Mathematical Framework for Interpreting k-GNN Expressivity via GIN-Graph Explanations

A Reverse-Engineering Reference

Contents

1 Problem Setting

Problem 1 (Graph Classification)**.** *Given a dataset* $\mathcal{D} = \{(G_i, y_i)\}_{i=1}^{N}$ *where each* $G_i = (V_i, E_i, \mathbf{X}_i)$ *is an attributed graph with node features* $\mathbf{X}_i \in \mathbb{R}^{|V_i| \times d}$ *and* $y_i \in \{0, 1, \ldots, C-1\}$ *is a class label, learn a function* $f_\theta : \mathcal{G} \to \mathbb{R}^C$ *that maps graphs to class logits.*

Problem 2 (Model-Level Explanation)**.** *Given a trained classifier* f_θ*, generate a synthetic graph* $\tilde{G}$ *that* maximally activates *a target class* c*, while being* structurally realistic *with respect to* $\mathcal{D}$*. The generated graph serves as a prototypical explanation of what the model considers class* c*.*

The pipeline has three stages:

1. **Train** a k-GNN classifier f_θ on $\mathcal{D}$ (Section 2).
2. **Train** a GIN-Graph generator G_ϕ to produce explanation graphs guided by f_θ (Section 3).
3. **Evaluate** generated explanations via composite metrics (Section 4).

2 Stage 1: k-GNN Classification

2.1 The Weisfeiler–Leman Hierarchy

The k-dimensional Weisfeiler–Leman (k-WL) test is a hierarchy of graph isomorphism tests. Each level k defines a message-passing scheme over k-element subsets ("k-sets") of nodes.

Definition 1 (k-Set)**.** *For a graph* $G = (V, E)$ *with* $|V| = n$*, the set of* k*-element subsets is*

$$\binom{V}{k} = \{S \subseteq V : |S| = k\}.$$

The total number of k*-sets is* $\binom{n}{k} = \frac{n!}{k!(n-k)!}$*.*

Definition 2 (k-Set Neighbourhood (Morris et al.))**.** *Two* k*-sets* S, S' *are* neighbours*, written* $S \sim S'$*, if and only if they differ in exactly one element, and the removed and added elements are adjacent:*

$$S \sim S' \iff |S \triangle S'| = 2 \;\text{ and }\; \exists\, u \in S \setminus S',\ w \in S' \setminus S : (u, w) \in E.$$

Equivalently, $S' = (S \setminus \{u\}) \cup \{w\}$ *where* $u \in S$*,* $w \notin S$*, and* $(u, w) \in E$*.*

2.2 1-GNN (Standard Message Passing)

A 1-GNN operates on individual nodes $v \in V$. At each layer $t \in \{1, \ldots, T\}$:

$$\mathbf{h}_v^{(t)} = \sigma\left(\mathbf{h}_v^{(t-1)}\mathbf{W}_1^{(t)} + \sum_{u \in \mathcal{N}(v)} \mathbf{h}_u^{(t-1)}\mathbf{W}_2^{(t)}\right) \tag{1}$$

where $\mathbf{h}_v^{(0)} = \mathbf{x}_v \in \mathbb{R}^d$ (input features), $\mathcal{N}(v) = \{u : (u, v) \in E\}$, and $\sigma = \mathrm{ReLU}$. The weight matrices are:

$$\mathbf{W}_1^{(t)} \in \mathbb{R}^{d_{t-1} \times d_t}, \quad \mathbf{W}_2^{(t)} \in \mathbb{R}^{d_{t-1} \times d_t}, \quad \text{(no bias)} \tag{2}$$

Concrete dimensions. With hidden dimension d_h and $T = 3$ layers:

$$\text{Layer 1:} \quad \mathbf{W}_1^{(1)}, \mathbf{W}_2^{(1)} \in \mathbb{R}^{d \times d_h} \tag{3}$$

$$\text{Layer 2:} \quad \mathbf{W}_1^{(2)}, \mathbf{W}_2^{(2)} \in \mathbb{R}^{d_h \times d_h} \tag{4}$$

$$\text{Layer 3:} \quad \mathbf{W}_1^{(3)}, \mathbf{W}_2^{(3)} \in \mathbb{R}^{d_h \times d_h} \tag{5}$$

The `OneGNNLayer` implementation uses PyTorch Geometric's `MessagePassing` with `aggr='add'`. The `message` function applies $\mathbf{W}_2$ to source node features, and aggregation sums over neighbours.

Matrix form (dense). For a batch of graphs with shared node count n:

$$\mathbf{H}^{(t)} = \sigma\Big(\mathbf{H}^{(t-1)}\mathbf{W}_1^{(t)} + \mathbf{A}\,\mathbf{H}^{(t-1)}\mathbf{W}_2^{(t)}\Big), \quad \mathbf{H}^{(t)} \in \mathbb{R}^{n \times d_t} \tag{6}$$

where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the adjacency matrix. The matrix product $\mathbf{A}\,\mathbf{H}^{(t-1)}$ computes the neighbourhood aggregation: row v of the result is $\sum_{u \in \mathcal{N}(v)} \mathbf{h}_u^{(t-1)}$. This form is fully differentiable with respect to $\mathbf{A}$, which is critical for GIN-Graph training (Section 3.5).

Batched form. For a batch of B graphs with fixed node count n:

$$\mathbf{H}^{(t)} = \sigma\Big(\mathbf{H}^{(t-1)}\mathbf{W}_1^{(t)} + \texttt{bmm}(\mathbf{A},\, \mathbf{H}^{(t-1)}\mathbf{W}_2^{(t)})\Big), \quad \mathbf{H}^{(t)} \in \mathbb{R}^{B \times n \times d_t} \tag{7}$$

where `bmm` is batched matrix multiplication: $[\mathbf{A}]_b \cdot [\mathbf{H}^{(t-1)}\mathbf{W}_2^{(t)}]_b$ for each $b \in \{1, \ldots, B\}$.

Graph-level readout.

$$\mathbf{z}_G^{(1)} = \sum_{v \in V} \mathbf{h}_v^{(T)} \in \mathbb{R}^{d_h} \tag{8}$$

This is a sum pool (`global_add_pool`), not mean pool, which preserves graph-size information.

2.3 2-GNN (Pair Message Passing)

A 2-GNN operates on ordered pairs $\{u, v\} \in \binom{V}{2}$ with $u < v$ (canonical ordering).

Initial features. For each pair $\{u, v\}$ with $u < v$:

$$\mathbf{f}_{\{u,v\}}^{(0)} = \Big[\mathbf{h}_u^{(T)} \parallel \mathbf{h}_v^{(T)} \parallel \mathbf{A}_{uv}\Big] \in \mathbb{R}^{2d_h+1} \tag{9}$$

where $\parallel$ denotes concatenation, $\mathbf{h}_u^{(T)}, \mathbf{h}_v^{(T)}$ are the final 1-GNN node embeddings (so the 2-GNN receives preprocessed features), and $\mathbf{A}_{uv} \in \{0, 1\}$ is the *iso-type*: 1 if $(u, v) \in E$, 0 otherwise.

Remark 1 (Canonical ordering)**.** *The ordering convention places the node with the smaller index first:* $\mathbf{f}_{\{i,j\}} = [\mathbf{h}_{\min(i,j)} \| \mathbf{h}_{\max(i,j)} \| \mathbf{A}_{ij}]$*. This ensures each unordered pair* $\{i, j\}$ *has a unique canonical feature representation regardless of the order in which* i *and* j *are enumerated.*

Message passing. At each layer $t \in \{1, \ldots, T_2\}$ (with $T_2 = 2$ by default):

$$\mathbf{f}_S^{(t)} = \sigma\left(\mathbf{f}_S^{(t-1)}\mathbf{W}_1^{(t)} + \sum_{S' \sim S} \mathbf{f}_{S'}^{(t-1)}\mathbf{W}_2^{(t)}\right) \tag{10}$$

This uses `KSetLayer`, which implements the aggregation via `index_add_`:

1. Compute $\mathbf{g}_{S'} = \mathbf{f}_{S'}^{(t-1)}\mathbf{W}_2^{(t)}$ for all source k-sets S'.
2. For each edge $(S' \to S)$, accumulate $\mathbf{g}_{S'}$ into position S using `index_add_`.
3. Add the self-transform $\mathbf{f}_S^{(t-1)}\mathbf{W}_1^{(t)}$ and apply σ.

Concrete dimensions. With $T_2 = 2$ layers:

$$\text{Layer 1:} \quad \mathbf{W}_1^{(1)}, \mathbf{W}_2^{(1)} \in \mathbb{R}^{(2d_h+1)\times d_h} \tag{11}$$

$$\text{Layer 2:} \quad \mathbf{W}_1^{(2)}, \mathbf{W}_2^{(2)} \in \mathbb{R}^{d_h \times d_h} \tag{12}$$

Neighbourhood enumeration. For a pair $\{u, v\}$, the neighbours per the Morris et al. definition decompose into two cases:

- **Case 1: Replace u with w.** Target pair $\{v, w\}$ (with $w \neq u, v$ and $(u, w) \in E$). The adjacency condition ensures that the removed element u is adjacent to the newly added element w.
- **Case 2: Replace v with w.** Target pair $\{u, w\}$ (with $w \neq u, v$ and $(v, w) \in E$).

Dense formulation. Storing all pair features in a tensor $\mathbf{F} \in \mathbb{R}^{n\times n\times d}$ (where $\mathbf{F}_{ij}$ holds the features for pair $\{i, j\}$), the neighbour aggregation decomposes into:

$$\text{agg}_1[i, j] = \sum_w \mathbf{A}_{iw} \cdot \mathbf{G}[j, w] \qquad \text{(replace } i \text{ with } w\text{, need } (i, w) \in E) \tag{13}$$

$$\text{agg}_2[i, j] = \sum_w \mathbf{A}_{jw} \cdot \mathbf{G}[i, w] \qquad \text{(replace } j \text{ with } w\text{, need } (j, w) \in E) \tag{14}$$

where $\mathbf{G} = \mathbf{F}\mathbf{W}_2 \in \mathbb{R}^{n\times n\times d_h}$.

Proposition 1 (Equivalence to Morris et al.)**.** *The einsum formulation*

$$agg_1 = \texttt{einsum}(\texttt{'biw,bjwd->bijd'}, \mathbf{A}, \mathbf{G}), \quad agg_2 = \texttt{einsum}(\texttt{'bjw,biwd->bijd'}, \mathbf{A}, \mathbf{G}) \tag{15}$$

computes exactly the Morris et al. neighbourhood aggregation for 2-sets, where b indexes the batch, i and j index the pair elements, w indexes the summation over potential replacement nodes, and d indexes the feature dimension.

Proof. For agg_1: the (b, i, j, d) entry is $\sum_w \mathbf{A}_b[i, w] \cdot \mathbf{G}_b[j, w, d]$. Since $\mathbf{A}_b[i, w] = 1$ only when $(i, w) \in E$, and $\mathbf{G}_b[j, w, d]$ is the transformed feature of pair $\{j, w\}$, this sums over all neighbours $\{j, w\}$ of $\{i, j\}$ obtained by replacing i with some w adjacent to i. This matches Case 1 of the Morris definition. The argument for agg_2 is symmetric.

The two aggregations combined give the full neighbourhood sum. Note that self-loops are removed ($\mathbf{A}$ has zero diagonal) to prevent $w = i$ or $w = j$. □

Graph-level readout. Pool over unique pairs (upper triangle only, matching the canonical ordering $i < j$):

$$\mathbf{z}_G^{(2)} = \sum_{i<j} \mathbf{f}_{\{i,j\}}^{(T_2)} \in \mathbb{R}^{d_h} \tag{16}$$

In the dense formulation, this is implemented as:

$$\mathbf{z}_G^{(2)} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbf{F}_{ij}^{(T_2)} = \left(\mathbf{F}^{(T_2)} \odot \mathbf{M}_{\text{triu}}\right).\text{sum}(\text{dims} = (1, 2)) \tag{17}$$

where $\mathbf{M}_{\text{triu}}$ is the upper-triangular mask with ones above the diagonal.

2.4 3-GNN (Triplet Message Passing)

A 3-GNN operates on 3-element subsets $\{a, b, c\} \in \binom{V}{3}$ with $a < b < c$.

Initial features.

$$\mathbf{f}^{(0)}_{\{a,b,c\}} = \left[\mathbf{h}^{(T)}_a \parallel \mathbf{h}^{(T)}_b \parallel \mathbf{h}^{(T)}_c \parallel \mathrm{iso}(\{a,b,c\})\right] \in \mathbb{R}^{3d_h+4} \tag{18}$$

The iso-type encodes the number of edges among the three nodes as a one-hot vector in $\mathbb{R}^4$:

$$e = |\{(a,b),(b,c),(a,c)\} \cap E| \in \{0,1,2,3\} \tag{19}$$

$$\mathrm{iso}(\{a,b,c\}) = \mathbf{e}_e \in \{0,1\}^4 \tag{20}$$

where $\mathbf{e}_e$ is the standard basis vector with 1 at position e. The four iso-types correspond to: no edges (independent set), one edge (path endpoint), two edges (path or wedge), three edges (triangle).

Soft iso-type (for generator gradient flow). When $\mathbf{A}$ contains continuous values from the generator:

$$e_{\text{soft}} = \mathbf{A}_{ab} + \mathbf{A}_{bc} + \mathbf{A}_{ac} \in [0,3] \tag{21}$$

$$\mathrm{iso}_k = \max(0,\ 1 - |e_{\text{soft}} - k|)\,, \quad k \in \{0,1,2,3\} \tag{22}$$

This is a *triangular basis function* centered at each integer k. Properties:

- At integer values: $e_{\text{soft}} = m \implies \mathrm{iso}_m = 1$ and $\mathrm{iso}_k = 0$ for $|k-m| \geq 1$. Recovers exact one-hot encoding.
- Between integers: e.g., $e_{\text{soft}} = 1.3 \implies \mathrm{iso}_1 = 0.7,\ \mathrm{iso}_2 = 0.3$. Smoothly interpolates.
- Partition of unity: $\sum_{k=0}^{3} \mathrm{iso}_k = 1$ for $e_{\text{soft}} \in [0,3]$.
- Differentiable: $\partial\,\mathrm{iso}_k / \partial\,\mathbf{A}_{ab}$ exists almost everywhere (piecewise linear).

Concrete dimensions. With $T_3 = 2$ layers:

$$\text{Layer 1:} \quad \mathbf{W}^{(1)}_1, \mathbf{W}^{(1)}_2 \in \mathbb{R}^{(3d_h+4)\times d_h} \tag{23}$$

$$\text{Layer 2:} \quad \mathbf{W}^{(2)}_1, \mathbf{W}^{(2)}_2 \in \mathbb{R}^{d_h \times d_h} \tag{24}$$

Neighbourhood enumeration. For a triplet $\{a,b,c\}$, three replacement cases:

- **Case 1: Replace c with d.** Need $(c,d) \in E$, target $\{a,b,d\}$.
- **Case 2: Replace b with d.** Need $(b,d) \in E$, target $\{a,c,d\}$.
- **Case 3: Replace a with d.** Need $(a,d) \in E$, target $\{b,c,d\}$.

Each target triplet is sorted to canonical form (min < mid < max) for lookup.

Message passing. Same form as Equation (10), but over 3-set neighbours using `KSetLayer` with the same `index_add_` aggregation.

Scalability. Full enumeration of 3-sets requires $\binom{n}{3}$ elements. For a graph with $n = 620$ nodes (PROTEINS), $\binom{620}{3} \approx 39.5$M triplets, requiring $\sim$30 GB of feature memory at $d_h = 64$. The implementation therefore samples up to M triplets randomly:

$$\mathcal{T} \subseteq \binom{V}{3}, \quad |\mathcal{T}| \leq M = 3000$$

Similarly, 2-sets are sampled with $M_2 = 5000$ for large graphs. Canonical encoding and binary search (`searchsorted`) are used for efficient k-set lookup, replacing dense $\mathcal{O}(n^k)$ lookup tables with $\mathcal{O}(M \log M)$ operations.

2.5 Hierarchical Models

The hierarchical models stack k-GNN levels, using lower-level embeddings as input to higher levels.

1-2-GNN (`Hierarchical12GNN`).

1. Run $T_1 = 3$ layers of 1-GNN (Eq. 1) to get node embeddings $\mathbf{h}_v^{(T_1)} \in \mathbb{R}^{d_h}$.
2. Compute sum-pooled 1-GNN embedding: $\mathbf{z}_G^{(1)} = \sum_v \mathbf{h}_v^{(T_1)} \in \mathbb{R}^{d_h}$.
3. Build pair features using Eq. (9) from the 1-GNN embeddings.
4. Run $T_2 = 2$ layers of 2-GNN (Eq. 10) on the pair features.
5. Compute sum-pooled 2-GNN embedding: $\mathbf{z}_G^{(2)} \in \mathbb{R}^{d_h}$.
6. Concatenate and classify:

$$\hat{\mathbf{y}} = \text{Classifier}_{12}\Big(\Big[\mathbf{z}_G^{(1)} \parallel \mathbf{z}_G^{(2)}\Big]\Big), \quad \Big[\mathbf{z}_G^{(1)} \parallel \mathbf{z}_G^{(2)}\Big] \in \mathbb{R}^{2d_h} \tag{25}$$

1-2-3-GNN (`Hierarchical123GNN`). Same as above, adding a 3-GNN branch:

1. Run 1-GNN ($T_1 = 3$ layers) $\to \mathbf{z}_G^{(1)} \in \mathbb{R}^{d_h}$.
2. Run 2-GNN ($T_2 = 2$ layers) using 1-GNN embeddings $\to \mathbf{z}_G^{(2)} \in \mathbb{R}^{d_h}$.
3. Build triplet features using Eq. (18) from the 1-GNN embeddings.
4. Run 3-GNN ($T_3 = 2$ layers) on the triplet features $\to \mathbf{z}_G^{(3)} \in \mathbb{R}^{d_h}$.
5. Concatenate and classify:

$$\hat{\mathbf{y}} = \text{Classifier}_{123}\Big(\Big[\mathbf{z}_G^{(1)} \parallel \mathbf{z}_G^{(2)} \parallel \mathbf{z}_G^{(3)}\Big]\Big), \quad \Big[\mathbf{z}_G^{(1)} \parallel \mathbf{z}_G^{(2)} \parallel \mathbf{z}_G^{(3)}\Big] \in \mathbb{R}^{3d_h} \tag{26}$$

Classifier MLP. All models use a 2-layer MLP with dropout:

$$\text{Classifier}(\mathbf{z}) = \mathbf{W}_{\text{out}}\ \text{ReLU}(\mathbf{W}_{\text{hid}}\,\mathbf{z} + \mathbf{b}_{\text{hid}}) + \mathbf{b}_{\text{out}} \tag{27}$$

with dropout ($p = 0.5$) applied after the ReLU. The weight dimensions depend on the model:

Model	$\mathbf{W}_{\text{hid}}$	Dropout	$\mathbf{W}_{\text{out}}$
1-GNN	$\mathbb{R}^{d_h \times d_h}$	$p = 0.5$	$\mathbb{R}^{d_h \times C}$
1-2-GNN	$\mathbb{R}^{2d_h \times d_h}$	$p = 0.5$	$\mathbb{R}^{d_h \times C}$
1-2-3-GNN	$\mathbb{R}^{3d_h \times d_h}$	$p = 0.5$	$\mathbb{R}^{d_h \times C}$

2.6 k-GNN Training Procedure

Loss function. Standard cross-entropy over graph labels:

$$\mathcal{L}_{\text{CE}}(\theta) = -\frac{1}{|\mathcal{B}|} \sum_{(G,y)\in\mathcal{B}} \log \frac{\exp(f_\theta(G)_y)}{\sum_{j=0}^{C-1} \exp(f_\theta(G)_j)} \tag{28}$$

where $\mathcal{B}$ is a mini-batch and $f_\theta(G)_y$ is the logit for the true class.

Optimizer. Adam with learning rate $\eta = 0.01$ and default momentum parameters $(\beta_1, \beta_2) = (0.9, 0.999)$:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{29}$$

where $\hat{m}_t$ and $\hat{v}_t$ are bias-corrected first and second moment estimates of the gradient.

Training protocol.

- Dataset split: 80% train, 20% test (fixed random seed 42).
- Batch size: 32 graphs per mini-batch.
- Epochs: 100 (default).
- Model selection: track best test accuracy across epochs; restore best model weights at the end.
- No learning rate scheduling, no weight decay, no data augmentation.

Parameter counts. For MUTAG ($d = 7$, $d_h = 64$, $C = 2$):

Model	1-GNN params	2-GNN params	3-GNN params	Classifier
1-GNN	$2d \cdot d_h + 4d_h^2$ $= 17{,}280$	—	—	$d_h^2 + d_h C$ $= 4{,}224$
1-2-GNN	17,280	$2(2d_h + 1)d_h + 2d_h^2$ $= 24{,}704$	—	$2d_h^2 + d_h C$ $= 8{,}320$
1-2-3-GNN	17,280	24,704	$2(3d_h + 4)d_h + 2d_h^2$ $= 33{,}280$	$3d_h^2 + d_h C$ $= 12{,}416$

(All counts include both $\mathbf{W}_1$ and $\mathbf{W}_2$ per layer; classifier includes biases.)

3 Stage 2: GIN-Graph Explanation Generator

3.1 Architecture Overview

The GIN-Graph system is a conditional GAN that generates explanation graphs. It consists of:

- A **generator** $G_\phi : \mathbb{R}^{d_z} \to (\tilde{\mathbf{A}}, \tilde{\mathbf{X}})$ that maps noise to graphs.
- A **discriminator** $D_\psi : (\mathbf{A}, \mathbf{X}) \to \mathbb{R}$ that scores graph realism.
- The **pretrained k-GNN** f_θ (frozen) that provides classification guidance.

3.2 Generator Architecture

3.2.1 Backbone MLP

A noise vector $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{d_z})$ with $d_z = 32$ is transformed by a 2-layer MLP:

$$\mathbf{a}_1 = \text{LeakyReLU}_{0.2}\Big(\mathbf{W}_1^{\text{bb}}\,\mathbf{z} + \mathbf{b}_1^{\text{bb}}\Big) \in \mathbb{R}^{d_g} \tag{30}$$

$$\mathbf{a}_2 = \text{Dropout}_{p_g}\Big(\text{LeakyReLU}_{0.2}\Big(\mathbf{W}_2^{\text{bb}}\,\mathbf{a}_1 + \mathbf{b}_2^{\text{bb}}\Big)\Big) \in \mathbb{R}^{2d_g} \tag{31}$$

where d_g is the generator hidden dimension (default $d_g = 128$), $p_g = 0$ (dropout disabled by default), and:

$$\text{LeakyReLU}_\alpha(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases}, \quad \alpha = 0.2 \tag{32}$$

The weight dimensions are:

$$\mathbf{W}_1^{\text{bb}} \in \mathbb{R}^{d_g \times d_z}, \quad \mathbf{b}_1^{\text{bb}} \in \mathbb{R}^{d_g} \qquad (d_z = 32 \to d_g = 128) \tag{33}$$

$$\mathbf{W}_2^{\text{bb}} \in \mathbb{R}^{2d_g \times d_g}, \quad \mathbf{b}_2^{\text{bb}} \in \mathbb{R}^{2d_g} \qquad (d_g = 128 \to 2d_g = 256) \tag{34}$$

3.2.2 Adjacency Head

Raw logits are projected from the backbone output, reshaped to $[n, n]$, and symmetrised:

$$\mathbf{L}_{\text{raw}} = \text{reshape}(\mathbf{W}_{\text{adj}}\, \mathbf{a}_2 + \mathbf{b}_{\text{adj}},\ [n, n]), \quad \mathbf{W}_{\text{adj}} \in \mathbb{R}^{n^2 \times 2d_g} \tag{35}$$

$$\mathbf{L} = \frac{\mathbf{L}_{\text{raw}} + \mathbf{L}_{\text{raw}}^\top}{2} \tag{36}$$

This symmetrisation of the *logits* (before Gumbel-Softmax) ensures both (i, j) and (j, i) receive the same input probability. For MUTAG ($n = 28$): $\mathbf{W}_{\text{adj}} \in \mathbb{R}^{784 \times 256}$ ($\sim$201K parameters).

3.2.3 Gumbel-Softmax Reparameterisation

To make discrete edge sampling differentiable, we use the Gumbel-Softmax trick with a straight-through estimator.

The Gumbel-Max trick. To sample from a categorical distribution with logits $\ell_1, \ldots, \ell_K$:

$$\text{sample} = \arg\max_k \left(\ell_k + g_k\right), \quad g_k \sim \text{Gumbel}(0, 1) \tag{37}$$

where $g_k = -\log(-\log(u_k))$, $u_k \sim \text{Uniform}(0, 1)$. This is exact but non-differentiable due to $\arg\max$.

Gumbel-Softmax relaxation (Jang et al., 2017; Maddison et al., 2017). Replace $\arg\max$ with softmax at temperature τ:

$$y_k = \frac{\exp((\ell_k + g_k)/\tau)}{\sum_{j=1}^{K} \exp((\ell_j + g_j)/\tau)}, \quad k = 1, \ldots, K \tag{38}$$

As $\tau \to 0$, $\mathbf{y}$ approaches a one-hot vector (recovering the discrete sample). As $\tau \to \infty$, $\mathbf{y}$ approaches a uniform distribution.

Straight-through estimator (`hard=True`). For edge sampling, we need discrete $\{0, 1\}$ values in the forward pass but continuous gradients in the backward pass:

$$\textbf{Forward:} \quad \hat{y}_k = \begin{cases} 1 & \text{if } k = \arg\max_j\ y_j \\ 0 & \text{otherwise} \end{cases} \tag{39}$$

$$\textbf{Backward:} \quad \frac{\partial \hat{y}_k}{\partial \ell_m} \approx \frac{\partial y_k}{\partial \ell_m} = \frac{\partial}{\partial \ell_m}\left[\frac{\exp((\ell_k + g_k)/\tau)}{\sum_j \exp((\ell_j + g_j)/\tau)}\right] \tag{40}$$

Implemented as: $\hat{\mathbf{y}} = \text{one\_hot}(\arg\max(\mathbf{y})) - \text{sg}(\mathbf{y}) + \mathbf{y}$, where sg is stop-gradient. The gradient of this expression is $\partial\hat{\mathbf{y}}/\partial\boldsymbol{\ell} = \partial\mathbf{y}/\partial\boldsymbol{\ell}$ since sg blocks the gradient of the first two terms, and only $\mathbf{y}$ contributes.

Application to edge sampling. For each entry (i, j) of the adjacency matrix, we construct a 2-class logit vector:

$$\boldsymbol{\ell}_{ij} = [\mathbf{L}_{ij},\ -\mathbf{L}_{ij}] \in \mathbb{R}^2 \tag{41}$$

and apply Gumbel-Softmax:

$$\tilde{\mathbf{A}}_{ij} = \text{GumbelSoftmax}(\boldsymbol{\ell}_{ij},\ \tau)_0 \tag{42}$$

where subscript 0 selects the "edge present" class. The probability of an edge (before Gumbel noise) is:

$$P(\text{edge at } (i,j)) = \sigma(2\mathbf{L}_{ij}) = \frac{1}{1+\exp(-2\mathbf{L}_{ij})} \tag{43}$$

since the logit difference is $\mathbf{L}_{ij} - (-\mathbf{L}_{ij}) = 2\mathbf{L}_{ij}$.

Temperature during training vs. evaluation.

- **Training** ($\tau = 1.0$): Higher temperature allows more exploration. The continuous gradients have moderate variance.
- **Evaluation** ($\tau = 0.1$): Lower temperature produces sharper, near-discrete outputs. The generator exploits learned logit patterns.

3.2.4 Post-Gumbel Symmetrisation (Critical)

Since Gumbel noise g_k is sampled independently for each (i, j) and (j, i) entry, we get $\tilde{\mathbf{A}}_{ij} \neq \tilde{\mathbf{A}}_{ji}$ even though the logits were symmetrised. We enforce symmetry using only the upper triangle:

$$\mathbf{U} = \text{triu}(\tilde{\mathbf{A}}, \text{diagonal} = 1), \quad \tilde{\mathbf{A}} \leftarrow \mathbf{U} + \mathbf{U}^\top \tag{44}$$

Remark 2 (Why not average?)**.** *The naive approach* $\tilde{\mathbf{A}} \leftarrow (\tilde{\mathbf{A}} + \tilde{\mathbf{A}}^\top)/2$ *creates entries equal to* 0.5 *when exactly one of* (i, j) *or* (j, i) *was sampled as an edge by Gumbel-Softmax with* `hard=True`*. During evaluation, edges are thresholded at* > 0.5*, so all* 0.5*-valued entries are* discarded*. This creates a systematic train–eval mismatch:*

- ***During training****: the degree loss sees continuous values with average degree* $\approx \mu_c$ *(correct).*
- ***During evaluation****: thresholding drops* $\sim$*50% of edges (those that were* 0.5*), reducing average degree to* $\approx \mu_c/2$ *and making degree score* $d \approx \exp(-(\mu_c/2)^2/(2\sigma_c^2)) \approx 0$*.*

On MUTAG ($\sigma_c \approx 0.07$*), this resulted in 0% valid explanations. The upper-triangle approach in Eq.* (44) *ensures each edge is decided by exactly one Gumbel sample, producing strictly* $\{0, 1\}$ *values.*

3.2.5 Node Feature Head

Raw logits for node types are projected and sampled via categorical Gumbel-Softmax:

$$\tilde{\mathbf{X}}_{\text{raw}} = \text{reshape}(\mathbf{W}_{\text{feat}}\, \mathbf{a}_2 + \mathbf{b}_{\text{feat}},\ [n, d]), \quad \mathbf{W}_{\text{feat}} \in \mathbb{R}^{nd \times 2d_g} \tag{45}$$

$$\tilde{\mathbf{X}} = \text{GumbelSoftmax}\Big(\tilde{\mathbf{X}}_{\text{raw}},\ \tau,\ \text{dim} = -1\Big) \in \{0, 1\}^{n \times d} \tag{46}$$

Each row is a one-hot vector selecting a node type from d categories. For MUTAG: $d = 7$ atom types, $\mathbf{W}_{\text{feat}} \in \mathbb{R}^{196 \times 256}$.

3.2.6 Generator Parameter Summary

Component	Dimensions	Parameters
Backbone layer 1	$\mathbb{R}^{d_z} \to \mathbb{R}^{d_g}$	$d_z \cdot d_g + d_g$
Backbone layer 2	$\mathbb{R}^{d_g} \to \mathbb{R}^{2d_g}$	$d_g \cdot 2d_g + 2d_g$
Adjacency head	$\mathbb{R}^{2d_g} \to \mathbb{R}^{n^2}$	$2d_g \cdot n^2 + n^2$
Feature head	$\mathbb{R}^{2d_g} \to \mathbb{R}^{nd}$	$2d_g \cdot nd + nd$

For MUTAG ($d_z = 32$, $d_g = 128$, $n = 28$, $d = 7$): backbone = 37,248, adj head = 201,488, feat head = 50,372. Total $\approx$ 289K parameters.

3.3 Discriminator Architecture

3.3.1 Dense GCN Layers

The discriminator uses a 2-layer dense GCN. Each `DenseGCNLayer` implements:

$$\mathbf{H}^{(l+1)} = \sigma\Big(\hat{\mathbf{A}}\,\mathbf{H}^{(l)}\,\mathbf{W}^{(l)}\Big) \tag{47}$$

where the normalised adjacency with self-loops is:

$$\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1}\,\tilde{\mathbf{A}}, \quad \tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_n, \quad \tilde{\mathbf{D}} = \mathrm{diag}\Big(\tilde{\mathbf{A}}\,\mathbf{1}\Big) \tag{48}$$

This is a *row-normalised* adjacency (each row sums to 1), equivalent to:

$$\hat{\mathbf{A}}_{ij} = \frac{\tilde{\mathbf{A}}_{ij}}{\sum_k \tilde{\mathbf{A}}_{ik}} = \frac{\mathbf{A}_{ij} + \delta_{ij}}{\deg(i) + 1} \tag{49}$$

The message-passing interpretation: each node's representation is the weighted average of its neighbours' transformed features (including itself via the self-loop).

Implementation detail. The degree is clamped to minimum 1 to avoid division by zero for isolated nodes:

$$\hat{\mathbf{A}}_{ij} = \frac{\tilde{\mathbf{A}}_{ij}}{\max\Big(1,\, \sum_k \tilde{\mathbf{A}}_{ik}\Big)} \tag{50}$$

Concrete dimensions. With $\sigma = \mathrm{ReLU}$ and hidden dimension d_g:

$$\text{Layer 1:} \quad \mathbf{W}^{(1)} \in \mathbb{R}^{d \times d_g}, \quad \mathbf{b}^{(1)} \in \mathbb{R}^{d_g} \quad (d \to d_g) \tag{51}$$

$$\text{Layer 2:} \quad \mathbf{W}^{(2)} \in \mathbb{R}^{d_g \times d_g}, \quad \mathbf{b}^{(2)} \in \mathbb{R}^{d_g} \quad (d_g \to d_g) \tag{52}$$

3.3.2 Graph-Level Output

After the GCN layers, a graph-level embedding is computed via mean pooling, followed by an MLP:

$$\bar{\mathbf{h}} = \frac{1}{n}\sum_{v=1}^{n} \mathbf{h}_v^{(2)} \in \mathbb{R}^{d_g} \tag{53}$$

$$D_\psi(\mathbf{X}, \mathbf{A}) = \mathbf{W}_3^{\mathrm{D}}\ \mathrm{LeakyReLU}_{0.2}\big(\mathbf{W}_2^{\mathrm{D}}\,\bar{\mathbf{h}} + \mathbf{b}_2^{\mathrm{D}}\big) + \mathbf{b}_3^{\mathrm{D}} \in \mathbb{R} \tag{54}$$

with $\mathbf{W}_2^{\mathrm{D}} \in \mathbb{R}^{d_g \times d_g}$ and $\mathbf{W}_3^{\mathrm{D}} \in \mathbb{R}^{1 \times d_g}$.

Remark 3 (No sigmoid)**.** *The discriminator outputs a raw score (not a probability), consistent with the WGAN framework where D_ψ approximates the Wasserstein distance rather than a classifier.*

3.4 WGAN-GP Training

3.4.1 Discriminator Loss

The discriminator maximises the Wasserstein distance between real and fake distributions, subject to a gradient penalty:

$$\mathcal{L}_D = \underbrace{\mathbb{E}_{\tilde{G}\sim G_\phi}[D_\psi(\tilde{G})]}_{\text{fake score (push down)}} - \underbrace{\mathbb{E}_{G\sim\mathcal{D}_c}[D_\psi(G)]}_{\text{real score (push up)}} + \underbrace{\lambda_{\text{GP}}\cdot\mathcal{L}_{\text{GP}}}_{\text{gradient penalty}} \tag{55}$$

where $\mathcal{D}_c$ is the subset of training graphs with label c.

3.4.2 Gradient Penalty

For interpolated samples between real and fake:

$$\epsilon \sim \text{Uniform}(0,1) \tag{56}$$

$$\hat{\mathbf{X}} = \epsilon\,\mathbf{X}_{\text{real}} + (1-\epsilon)\,\tilde{\mathbf{X}} \tag{57}$$

$$\hat{\mathbf{A}} = \epsilon\,\mathbf{A}_{\text{real}} + (1-\epsilon)\,\tilde{\mathbf{A}} \tag{58}$$

The gradient penalty enforces the 1-Lipschitz constraint:

$$\mathcal{L}_{\text{GP}} = \mathbb{E}_\epsilon\left[\left(\left\|\nabla_{(\hat{\mathbf{X}},\hat{\mathbf{A}})} D_\psi(\hat{\mathbf{X}},\hat{\mathbf{A}})\right\|_2 - 1\right)^2\right] \tag{59}$$

The gradient is computed jointly over both inputs by flattening $\nabla_{\hat{\mathbf{X}}} \in \mathbb{R}^{nD}$ and $\nabla_{\hat{\mathbf{A}}} \in \mathbb{R}^{n^2}$, then computing the L2 norm of the concatenated vector:

$$\left\|\nabla_{(\hat{\mathbf{X}},\hat{\mathbf{A}})} D_\psi\right\|_2 = \sqrt{\|\nabla_{\hat{\mathbf{X}}} D_\psi\|_2^2 + \|\nabla_{\hat{\mathbf{A}}} D_\psi\|_2^2} \tag{60}$$

Default: $\lambda_{\text{GP}} = 10$.

3.4.3 Real Data Preparation

Real graphs from $\mathcal{D}_c$ are converted to dense format $(\mathbf{X}_{\text{real}}, \mathbf{A}_{\text{real}}) \in \mathbb{R}^{n\times d} \times \mathbb{R}^{n\times n}$ by:

- **Graphs with $< n$ nodes**: zero-pad features and adjacency to size n.
- **Graphs with $> n$ nodes**: truncate to first n nodes, filter edges to only include nodes $< n$.
- **Graphs with exactly n nodes**: use directly.

3.4.4 Discriminator Training Step

Algorithm 1 Single Discriminator Step

Require: Real batch $(\mathbf{X}_r, \mathbf{A}_r)$, batch size B

1: Sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{d_z})^B$
2: $(\tilde{\mathbf{A}}, \tilde{\mathbf{X}}) \leftarrow G_\phi(\mathbf{z}, \tau)$ ▷ Generate fake graphs
3: $s_r \leftarrow D_\psi(\mathbf{X}_r, \mathbf{A}_r)$ ▷ Score real
4: $s_f \leftarrow D_\psi(\text{sg}(\tilde{\mathbf{X}}), \text{sg}(\tilde{\mathbf{A}}))$ ▷ Score fake (detached)
5: Compute $\mathcal{L}_{\text{GP}}$ via Eq. (59)
6: $\mathcal{L}_D \leftarrow \text{mean}(s_f) - \text{mean}(s_r) + \lambda_{\text{GP}} \cdot \mathcal{L}_{\text{GP}}$
7: Update ψ via Adam to minimise $\mathcal{L}_D$

The discriminator is trained $n_{\text{critic}} = 1$ times per generator update (reduced from the standard WGAN-GP $n_{\text{critic}} = 5$ since the GNN guidance loss also stabilises training).

3.5 Dense Forward Pass Through Pretrained k-GNN

The generator produces dense matrices $(\tilde{\mathbf{A}}, \tilde{\mathbf{X}}) \in \mathbb{R}^{B\times n\times n} \times \mathbb{R}^{B\times n\times d}$, but the pretrained k-GNN was trained on sparse PyTorch Geometric graph data. A differentiable *wrapper* (`DenseToSparseWrapper`) converts between these representations while preserving gradient flow through $\tilde{\mathbf{A}}$.

3.5.1 1-GNN Wrapper (Fully Differentiable)

Direct application of the batched dense form (Eq. 6) using the pretrained `OneGNNLayer` weights:

$$\mathbf{H}^{(t)} = \texttt{layer.activation}\Big(\texttt{layer.W1}(\mathbf{H}^{(t-1)}) + \texttt{bmm}(\tilde{\mathbf{A}}, \texttt{layer.W2}(\mathbf{H}^{(t-1)}))\Big) \tag{61}$$

Graph embedding: $\mathbf{z}_G^{(1)} = \sum_v \mathbf{h}_v^{(T)}$ (sum over node dimension).

Gradient flow: $\tilde{\mathbf{A}}$ appears explicitly in the matrix multiplication, so $\partial\mathcal{L}/\partial\tilde{\mathbf{A}}_{ij}$ propagates through all T layers. Each layer contributes:

$$\frac{\partial \mathbf{H}^{(t)}}{\partial \tilde{\mathbf{A}}_{ij}} = \sigma'(\cdot) \cdot \mathbf{e}_i \, (\mathbf{H}^{(t-1)}\mathbf{W}_2^{(t)})_j^\top \tag{62}$$

where $\mathbf{e}_i$ is the i-th standard basis vector and σ' is the ReLU derivative.

3.5.2 2-GNN Wrapper (Fully Differentiable)

Step 1: Build canonical pair features. For each (i, j) with $i < j$ (upper triangle) and $i > j$ (lower triangle), construct features with consistent canonical ordering:

$$\mathbf{F}_{ij} = \begin{cases} [\mathbf{h}_i \| \mathbf{h}_j \| \tilde{\mathbf{A}}_{ij}] & \text{if } i < j \\ [\mathbf{h}_j \| \mathbf{h}_i \| \tilde{\mathbf{A}}_{ij}] & \text{if } i > j \end{cases} \tag{63}$$

In compact form using upper/lower masks:

$$\text{first} = \mathbf{h}_i \cdot \mathbb{1}[i < j] + \mathbf{h}_j \cdot \mathbb{1}[i > j] \tag{64}$$

$$\text{second} = \mathbf{h}_j \cdot \mathbb{1}[i < j] + \mathbf{h}_i \cdot \mathbb{1}[i > j] \tag{65}$$

$$\mathbf{F}_{ij} = [\text{first} \| \text{second} \| \tilde{\mathbf{A}}_{ij}] \in \mathbb{R}^{2d_h+1} \tag{66}$$

The iso-type $\tilde{\mathbf{A}}_{ij}$ is continuous (not thresholded), preserving gradient flow.

Step 2: Apply pretrained 2-GNN layers. Self-loops are removed from $\tilde{\mathbf{A}}$ to prevent a pair from being its own neighbour:

$$\mathbf{A}_{\text{clean}} = \tilde{\mathbf{A}} \odot (\mathbf{1} - \mathbf{I}_n) \tag{67}$$

Then the einsum aggregation (Eq. 15) is applied with $\mathbf{A}_{\text{clean}}$.

Step 3: Pool upper triangle.

$$\mathbf{z}_G^{(2)} = \Big(\mathbf{F}^{(T_2)} \odot \mathbf{M}_{\text{triu}}\Big).\text{sum}(\text{dims} = (1, 2)) \tag{68}$$

Gradient flow: $\tilde{\mathbf{A}}$ appears in three places: (a) the iso-type feature, (b) the einsum aggregation weights, and (c) the pair-to-pair connectivity. All three paths are differentiable.

3.5.3 3-GNN Wrapper (Partial Gradient Flow)

Full dense 3-GNN would require $\mathbf{F} \in \mathbb{R}^{B\times n\times n\times n\times d}$, which is $\mathcal{O}(Bn^3d)$ in memory. Instead, a hybrid approach is used:

1. **Structure** (triplet selection, edge construction): computed with `torch.no_grad()` using binarised $\tilde{\mathbf{A}}$ (threshold at 0.5). **No gradient.**
2. **Node features**: $\mathbf{h}_a, \mathbf{h}_b, \mathbf{h}_c$ from the dense 1-GNN path. **Gradient flows.**
3. **Iso-types**: soft iso-types from continuous $\tilde{\mathbf{A}}$ via Eq. (22). **Gradient flows.**
4. **Message passing**: runs through pretrained `KSetLayer` weights using `index_add_` on the sparse triplet graph. **Gradient flows through features.**

This gives *partial* gradient flow: the generator can optimise the soft iso-types (edge probabilities within triplets) and node features, but cannot change which triplets exist or how they connect. In practice, the 1-GNN and 2-GNN components dominate the gradient signal.

3.6 Training Objective

3.6.1 Generator Loss

The generator loss combines three terms with dynamic weighting:

$$\mathcal{L}_G = (1-\lambda_t)\,\mathcal{L}_{\text{GAN}} + \lambda_t\,\mathcal{L}_{\text{GNN}} + \mathcal{L}_{\text{degree}} \tag{69}$$

GAN loss. Encourages realistic graphs:

$$\mathcal{L}_{\text{GAN}} = -\frac{1}{B}\sum_{b=1}^{B} D_\psi(G_\phi(\mathbf{z}_b)) \tag{70}$$

Minimising this pushes the discriminator score of fake graphs *up*, making them score closer to real graphs.

GNN guidance loss. Encourages target-class classification through the frozen pretrained k-GNN:

$$\mathcal{L}_{\text{GNN}} = \frac{1}{B}\sum_{b=1}^{B} \text{CE}\Big(f_\theta(\tilde{\mathbf{X}}_b, \tilde{\mathbf{A}}_b),\ c\Big) = -\frac{1}{B}\sum_{b=1}^{B} \log \frac{\exp(f_\theta(\tilde{G}_b)_c)}{\sum_{j=0}^{C-1}\exp(f_\theta(\tilde{G}_b)_j)} \tag{71}$$

The gradient $\partial\mathcal{L}_{\text{GNN}}/\partial\phi$ flows through: the dense wrapper (Section 3.5) $\rightarrow$ the Gumbel-Softmax (Section 3.2.3) $\rightarrow$ the generator backbone.

Degree regularisation loss. Penalises deviation from class-specific degree statistics:

$$\mathcal{L}_{\text{degree}} = \lambda_d \cdot \frac{1}{B}\sum_{b=1}^{B}\left(\frac{\bar{d}_b - \mu_c}{\sigma_c}\right)^2 \tag{72}$$

where:

- $\bar{d}_b = 2|\tilde{E}_b|/|\tilde{V}_b|$ is the average degree of generated graph b.
- $|\tilde{E}_b| = \sum_{i<j}\tilde{\mathbf{A}}^{(b)}_{ij}$ (continuous edge count during training).
- $|\tilde{V}_b| = \sum_i \mathbb{1}[\sum_j \tilde{\mathbf{A}}^{(b)}_{ij} > 0.5]$ (active node count, clamped ≥ 1).

- μ_c, σ_c are the mean and standard deviation of average degrees in class c.
- $\lambda_d = 1.0$ is the base weight (default).

Remark 4 (Auto-scaling property)**.** *The normalisation by σ_c makes the loss scale-invariant across datasets:*

$$\frac{\partial \mathcal{L}_{degree}}{\partial \bar{d}_b} = \frac{2\lambda_d}{B} \cdot \frac{\bar{d}_b - \mu_c}{\sigma_c^2} \tag{73}$$

For MUTAG ($\sigma_c \approx 0.07$): gradient magnitude $\propto 1/0.07^2 \approx 204$. A deviation of 0.1 produces gradient ≈ 29.
For PROTEINS ($\sigma_c \approx 0.42$): gradient magnitude $\propto 1/0.42^2 \approx 5.7$. A deviation of 0.1 produces gradient ≈ 1.4.
This means the loss automatically enforces tighter degree control on datasets with narrower degree distributions.

Remark 5 (Gradient through $\bar{d}_b$)**.** *Since $\bar{d}_b = 2\sum_{i<j} \tilde{\mathbf{A}}_{ij}/|\tilde{V}_b|$ and $\tilde{\mathbf{A}}_{ij}$ are continuous during training (Gumbel-Softmax), the gradient flows:*

$$\frac{\partial \bar{d}_b}{\partial \tilde{\mathbf{A}}_{ij}} = \frac{2}{|\tilde{V}_b|} \quad \textit{for } i < j \tag{74}$$

(assuming $|\tilde{V}_b|$ is treated as constant, which the implementation does by computing it without gradient).

3.6.2 Dynamic Weighting Schedule

The weight λ_t controls the balance between GAN and GNN losses. It follows a sigmoid schedule:

Schedule function. Given total iterations T, transition point $p \in [0, 1]$, steepness $k > 0$:

$$\lambda_t = \lambda_{\min} + (\lambda_{\max} - \lambda_{\min}) \cdot \sigma\left(k \cdot \left(\frac{2(t/T - p)}{1 - p} - 1\right)\right) \tag{75}$$

where $\sigma(x) = 1/(1 + e^{-x})$ is the sigmoid function. Default parameters: $\lambda_{\min} = 0$, $\lambda_{\max} = 1$, $p = 0.4$, $k = 10$.

Normalised progress. The argument to the sigmoid maps the training progress to $[-k, k]$:

$$\xi(t) = k \cdot \left(\frac{2(t/T - p)}{1 - p} - 1\right) \tag{76}$$

Key values:

- At $t = 0$: $\xi = k \cdot (2(-p)/(1-p) - 1) = k \cdot (-(2p + 1 - p)/(1 - p)) = -k \cdot (1 + p)/(1 - p)$. For $p = 0.4$: $\xi = -10 \cdot 1.4/0.6 \approx -23.3$, so $\sigma(\xi) \approx 0$ and $\lambda_0 \approx 0$.
- At $t = pT$ (transition point): $\xi = k \cdot (0 - 1) = -k = -10$, so $\sigma(-10) \approx 0.000045$ and $\lambda \approx 0$.
- At $t = (1 + p)T/2$ (midpoint of active phase): $\xi = 0$, so $\sigma(0) = 0.5$ and $\lambda = 0.5$.
- At $t = T$: $\xi = k \cdot (2/(1-p) - 2/(1-p) + 2p/(1-p)/(1-p) \cdot \ldots)$. For $p = 0.4$: $\xi = 10 \cdot 1 = 10$, so $\sigma(10) \approx 1$ and $\lambda_T \approx 1$.

Derivative of λ_t.

$$\frac{d\lambda_t}{dt} = (\lambda_{\max} - \lambda_{\min}) \cdot \sigma(\xi)\,(1 - \sigma(\xi)) \cdot \frac{2k}{T(1-p)} \tag{77}$$

The maximum rate of change occurs at $\xi = 0$ (where $\sigma(\xi)(1-\sigma(\xi)) = 1/4$):

$$\left.\frac{d\lambda_t}{dt}\right|_{\max} = \frac{k(\lambda_{\max} - \lambda_{\min})}{2T(1-p)} \tag{78}$$

For $k = 10$, $T = 2400$ (300 epochs $\times$ 8 batches), $p = 0.4$: max rate ≈ 0.0035 per iteration.

Intuition.

- **Early training** ($t < pT$): $\lambda_t \approx 0$, so $\mathcal{L}_G \approx \mathcal{L}_{\text{GAN}} + \mathcal{L}_{\text{degree}}$. The generator learns realistic graph structures and degree distributions without class-specific pressure.
- **Transition** ($t \approx pT$ to $t \approx T$): λ_t smoothly increases from 0 to 1.
- **Late training** ($t \approx T$): $\lambda_t \approx 1$, so $\mathcal{L}_G \approx \mathcal{L}_{\text{GNN}} + \mathcal{L}_{\text{degree}}$. The generator produces class-specific graphs while the degree loss maintains structural validity.

3.6.3 Generator Training Step

Algorithm 2 Single Generator Step

Require: Batch size B, current λ_t

1: Sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{d_z})^B$
2: $(\tilde{\mathbf{A}}, \tilde{\mathbf{X}}) \leftarrow G_\phi(\mathbf{z}, \tau = 1.0, \text{hard} = \text{True})$
3: $\mathcal{L}_{\text{GAN}} \leftarrow -\text{mean}(D_\psi(\tilde{\mathbf{X}}, \tilde{\mathbf{A}}))$
4: $\boldsymbol{\ell} \leftarrow f_\theta(\tilde{\mathbf{X}}, \tilde{\mathbf{A}})$ ▷ Forward through frozen k-GNN via dense wrapper
5: $\mathcal{L}_{\text{GNN}} \leftarrow \text{CE}(\boldsymbol{\ell},\ c \cdot \mathbf{1}_B)$
6: $\mathcal{L}_{\text{degree}} \leftarrow$ Eq. (72)
7: $\mathcal{L}_G \leftarrow (1 - \lambda_t)\,\mathcal{L}_{\text{GAN}} + \lambda_t\,\mathcal{L}_{\text{GNN}} + \mathcal{L}_{\text{degree}}$
8: Update ϕ via Adam to minimise $\mathcal{L}_G$

3.6.4 Optimiser Details

Both generator and discriminator use Adam with:

$$\text{lr} = 0.001, \quad \beta_1 = 0.5, \quad \beta_2 = 0.999 \tag{79}$$

The reduced $\beta_1 = 0.5$ (vs. default 0.9) is standard practice for GAN training, reducing momentum to prevent oscillation between the generator and discriminator.

4 Stage 3: Evaluation

4.1 Evaluation Protocol

At evaluation time, the generator produces N_{samples} (default 100) explanation graphs with low temperature:

$$(\tilde{\mathbf{A}}_i, \tilde{\mathbf{X}}_i) = G_\phi(\mathbf{z}_i, \tau = 0.1, \text{hard} = \text{True}), \quad \mathbf{z}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{d_z}) \tag{80}$$

The low temperature ($\tau = 0.1$ vs. $\tau = 1.0$ during training) produces sharper discrete outputs.

Each graph is then evaluated through the frozen pretrained k-GNN (using the same dense wrapper as training) to obtain class probabilities.

4.2 Per-Graph Metrics

For each generated graph $\tilde{G}$, we compute:

Edge thresholding. The evaluation operates on hard-thresholded adjacency:

$$\mathbf{A}_{ij}^{\text{eval}} = \begin{cases} 1 & \text{if } \tilde{\mathbf{A}}_{ij} > 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{81}$$

Self-loops are removed: $\mathbf{A}_{ii}^{\text{eval}} = 0$. Symmetry is enforced: $\mathbf{A}^{\text{eval}} = \max(\mathbf{A}^{\text{eval}}, (\mathbf{A}^{\text{eval}})^\top)$.

Active nodes and edges.

$$\text{degree}(v) = \sum_u \mathbf{A}_{vu}^{\text{eval}} \tag{82}$$

$$\tilde{V} = \{v : \text{degree}(v) > 0\}, \quad |\tilde{V}| = \text{active nodes} \tag{83}$$

$$|\tilde{E}| = \frac{1}{2}\sum_{i,j} \mathbf{A}_{ij}^{\text{eval}}, \quad \bar{d} = \frac{2|\tilde{E}|}{|\tilde{V}|} \tag{84}$$

Prediction probability.

$$p = \text{softmax}(f_\theta(\tilde{G}))_c = \frac{\exp(f_\theta(\tilde{G})_c)}{\sum_j \exp(f_\theta(\tilde{G})_j)} \tag{85}$$

Embedding similarity. Cosine similarity between $\tilde{G}$'s embedding and the class centroid:

$$s = \frac{\mathbf{z}_{\tilde{G}} \cdot \bar{\mathbf{z}}_c}{\|\mathbf{z}_{\tilde{G}}\| \cdot \|\bar{\mathbf{z}}_c\|}, \quad \bar{\mathbf{z}}_c = \frac{1}{|\mathcal{D}_c|}\sum_{G \in \mathcal{D}_c} \mathbf{z}_G \tag{86}$$

Remark 6 (Current simplification)**.** *The implementation currently sets $s = p$ (embedding similarity equals prediction probability), making the effective validation score $v = (p^2 \cdot d)^{1/3}$. Computing the true centroid $\bar{\mathbf{z}}_c$ would require a forward pass over the entire training set of class c to collect embeddings.*

Degree score. Gaussian kernel measuring structural plausibility:

$$d = \exp\left(-\frac{(\bar{d} - \mu_c)^2}{2\sigma_c^2}\right) \in [0, 1] \tag{87}$$

This is maximised ($d = 1$) when $\bar{d} = \mu_c$ and decays exponentially. The decay rate depends on σ_c:

- MUTAG ($\sigma_c \approx 0.07$): d drops below 0.5 when $|\bar{d} - \mu_c| > 0.083$. Very strict.
- PROTEINS ($\sigma_c \approx 0.42$): d drops below 0.5 when $|\bar{d} - \mu_c| > 0.50$. More forgiving.

Validation score. Geometric mean of the three components:

$$v = (s \cdot p \cdot d)^{1/3} \tag{88}$$

The geometric mean is sensitive to low values in *any* component — a graph cannot achieve $v > 0.5$ unless all of $s, p, d > 0.125$.

Validity. A graph is "valid" if both conditions hold:

1. Degree within tolerance: $|\bar{d} - \mu_c| \leq 3\sigma_c$
2. Score above threshold: $v \geq 0.5$

Granularity. Measures how much smaller the explanation is than typical graphs:

$$\kappa = 1 - \min\left(1, \frac{|\tilde{V}|}{\bar{n}_c}\right) \in [0, 1) \tag{89}$$

where $\bar{n}_c$ is the mean node count in class c. High κ means a fine-grained (small, focused) explanation.

5 Gradient Flow Analysis

Understanding where gradients flow (and don't) is critical for interpreting what the generator can optimise.

5.1 Full Gradient Decomposition

$$\frac{\partial \mathcal{L}_G}{\partial \phi} = \frac{\partial \mathcal{L}_G}{\partial \tilde{\mathbf{A}}} \cdot \frac{\partial \tilde{\mathbf{A}}}{\partial \phi} + \frac{\partial \mathcal{L}_G}{\partial \tilde{\mathbf{X}}} \cdot \frac{\partial \tilde{\mathbf{X}}}{\partial \phi} \tag{90}$$

Through $\tilde{\mathbf{A}}$.

- $\mathcal{L}_{\text{GAN}}$: Gradient flows through D_ψ, which uses $\tilde{\mathbf{A}}$ via $\hat{\mathbf{A}} = (\tilde{\mathbf{A}} + \mathbf{I})/\mathbf{D}$ in the DenseGCN layers. ✓
- $\mathcal{L}_{\text{GNN}}$ (1-GNN): Flows through $\tilde{\mathbf{A}}\mathbf{H}\mathbf{W}_2$ at each layer. ✓
- $\mathcal{L}_{\text{GNN}}$ (2-GNN): Flows through (a) iso-type feature $\tilde{\mathbf{A}}_{ij}$, (b) einsum aggregation $\sum_w \tilde{\mathbf{A}}_{iw} \cdot \mathbf{G}[j, w]$, (c) self-loop removal mask. ✓
- $\mathcal{L}_{\text{GNN}}$ (3-GNN): Partial — flows through soft iso-types (Eq. 22) but not structure selection. ∼
- $\mathcal{L}_{\text{degree}}$: $\bar{d} = \frac{2}{|\tilde{V}|} \sum_{i<j} \tilde{\mathbf{A}}_{ij}$, directly differentiable. ✓

Through $\tilde{\mathbf{X}}$.

- $\mathcal{L}_{\text{GAN}}$: Gradient flows through D_ψ input layer. ✓
- $\mathcal{L}_{\text{GNN}}$: Flows through the initial embedding $\mathbf{H}^{(0)} = \tilde{\mathbf{X}}$ in all k-GNN levels. ✓
- $\mathcal{L}_{\text{degree}}$: No dependence on $\tilde{\mathbf{X}}$. —

5.2 Through Gumbel-Softmax

With `hard=True`, the straight-through estimator gives:

$$\frac{\partial \tilde{\mathbf{A}}_{ij}}{\partial \mathbf{L}_{ij}} \approx \frac{\partial \,\text{softmax}([\mathbf{L}_{ij}/\tau, -\mathbf{L}_{ij}/\tau])_0}{\partial \mathbf{L}_{ij}} \tag{91}$$

Let $q = \text{softmax}([\mathbf{L}/\tau, -\mathbf{L}/\tau])_0 = \sigma(2\mathbf{L}/\tau)$. Then:

$$\frac{\partial q}{\partial \mathbf{L}} = \frac{2}{\tau}\, q\,(1 - q) \tag{92}$$

At $\tau = 1$: derivative peaks at $\mathbf{L} = 0$ with value 0.5, decays for large $|\mathbf{L}|$. At $\tau = 0.1$: derivative is sharply peaked around $\mathbf{L} = 0$ with peak value 5.0.

5.3 Gradient Flow Summary by Model

Path	1-GNN	1-2-GNN	1-2-3-GNN
$\mathcal{L}_{\text{GAN}} \xrightarrow{\tilde{\mathbf{A}}} \phi$	Full	Full	Full
$\mathcal{L}_{\text{GAN}} \xrightarrow{\tilde{\mathbf{X}}} \phi$	Full	Full	Full
$\mathcal{L}_{\text{GNN}} \xrightarrow{\tilde{\mathbf{A}}} \phi$ (1-GNN)	Full	Full	Full
$\mathcal{L}_{\text{GNN}} \xrightarrow{\tilde{\mathbf{A}}} \phi$ (2-GNN)	—	Full	Full
$\mathcal{L}_{\text{GNN}} \xrightarrow{\tilde{\mathbf{A}}} \phi$ (3-GNN)	—	—	Partial
$\mathcal{L}_{\text{degree}} \xrightarrow{\tilde{\mathbf{A}}} \phi$	Full	Full	Full

6 Putting It All Together

Algorithm 3 Full Pipeline

Input: Dataset $\mathcal{D}$, target class c, model type $k \in \{1, 12, 123\}$

// Stage 1: Train k-GNN
Initialise f_θ with appropriate architecture (Section 2.5)
for epoch $= 1, \ldots, 100$ **do**
    for each batch $\mathcal{B} \subset \mathcal{D}_{\text{train}}$ **do**
        $\mathcal{L}_{\text{CE}} \leftarrow$ Eq. (28)
        Update θ via Adam($\eta = 0.01$)
    end for
end for
Restore best θ (by test accuracy), freeze θ

// Stage 2: Train GIN-Graph
Compute class stats: $\mu_c, \sigma_c, \bar{n}_c$ from $\mathcal{D}_c = \{G \in \mathcal{D} : y = c\}$
Initialise generator G_ϕ (Section 3.2), discriminator D_ψ (Section 3.3)
Initialise weight scheduler (Eq. 75) with $T =$ epochs $\times\, |\mathcal{D}_c|/B$
for epoch $t = 0, \ldots, 299$ **do**
    for each batch of B graphs from $\mathcal{D}_c$ **do**
        Discriminator: $n_{\text{critic}} = 1$ steps of Algorithm 1
        Generator: one step of Algorithm 2 with current λ_t
    end for
end for

// Stage 3: Generate and Evaluate
for $i = 1, \ldots, N_{\text{samples}}$ **do**
    $(\tilde{\mathbf{A}}_i, \tilde{\mathbf{X}}_i) = G_\phi(\mathbf{z}_i, \tau = 0.1)$
    Forward through frozen f_θ via dense wrapper $\rightarrow$ compute p_i
    Threshold $\tilde{\mathbf{A}}_i > 0.5 \rightarrow$ compute $\bar{d}_i, d_i, v_i, \kappa_i$
end for
Rank by v_i, report top-k explanations

Dataset	Model	Class	Test Acc	Valid %	Val Score	Pred Prob
MUTAG	1-GNN	0 (Mutagen)	86.8%	38%	0.396	1.000
MUTAG	1-GNN	1 (Non-Mut.)	86.8%	100%	1.000	1.000
MUTAG	1-2-GNN	0 (Mutagen)	89.5%	30%	0.304	1.000
MUTAG	1-2-GNN	1 (Non-Mut.)	89.5%	94%	0.811	0.990
PROTEINS	1-GNN	0 (Non-Enz.)	78.0%	100%	0.941	0.918
PROTEINS	1-GNN	1 (Enzyme)	78.0%	100%	0.995	1.000
PROTEINS	1-2-GNN†	0 (Non-Enz.)	65.0%	97%	0.655	0.590
PROTEINS	1-2-GNN†	1 (Enzyme)	65.0%	44%	0.412	0.410

Table 1: Results across all configurations. †Only 30 epochs (vs. 300 for others).

7 Observed Results and Interpretation

7.1 Summary Table

7.2 Key Observations

1. Class asymmetry in MUTAG. Class 1 (Non-Mutagen) is dramatically easier to explain than class 0 (Mutagen). This stems from two factors:

- **Degree statistics**: MUTAG has $\sigma_c \approx 0.07$, making the degree score d in Equation (87) extremely sensitive. Even small structural deviations yield $d \approx 0$.
- **Feature complexity**: Mutagenic compounds require specific functional groups (nitro groups, halogens), while non-mutagenic molecules are structurally simpler (carbon backbones).

2. Higher-order $\neq$ better explanations. On MUTAG, the 1-GNN produces *better* explanations than the 1-2-GNN despite lower classification accuracy. This is significant for the thesis question:

- The 1-2-GNN has higher expressive power (can distinguish more graph structures).
- But its explanations collapse to simpler structures (mostly carbon-only graphs).
- This suggests the pair-level features in the 1-2-GNN encode *structural patterns* rather than *chemical features*, making it harder for the generator to satisfy both structure and chemistry simultaneously.

3. PROTEINS is more amenable. The wider degree distribution ($\sigma_c \approx 0.42$) makes validity easier to achieve. The 1-GNN achieves near-perfect scores on both classes.

8 Open Questions and Directions

1. **Embedding similarity.** Currently $s = p$, so $v = (p^2 d)^{1/3}$. Computing the true class centroid $\bar{\mathbf{z}}_c$ would decouple prediction confidence from embedding space alignment. Does the generator produce graphs that cluster with real graphs in embedding space, or does it find adversarial shortcuts?
2. **1-2-3-GNN explanations.** The most expressive model has not been tested. The 3-GNN component has only partial gradient flow (structure is computed without gradients). Does this limitation prevent the generator from learning, or does the soft iso-type signal suffice?

3. **MUTAG class 0 difficulty.** The tight degree distribution ($\sigma = 0.07$) severely penalises the generator. Possible directions:
   - Adaptive degree tolerance (wider early, tighter late)
   - Multi-scale degree loss (penalise both average degree and degree distribution shape)
   - Curriculum learning: start with relaxed validity, progressively tighten
4. **Expressiveness–interpretability tradeoff.** The 1-2-GNN is more expressive but produces less diverse explanations. Is this a fundamental tension? The k-WL hierarchy distinguishes more structures, but the space of "maximally activating" graphs may shrink as the model becomes more discriminative.
5. **Beyond degree score.** The current validity metric uses only average degree. Richer structural metrics could include:
   - Clustering coefficient distribution
   - Motif counts (triangles, cycles)
   - Graph spectrum alignment (eigenvalues of the Laplacian)
6. **Temperature scheduling.** The Gumbel-Softmax temperature τ is fixed at 1.0 during training and 0.1 at evaluation. Annealing τ during training (high $\rightarrow$ low) could improve discrete structure quality.
7. **Per-class generator architecture.** Each GIN-Graph generator is trained for a single class. A class-conditional generator $G_\phi(\mathbf{z}, c)$ could share learned graph structure priors across classes while specialising the output.

9 Notation Reference

Symbol	Meaning
$G = (V, E, \mathbf{X})$	Graph with nodes, edges, features
$\mathbf{A} \in \{0,1\}^{n \times n}$	Adjacency matrix
$\mathbf{X} \in \mathbb{R}^{n \times d}$	Node feature matrix
d	Input feature dimension
d_h	k-GNN hidden dimension (default 64)
d_g	Generator/discriminator hidden dimension (default 128)
d_z	Latent noise dimension (default 32)
n	Maximum number of nodes for generation
C	Number of classes
$\mathbf{h}_v^{(t)}$	Node v embedding at layer t
$\mathbf{f}_S^{(t)}$	k-set S embedding at layer t
$\mathbf{z}_G^{(k)}$	Graph-level embedding from k-GNN level
$\mathbf{W}_1, \mathbf{W}_2$	Self-transform and neighbour-transform weights
f_θ	Pretrained k-GNN classifier (frozen)
G_ϕ	Generator network
D_ψ	Discriminator network
$\tilde{\mathbf{A}}, \tilde{\mathbf{X}}$	Generated adjacency / features
τ	Gumbel-Softmax temperature
λ_t	Dynamic weight at step t
λ_{GP}	Gradient penalty weight (default 10)
λ_d	Degree loss weight (default 1)
μ_c, σ_c	Class c degree mean / std
$\bar{n}_c$	Class c mean node count
p	Prediction probability
s	Embedding similarity
d (metric)	Degree score
v	Validation score $(s \cdot p \cdot d)^{1/3}$
κ	Granularity