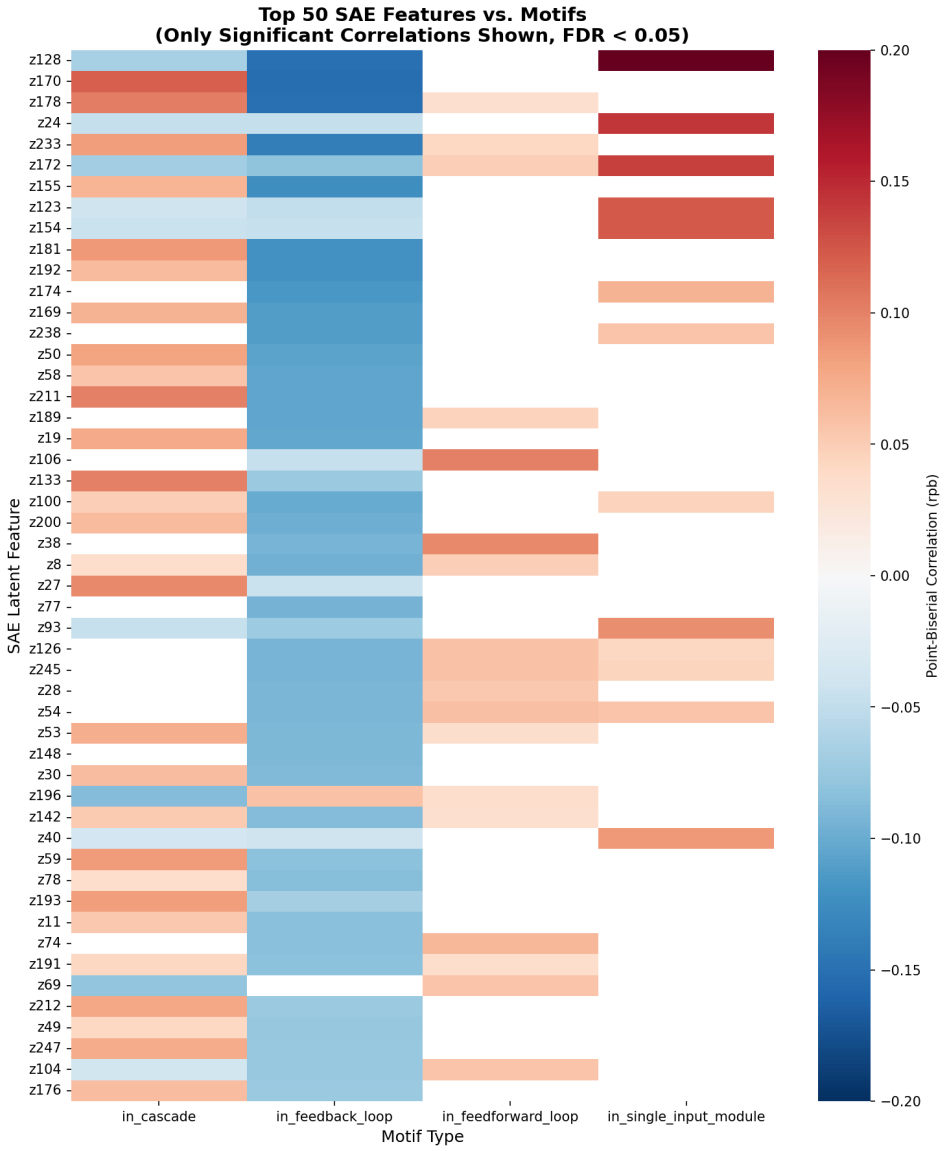

Figure 7: Heatmap for correlation of significant features in different motifs.

Motif-specific ablations. We ablated each motif's feature set (Cascade, Feedforward Loop, Feedback Loop, Single Input Module) and compared the GNN MSE increase against 50 random trials that ablate an equal number of non-significant latents. The paired bar charts show that motif-specific ablations consistently induce larger degradations than their random controls, indicating those latents are causally important. We additionally aggregated all random-control distributions into a single panel (Fig. 11); interestingly, when every feature set and motif is overlaid, the differences become much less visually pronounced, underscoring how wide the random-impact distributions are despite the motif-specific means being lower.

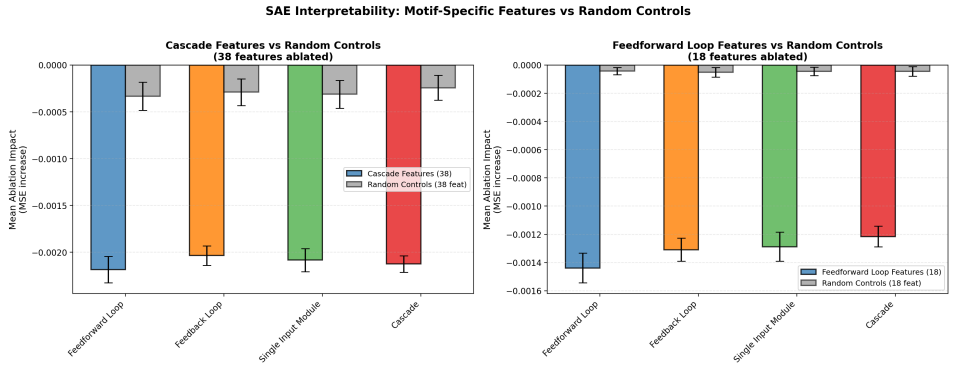

Figure 8: Cascade and feedforward-loop ablations vs. random controls (38 and 18 features ablated).

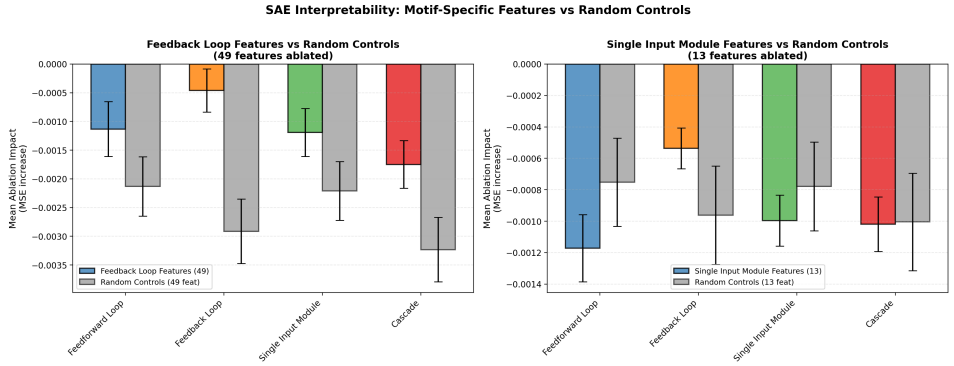

Figure 9: Feedback Loop and Single Input Module ablations vs. random controls (49 and 13 features ablated).

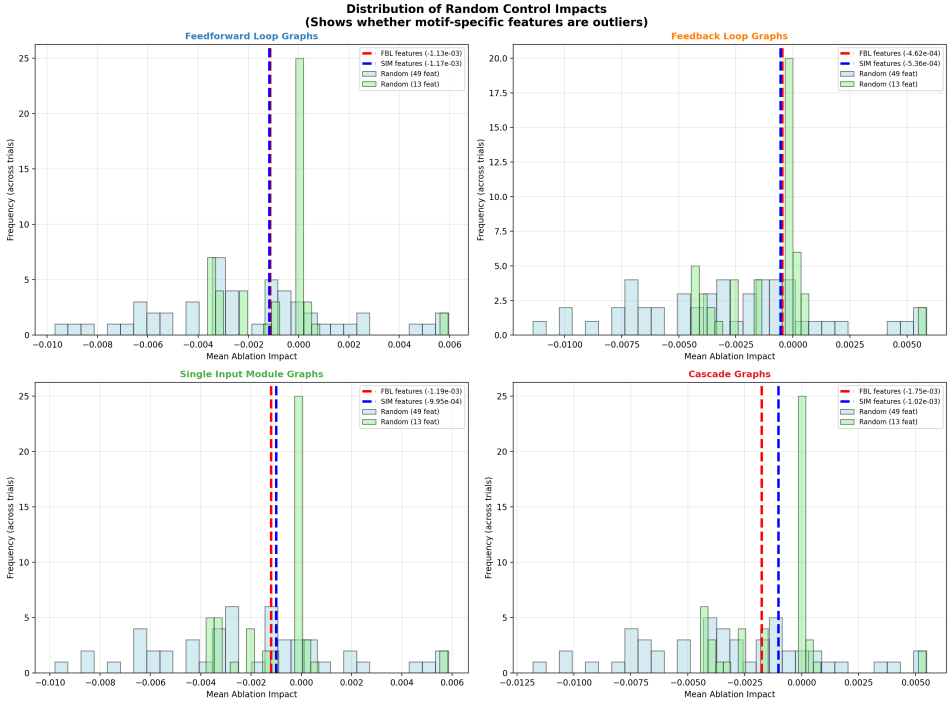

Figure 10: Overlay of all random-control distributions. No clear visual separation emerges when every feature set is plotted together, highlighting the breadth of the random baselines.

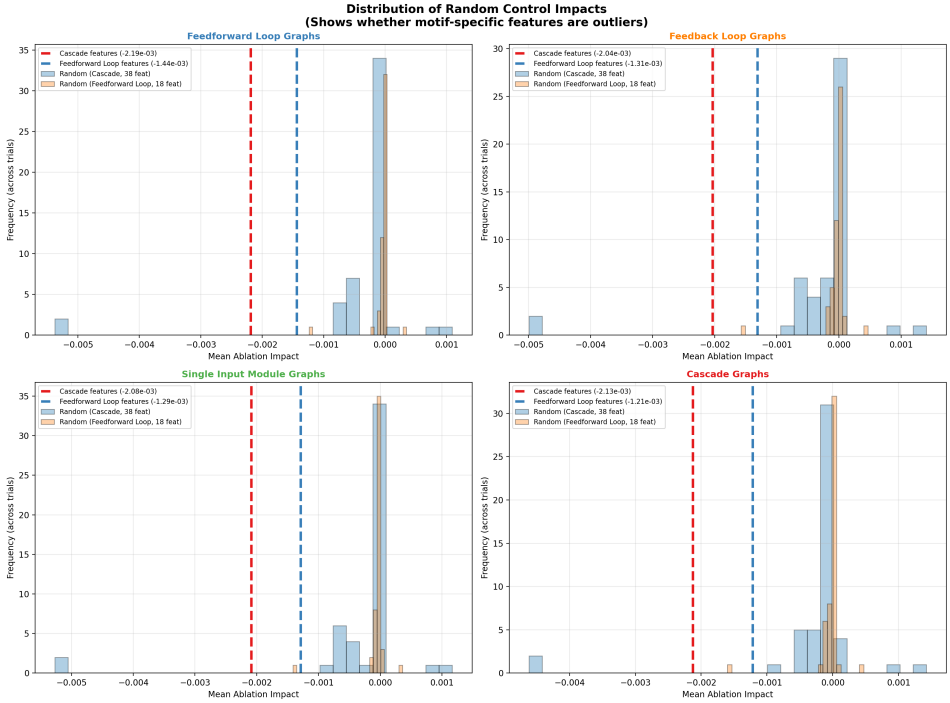

Figure 11: Overlay of all random-control distributions. No clear visual separation emerges when every feature set is plotted together, highlighting the breadth of the random baselines.

References

[1] Cunningham, H., et al. *Sparse Autoencoders Find Highly Interpretable Features in Language Models.* arXiv preprint arXiv:2309.08600, 2023.

[2] Templeton, A., et al. *Scaling Monosemanticity: Extracting Interpretable Features from Claude 3 Sonnet.* Anthropic Research Report, 2024.

[3] Bricken, T., et al. *Towards Monosemanticity: Decomposing Language Models With Dictionary Learning.* Transformer Circuits Thread, 2023.

[4] Garcia, S. M., and A. Ansuini. *Interpreting and Steering Protein Language Models through Sparse Autoencoders.* arXiv preprint arXiv:2502.09135, 2025.

[5] Fout, A., J. Byrd, B. Shariat, and A. Ben-Hur. *Protein Interface Prediction Using Graph Convolutional Networks.* In *Proceedings of NeurIPS*, 2017.

[6] Kipf, T. N., and M. Welling. *Semi-Supervised Classification with Graph Convolutional Networks.* In *Proceedings of ICLR*, 2017.

[7] Zitnik, M., M. Agrawal, and J. Leskovec. *Modeling Polypharmacy Side Effects with Graph Convolutional Networks. Bioinformatics*, vol. 34, no. 13, pp. i457–i466, 2018.

[8] Veličković, P., G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. *Graph Attention Networks.* In *Proceedings of ICLR*, 2018.

[9] Dwivedi, V. P., C. K. Joshi, and X. Bresson. *Benchmarking Graph Neural Network Interpretability.* In *Proceedings of NeurIPS*, 2022.

[10] Akiba, T., S. Sano, T. Yanase, T. Ohta, and M. Koyama. *Optuna: A Next-generation Hyperparameter Optimization Framework.* In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631, 2019.

[11] Siebenmorgen, T., F. Menezes, S. Benassou, E. Merdivan, K. Didi, A. Santos Dias Mourão, R. Kitel, P. Liò, S. Kesselheim, M. Piraud, F. J. Theis, M. Sattler, and G. M. Popowicz. *MISATO: Machine Learning Dataset of Protein–Ligand Complexes for Structure-Based Drug Discovery. Nature Computational Science*, vol. 4, no. 5, pp. 367–378, 2024.

[12] Pahng, A., et al. *Embedding Comparison Studies.* 2024. (Note: Please add full citation details if available)

[13] Kojaku, S., et al. *Embedding Standardization Practice.* 2024. (Note: Please add full citation details if available)

[14] Hou, Z., X. Liu, Y. Cen, Y. Dong, H. Yang, C. Wang, and J. Tang. *GraphMAE: Self-Supervised Masked Graph Autoencoders.* In *Proceedings of the 28th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 594–604, 2022.

[15] Li, J., R. Wu, W. Sun, L. Chen, S. Tian, L. Zhu, C. Meng, Z. Zheng, and W. Wang. *What's Behind the Mask: Understanding Masked Graph Modeling for Graph Autoencoders.* In *Proceedings of the 29th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1268–1279, 2023.

[16] Zhou, K., et al. *AutoML for Graph Neural Networks: A Survey.* Available at: `https://graph-neural-networks.github.io`, 2022. (Note: Please add full citation details if this is a published paper)

[17] Knyazev, B., G. W. Taylor, and M. Amer. *Understanding Attention and Generalization in Graph Neural Networks.* In *Proceedings of NeurIPS*, pp. 4202–4212, 2019.

[18] Prechelt, L. *Early Stopping — But When?* In *Neural Networks: Tricks of the Trade*, G. B. Orr and K.-R. Müller (Eds.), Lecture Notes in Computer Science, vol. 1524, pp. 55–69, Springer, 1998.