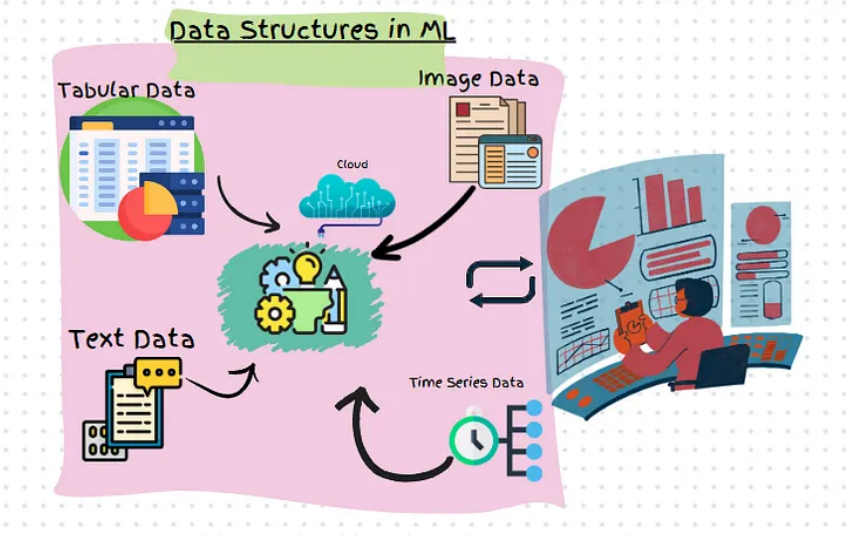
Understanding and Explaining Graph Neural Networks: From Post-hoc Methods to Self-Explainable Models

Céline Robardet – joint work with Marc Plantevit and Alessio Ragno

LIRIS - Laboratoire d'InfoRmatique en Image et Systèmes d'information (UMR 5205)

May 6, 2026

Data structures and Machine Learning

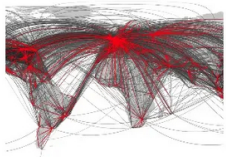


Adithya Prasad Pandelu, Understanding Data in ML, medium.

Graph Data



Social Graphs



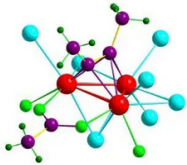
Transportation Graphs



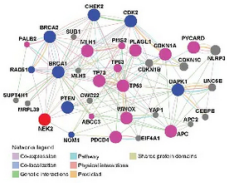
Brain Graphs



Web Graphs



Molecular Graphs

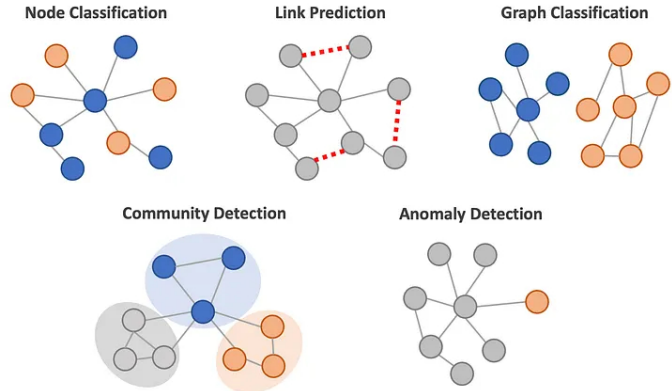


Gene Graphs

Explainability of Graph Neural Network Explaining the predictions of Graph Neural Network using GNNExplainer, medium.

Machine Learning tasks on Graphs

- **Node classification:** predict labels of unlabeled nodes
- **Link prediction:** predict missing or future edges between nodes
- **Graph classification:** assign a label to an entire graph
- **Community detection:** identify groups of densely connected nodes
- **Anomaly detection:** detect unusual nodes, edges, or subgraphs



Graph Neural Networks with PyG on Node Classification, Link Prediction, and Anomaly Detection, Tomonori Masui, medium.

Graph comparison: a key challenge

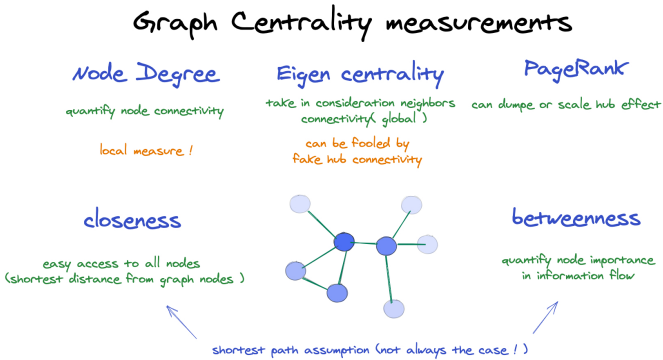
- Fundamental for graph learning tasks: clustering, classification, retrieval
- Many ML methods rely on similarities or distances between instances
- **Many problems related to it are NP-complete**
  - Subgraph isomorphism,
  - Finding largest common subgraph

We are interested in algorithms capable of measuring the similarity between two graphs in **polynomial time**.

Pre-neural ML on Graphs

- **Two-step pipeline:**
  - Extract handcrafted graph features
  - Train a model on feature vectors
- **Key limitation:**
  - Performance strongly depends on feature quality

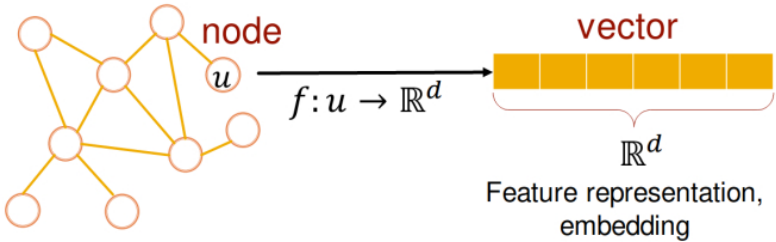
E.g., MLP on the features



Notes on graph theory – Centrality measures, Anas Ait Aomar, medium.

Toward Graph Representation Learning

- Learn **low-dimensional embeddings** for nodes
- Goal: **Similar nodes have similar representations**
- Enables:
  - Automatic feature learning
  - Better generalization across tasks



Jure Leskovec, `https://web.stanford.edu/class/cs224w/`.

Graph Representation Learning

Encode nodes so that similarity in the embedding space ≈ similarity in the graph.

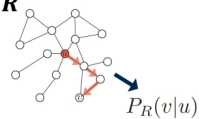
Three design choices:

- **Encoder**: $Enc : V \rightarrow \mathbb{R}^{d \times |V|}$, $Enc(u) := z_u$. When it is learnt, we have an early GNNs.
- **Decoder**: Given two embeddings, measure the similarity. Usually chosen to be the dot product $Dec(z, w) = z \cdot w$
- **Similarity in the graph space**: Any distance measure between nodes in the graph:

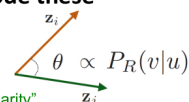
$\Rightarrow$ $similarity(u, v) \approx Dec(Enc(u), Enc(v))$

DeepWalk

1. **Estimate probability of visiting node v on a random walk starting from node u using some random walk strategy R**



2. **Optimize embeddings to encode these random walk statistics:**



Similarity in embedding space (Here: dot product=$\cos(\theta)$) encodes random walk "similarity"

Jure Leskovec, `https://web.stanford.edu/class/cs224w/`.

Toward Graph Neural Networks

- Deep learning is highly effective on regular data structures: images (grids), signals, sequences
- Graphs are irregular and more complex than grids:
  - no fixed node ordering
  - variable number of nodes and edges
  - no natural reference frame
  - often dynamic / evolving structure
- Standard CNNs/MLP models do not directly apply to graphs
- **Graph Neural Networks (GNNs)** learn directly from graph structure and node features
- We use a **Deep Graph Encoder (Enc)** to build graph representations

Graph Neural Networks

GNNs are designed to respect graph symmetries using two types of function:

Permutation **equivariant** functions

Reordering the nodes reorders the outputs in the same way.

Example: Node embeddings / node classification

$$f(PX, PAP^\top) = Pf(X, A)$$

Output follows the same shuffle

Permutation **invariant** functions

Reordering the nodes does **not** change the output.

Example: Graph classification

$$f(PX, PAP^\top) = f(X, A)$$

Output remains identical

GCN layers are equivariant, pooling/readout layers are invariant.

Graph Convolutional Networks (GCN)

GCNs learn node representations by aggregating information from neighbors.

One GCN Layer

For each node:

1. Collect embedding vectors from neighbors
2. Aggregate them (sum / mean)
3. Combine with own embedding
4. Apply a non-linear activation (e.g. ReLU)



GCN = progressive diffusion of information

Why is GCN layer equivariant?

- it relies on local graph structure
- shared weights
- symmetric aggregation

Therefore, outputs do not depend on node ordering.

Graph-level prediction based on

sum / mean pooling

This operation is **permutation invariant**.

GraphSAGE

GraphSAGE learns node representations by sampling and aggregating neighbor information.

One GraphSAGE Layer

For each node:

1. Sample a fixed-size set of neighbors
2. Aggregate neighbor embeddings using (mean / max / LSTM)
3. Concatenate with node's own embedding
4. Apply a learnable transformation + ReLU

GraphSAGE is designed for large graphs:

- avoids using all neighbors
- fixed computation per node
- efficient mini-batch training
- scalable to millions of nodes

Graph Isomorphism Network (GIN)

For each node:

1. Aggregate neighbor embeddings using **sum**
2. Add node's own embedding
3. Weight self-information with ϵ
4. Apply an MLP + ReLU

$$h_v^{(k)} = \text{MLP}^{(k)}\left((1+\epsilon)h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)}\right)$$

Why using MLP?

- Adds nonlinearity after sum aggregation
- Makes mapping over neighborhoods injective (expressive): distinguishes graph structures better than mean/max aggregators
- Enables separation of structurally different graphs

Using **sum aggregation** preserves multiset information:

- mean may lose counts
- max may lose frequency information
- sum is more discriminative

Graph Attention Network (GAT)

Not all neighbors contribute equally

For each node:

1. Compute linear projection of node embeddings
2. Compute attention scores between neighbors
3. Normalize attention coefficients (softmax)
4. Aggregate weighted neighbor embeddings
5. Apply nonlinearity (ReLU)

$$h_v^{(k)} = ReLU\left(\sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(k)} W^{(k)} h_u^{(k-1)}\right)$$

Learns **which neighbors matter**:

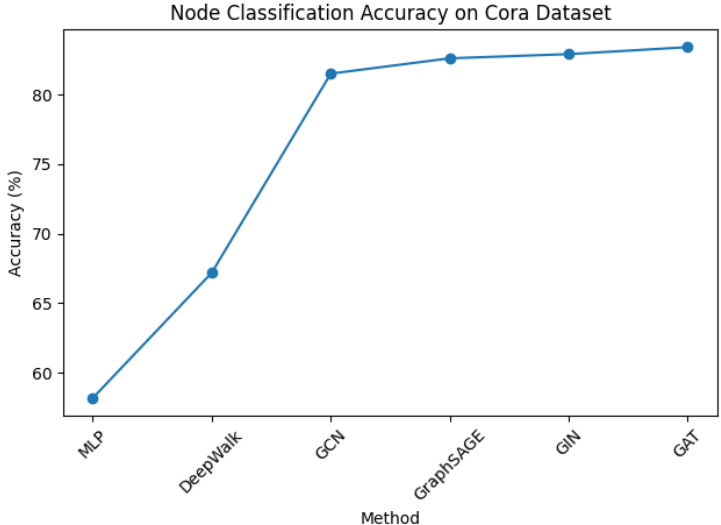
- replaces fixed mean/sum with learned weights
- adaptive importance per edge
- important neighbors get higher attention
- irrelevant neighbors are downweighted
- attention is learned end-to-end

Attention Coefficients

$$\alpha_{vu} = \text{softmax}_u\left(\text{LeakyReLU}\left(a^\top [Wh_v \,\|\, Wh_u]\right)\right)$$

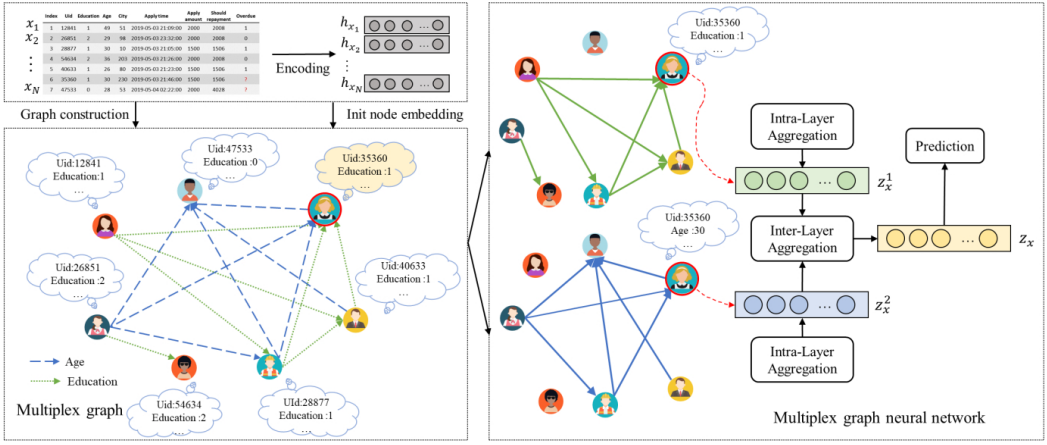
Multi-head attention = richer relations

Graph Neural Networks Performances



Data from Xing Li, Wei Wei, Ruizhi Zhang, Zhenyu Shi, Zhiming Zheng, and Xiangnan Feng. 2023. Representation Learning of Enhanced Graphs Using Random Walk Graph Convolutional Network. ACM Trans. Intell. Syst. Technol. 14, 3, Article 46 (March 2023)

GNNs can be applied to many data types, including tabular data



TabGNN: Multiplex Graph Neural Network for Tabular Data Prediction: Xiawei Guo et al., DLP-KDD, 2021.

The Interpretability Challenge in GNNs

- **Graph Neural Networks (GNNs)** achieve state-of-the-art results in:
  - Node classification
  - Link prediction
  - Graph classification
- **Problem**: Complex message-passing mechanisms make predictions opaque
- **Impact**: Limited adoption in critical domains
  - Healthcare
  - Finance
  - Scientific discovery

→ *Interpretability is essential for trust and regulatory compliance*

Explaining GNNs: Post-hoc Approaches

The first generation of techniques for interpreting trained GNN models.

Given a trained GNN f_θ, identify which parts of the graph are most influential for a prediction:

- important nodes
- important edges
- important node features

Find a mask $\mathcal{E}$ such that the model decision is preserved:

$$\mathcal{E}^* = \arg\max_{\mathcal{E}} \mathcal{F}\big(f_\theta(G), f_\theta(G \odot \mathcal{E})\big)$$

with $\mathcal{F}$ is a fidelity function.

Objectives

- remove irrelevant information
- keep decisive structure
- maintain prediction fidelity

Main families depends on how the mask is learned:

- gradient-based
- optimization-based
- perturbation methods

Explain = find the smallest graph evidence supporting the prediction

Why explanations of black boxes may be misleading?

Main arguments from Cynthia Rudin

1. There is no necessarily a trade-off between accuracy and interpretability:

- It is often assumed that accurate models must be complex
- Interpretable models can achieve competitive performance

2. Explanations are not always faithful to what the original model computes

- they approximate the original model
- they are not guaranteed faithful
- they may oversimplify complex behavior

3. Why fidelity is impossible
If an explanation were perfectly faithful:

$$\text{Explanation} = \text{Original Model}$$

4. Practical risks

- explanations may rely on different features to the ones used by the model
- they may hide unstable decisions
- they may not make sense to users
- they may omit critical details

For high-stakes decisions, transparency should be built into the model itself.

From Explainable ML to Interpretable ML

Explainable ML

- Train black-box model first
- Add post-hoc explanation later
- Example: SHAP, LIME, saliency maps
- Explanation may be incomplete
- Faithfulness not guaranteed

Interpretable ML

- Model is transparent by design
- Prediction process directly understandable
- No second explainer required
- Easier auditing and debugging
- Better suited for regulation

Interpretability should be built in, not added afterward.

Logic-Based Explainability[1][2][3]

Why Logic?

- Human-interpretable rules
- Explicit feature interactions
- Globally consistent

Graded Modal Logic (GML)

- Natural fit for graphs
- Express numerical constraints
- Example: "At least 3 neighbors satisfy property P"

Existing Work

- **Barceló et al.**: GNNs can represent GML statements
- **TELL**: Logic rules layer from tabular data
- Problem: No GNN architecture for extracting logic rules

---

[1] Barceló, Pablo, et al. "The logical expressiveness of graph neural networks." International conference on learning representations. 2020.

[2] Pierre Nunn, Marco Sälzer, François Schwarzentruber, Nicolas Troquard: A Logic for Reasoning about Aggregate-Combine Graph Neural Networks. IJCAI 2024: 3532-3540

[3] Benedikt, M., Lu, C. H., & Tan, T. (2024). Decidability of graph neural networks via logical characterizations. arXiv preprint arXiv:2404.18151.

LogiX-GIN: Self-Explainable Logic GNN

Key Innovation: First GNN architecture intrinsically convertible to logic rules

Technical Components

1. **Based on GIN architecture**: Sum aggregation enables counting (GML compatibility)
2. **Novel preprocessing**: Automatic feature binarization via learnable intervals
3. **Distillation-based pretraining**: Address vanishing gradients in multi-layer settings

Training Strategy:

- Pretrain black-box GIN
- Layer-wise distillation to LogiX-GIN
- Extract interpretable GML rules directly from weights

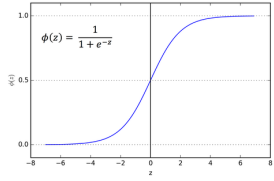
Bridging the gap between GNN performance and logical interpretability

On Logic-based Self-Explainable Graph Neural Networks Alessio Ragno Marc Plantevit Céline Robardet, NeurIPS 2025.
https://openreview.net/forum?id=OtAiYPP6GA

Transparent Explainable Logic Layer (TELL)[4]

Architecture: Constrained linear layer with non-negative weights

$$y = \sigma(XW^+ + b), \quad W^+ \in \mathbb{R}^{I \times O}_{\geq 0}$$



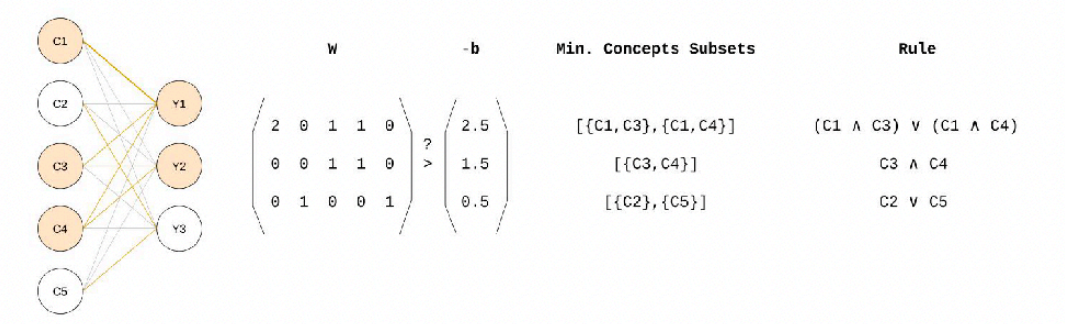
Logic Rule Extraction:
Binarized output $y^{\text{bin}}[j] = \mathbb{1}_{y[j]>0.5}$ satisfies:

$$y^{\text{bin}}[j] = 1 \iff y[j] > 0.5 \iff XW^+[j] + b[j] > 0 \iff \sum_{i \in S} W^+[i,j] > -b[j]$$

Logic rule in DNF: $E_j = \bigvee_{S \in \mathcal{S}_j} \bigwedge_{i \in S} x_i$, where $\mathcal{S}_j$ contains minimal input subsets satisfying the threshold.

[4] Alessio Ragno, Marc Plantevit, Céline Robardet, Roberto Capobianco: Transparent Explainable Logic Layers. ECAI 2024

TELL Example



TELL layer can be used as last GNN layer to provide logic-based explanations[5].

[5] Alessio Ragno, Marc Plantevit, Céline Robardet: Faithful Explanations for Graph Classification Using Logic. ECML/PKDD (4) 2025: 113-129

Graded Modal Logic

Definition: Fragment of first-order logic for expressing local graph properties with counting.

Language:

- Binary relation: $E(x,y)$ (adjacency)
- Unary predicates: $\{P_1(x),\ldots,P_l(x)\}$ (node properties)

Properties:

- Local evaluation: depends only on node and its neighborhood
- All quantification conditioned by adjacency $E(x,y)$
- Less expressive than first-order logic, but more tractable for graphs

Graded Modal Logic - Semantics

Graph Structure: $G = (V, E, \{P_i\}_{i \in I})$

- V: finite set of nodes
- $E \subseteq V \times V$: symmetric edge relation
- $P_i \subseteq V$: unary relations interpreting predicates

Satisfaction Relation:

- $G, v \models P(x)$: node v satisfies predicate P
- $G, v \models \neg\psi(x)$: node v does not satisfy $\psi(x)$
- $G, v \models \psi_1(x) \wedge \psi_2(x)$: v satisfies both formulas
- $G, v \models \psi_1(x) \vee \psi_2(x)$: v satisfies at least one formula
- $G, v \models \exists^{\geq N} y\,(E(x,y) \wedge \psi_3(y))$: node v has at least N neighbors u such that u satisfies ψ_3

Connection to GNNs: Barceló et al. show GML precisely characterizes node properties computable by some GNN classes.

Graded Modal Logic - Examples

Unary Predicates: Blue(x), Red(x), Green(x) denote node colors

Example 1: Atomic predicate

$$\varphi(x) := \mathit{Blue}(x)$$

Formula holds if x is blue.

Example 2: At least two red neighbors

$$\varphi(x) := \exists^{\geq 2} y\,(E(x,y) \wedge \mathit{Red}(y))$$

Node x has at least two neighbors that are red.

Example 3: Blue node without red neighbors

$$\varphi(x) := \mathit{Blue}(x) \wedge \neg\exists^{\geq 1} y\,(E(x,y) \wedge \mathit{Red}(y))$$

Node x is blue and has no red neighbors.

Graded Modal Logic - Examples (2)

Example 4: Disjunction of neighborhood conditions.

$$\varphi(x) := \left(\exists^{\geq 3} y\,(E(x,y) \wedge \mathit{Red}(y))\right) \vee \left(\exists^{\geq 1} y\,(E(x,y) \wedge \mathit{Green}(y))\right)$$

This means that x has either at least three red neighbors, or at least one green neighbor.

Example 5: Nested neighborhood property.

$$\varphi(x) := \exists^{\geq 1} y\,(E(x,y) \wedge \exists^{\geq 1} z\,(E(y,z) \wedge \mathit{Green}(z)))$$

This states that x has a neighbor y which itself has at least one green neighbor z.

Background Summary

Graph Neural Networks (GIN) Message-passing architecture with sum aggregation and MLP updates, providing maximum discriminative power.

Transparent Explainable Logic Layer (TELL) Neural layer with non-negative weight constraints enabling direct extraction of logic rules in DNF. Adaptive thresholding handles continuous inputs.

Graded Modal Logic (GML) First-order logic fragment for expressing local graph properties with counting quantifiers. Natural correspondence with GNN computations.

These three components form the foundation for LogiX-GIN

LogiX-GIN Layer Architecture

Core Operation:

$$h_v^{(k)} = \lambda^{(k)}\left(\beta^{(k)}\left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} h_u^{(k-1)}\right)\right)$$

Three-Step Process:

1. **Aggregation**: Sum embeddings from node v and neighbors

$$a_v^{(k)} = \sum_{u \in \mathcal{N}(v) \cup \{v\}} h_u^{(k-1)}$$

2. **Binarization** (β): Convert integer counts to binary via learnable intervals
3. **Logic Transformation** (λ): Apply TELL-style logic layer with non-negative weights

Sum aggregation enables counting, essential for GML compatibility

Parametric Fourier Step Function β

Challenge: Need to activate when counts fall within specific ranges

Solution: Fourier-based step function

$$\tilde{a}_v^{(k)} = \beta^{(k)}\left(a_v^{(k)}\right) = \text{clamp}\left(\frac{1}{2}\left(\frac{\sum_{i=0}^{\tau}\frac{\sin((2i+1)(a_v^{(k)}\odot\tilde{W}^{(k)}+\tilde{b}^{(k)}))}{2i+1}}{\sum_{i=0}^{\tau}\frac{\sin\left((2i+1)\frac{\pi}{2}\right)}{2i+1}}+1\right), 0, 1\right)$$

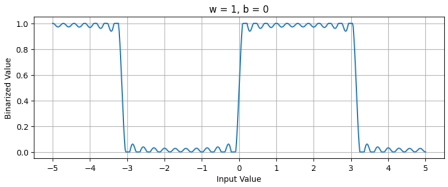
Key Properties:

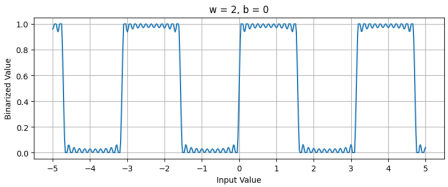
- Learnable parameters: $\tilde{W}^{(k)}, \tilde{b}^{(k)} \in \mathbb{R}^d_{>0}$
- Activates when count falls in intervals:

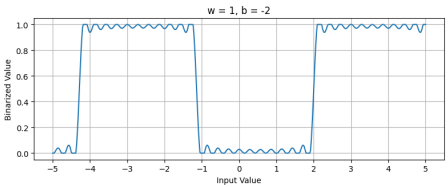
$$\tilde{a}[i]_v^{(k)} = 1 \iff a[i]_v^{(k)} \in \bigcup_{n\in\mathbb{Z}}\left[\frac{2\pi n-\tilde{b}[i]^{(k)}}{\tilde{W}[i]^{(k)}}, \frac{(2\pi n+\pi)-\tilde{b}[i]^{(k)}}{\tilde{W}[i]^{(k)}}\right]$$

- Enables counting-based literals in GML

Parametric Fourier Step Function β Examples







3 examples illustrating how the parameters w (width) and b (bias or displacement) affect the shape and position of the intervals.

1. $(w = 1, b = 0)$ shows a baseline case where, in this interval of input values $(-5 < x < 5)$, the output corresponds to a 1 if $x < -3$ or $0 < x < 3$.
2. $(w = 2, b = 0)$ highlights how w can be used to modify the width of the intervals
3. $(w = 1, b = -2)$ shows how the parameter b is instead responsible to apply a displacement of the thresholds (in this case of -2)

Logic Transformation λ

TELL-Inspired Layer:

$$h_v^{(k)} = \lambda^{(k)}\left(\tilde{a}_v^{(k)}\right) = \sigma\left(\frac{\tilde{a}_v^{(k)} W^{+(k)} + b^{(k)}}{\tau}\right), \quad \tau = 10^{-4}$$

Key Features:

- Non-negative weights: $W^{+(k)} \geq 0$
- Ensures monotonicity
- Enables direct logic rule extraction

Logic Rule Extraction:

- Rules in propositional logic over activations $\tilde{a}_v^{(k)}$
- Correspond to GML formulas over literals $h_u^{(k-1)}$
- Identified by weight sets where $\sum W^{+(k)} > -b^{(k)}$

Direct mapping from network parameters to logical expressions

LogiX-GIN Full Architecture

Pipeline:

1. **5 LogiX-GIN layers**: Learn hierarchical logic rules
2. **Global readout**: Sum, mean, max aggregation
3. **Final TELL layer**: Learn GML rules over graph-level features

Output: Graph-level predictions with interpretable logic rules

Training Strategy: Knowledge Distillation

Challenge: Multiple sigmoid activations cause vanishing gradients

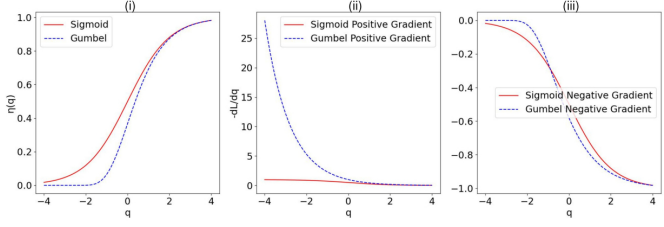
Two-Phase Training:

Phase 1: Layer-wise Distillation

1. Train auxiliary GIN model with Gumbel-sigmoid activations
2. Use GIN as teacher for LogiX-GIN
3. Minimize cross-entropy between hidden states at each layer
4. Provides layer-wise pretraining

Phase 2: End-to-End Finetuning

1. Standard supervised learning on full LogiX-GIN
2. Cross-entropy loss + L_1 regularization on W^+
3. Optimized via gradient descent



Post-Training Pruning for Interpretability

Motivation: Learned rules can be overly complex

Hoyer Regularization:

$$\mathcal{L}_{\text{Hoyer}} = \frac{\|W^+\|_1}{\|W^+\|_2 + \delta}$$

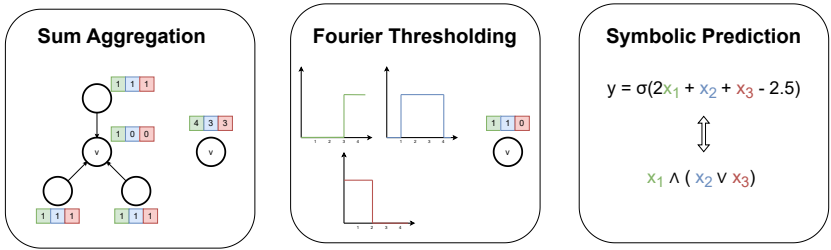
Pruning Procedure:

- Train layers individually in reverse order (last to first)
- Hoyer loss promotes structured sparsity
- Encourages few large weights, pushes others to zero
- Reduces rule complexity while maintaining performance

Benefits:

- Simpler, more readable logic rules
- Direct impact on rule size (fewer weights to sum for threshold)
- Minimal performance degradation

LogiX-GIN Pipeline Overview



$$\sum_{v' \in \mathcal{N}(v) \cup \{v\}} \geq 3 \wedge \left(1 \leq \sum_{v' \in \mathcal{N}(v) \cup \{v\}} \leq 4 \vee \sum_{v' \in \mathcal{N}(v) \cup \{v\}} \leq 2\right)$$

Node features from a target node and its neighborhood are aggregated by summation, then binarized by a learnable Fourier step function into discrete on/off literals. These binarized signals pass through a logic layer with non-negative weights, yielding interpretable logical activations.

Training: A teacher GIN model guides distillation, and Hoyer-style pruning removes redundant rules.

LogiX-GIN Explanations

Multiple Explanation Formats:

1. **Layer-wise Logic Rules**
   - Fine-grained rules for each layer
   - Exact reflection of model behavior
   - Detailed view of hierarchical reasoning
2. **Global Logic Rules**
   - Identify motifs that activate class-specific rules
   - High-level overview of decision process
   - Easier to interpret for end users
3. **Node Attributions**
   - Visualize node-level contributions
   - Shows which nodes matter for prediction
   - Complements logic-based explanations

Flexible explanation framework for different stakeholder needs

Comparison of LogiX-GIN and GIN

Table: Accuracy on Graph and Node Classification. Values are reported as $\mu \pm \sigma$ over 10 seeds. We report the accuracy of the black-box model and LogiX-GIN. For an easier comparison, the column "Acc. Decay" indicates the difference between the mean accuracy of the black-box and the LogiX-GIN.

Dataset	Black-box GIN	LogiX-GIN	Acc. Decay	Class. Obj.
BAMultiShapes	100.00 ± 0.00	100.00 ± 0.00	0.00	Graph
BA2Motifs	100.00 ± 0.00	100.00 ± 0.00	0.00	Graph
BBBP	87.95 ± 2.07	85.90 ± 0.99	2.05	Graph
MUTAG	84.74 ± 8.02	82.63 ± 5.58	2.11	Graph
NCI1	81.87 ± 1.39	78.93 ± 1.51	2.94	Graph
PROTEINS	72.59 ± 3.04	72.05 ± 5.77	0.54	Graph
Mutagenicity	82.21 ± 1.87	79.31 ± 1.25	2.90	Graph
BaShapes	97.14 ± 0.95	94.29 ± 2.52	2.85	Node
BaCommunity	81.00 ± 1.59	84.14 ± 4.55	-3.14	Node
TreeGrid	100.00 ± 0.00	98.71 ± 0.87	1.29	Node

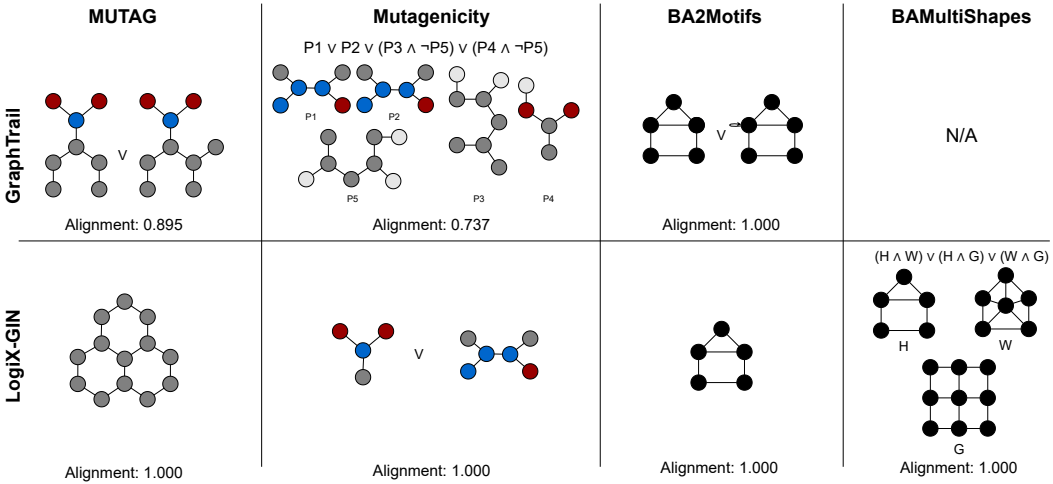
→ Incorporating logic has a limited impact on predictive accuracy overall.

Comparison of SE-GNNs across graph classification datasets

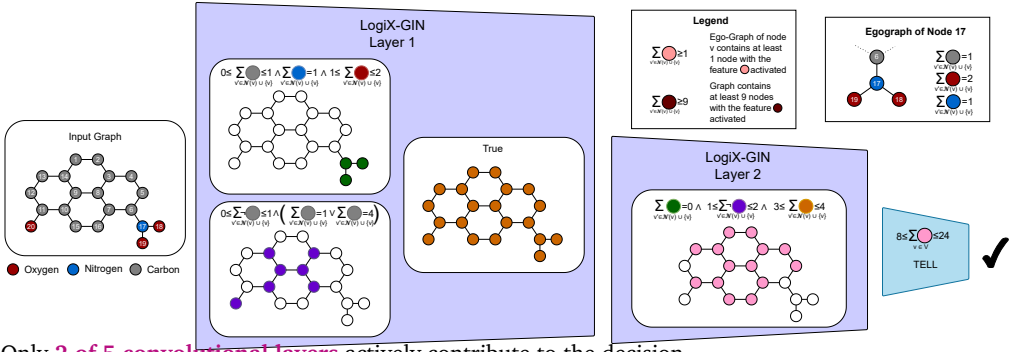
Dataset	LogiX-GIN	PiGNN	GIB	KerGNN	GNAN
Ba2Motifs	100.00 ± 0.00	99.89 ± 0.31	100.00 ± 0.00	98.80 ± 0.75	49.10 ± 0.66
BaMultiShapes	100.00 ± 0.00	85.40 ± 5.08	97.60 ± 2.06	83.20 ± 1.94	49.10 ± 0.66
MUTAG	82.63 ± 5.58	82.51 ± 10.48	90.53 ± 6.14	82.11 ± 9.18	55.79 ± 16.43
Mutagenicity	79.31 ± 1.25	82.39 ± 1.68	80.14 ± 0.98	73.32 ± 2.93	55.36 ± 0.52
NCI1	78.93 ± 1.51	78.54 ± 2.74	78.15 ± 1.32	69.00 ± 1.38	50.80 ± 1.15
PROTEINS	72.05 ± 5.77	70.00 ± 2.44	68.47 ± 5.00	72.97 ± 4.87	57.67 ± 2.62
BBBP	85.90 ± 0.99	83.54 ± 0.37	84.78 ± 2.57	84.12 ± 2.09	22.80 ± 0.86

→ Compared to other self-interpretable approaches, our method reaches state-of-the-art performances on 5 datasets out of 7.

Visual comparison between rules extracted with GraphTrail and LogiX-GIN

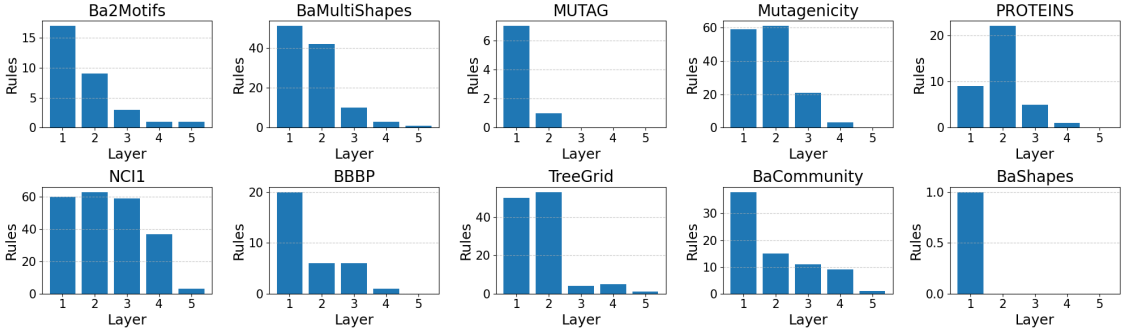


Explaining the single layers of LogiX-GIN (MUTAG)



- Only **2 of 5 convolutional layers** actively contribute to the decision.
- **Layer 1:** captures low-level atom features and detects functional groups.
  - Rule 1 (green): detects NO_2 group by counting N, O, and C atoms.
  - Rule 2 (violet): identifies aromatic rings via carbon-neighborhood patterns.
  - Feature 3 (orange): activates across all nodes used later for connectivity counting.
- **Layer 2:** combines features from Layer 1; decision based on Feature 4 (pink) activations.
- Model reasoning is **count-based**: predictions remain stable as long as key activations stay within learned ranges.

Analyzing rules' complexity



- Early layers contain the largest number of rules: they extract low-level graph patterns.
- Deeper layers combine or filter these patterns, often showing **redundancy**.
- In most datasets, only the first 1 or 2 layers are actively used for prediction.
- **Mutagenicity** and **NCI1** require more literals, reflecting higher molecular complexity and graph variability.
- Highlights the **hierarchical structure** of logic rules: dense low-level extraction selective high-level abstraction.

Painfull points

Table: Average and standard deviation of training times (in milliseconds) for GIN and LogiX-GIN across datasets (updated values).

Dataset	GIN (avg ± std)	LogiX-GIN (avg ± std)	Factor
Ba2Motifs	2727.00 ± 207.80	6671.67 ± 177.83	x 2.44
BaMultiShapes	1193.67 ± 35.37	6354.00 ± 201.92	x 5.3
MUTAG	279.00 ± 122.06	797.00 ± 26.87	x 2.8
Mutagenicity	3656.00 ± 40.90	5748.33 ± 27.38	x 1.5
PROTEINS	1872.33 ± 13.89	11077.00 ± 4.97	x 5.9
NCI1	1695.33 ± 48.80	49646.67 ± 636.76	x 29.2
BBBP	2044.00 ± 77.94	19855.00 ± 46.50	x 9.7
TreeGrid	265.33 ± 19.94	4361.00 ± 12.03	x 16
BaCommunity	497.33 ± 11.09	21057.67 ± 3.77	x 42
BaShapes	1072.67 ± 73.19	16680.67 ± 4.50	x 15.5

Conclusion

LogiX-GIN: A logic-based, self-explainable GNN

- Introduces logical reasoning *within* GNN layers for self-explainability.
- Each layer can be translated into logical formulas in **GML formalism**.
- Provides transparent and faithful representations of the models reasoning.
- Achieves competitive accuracy on both synthetic and real-world datasets for graph and node classification.

Key takeaway: LogiX-GIN bridges the gap between symbolic reasoning and neural computation, achieving interpretability *without* sacrificing performance.

Limitations and Future Directions

Current limitations:

- No support for **edge features**, reducing expressiveness.
- **Sigmoid activations** and learnable binarization cause training instability in deep or large graphs.
- **High computational cost** from multi-step distillation training.
- Thresholding on non-semantic features may reduce interpretability.
- Reasoning constrained by **GML expressivity**.

Future work:

- Integrate with **prototype- or concept-based** reasoning for case-based explanations.
- Explore more **scalable and efficient architectures**.
- Extend to richer **logic formalisms** or broader graph tasks.
- Combine multiple explanation forms (logic + prototypes) for deeper interpretability.