BEYOND TRAINED MODELS: COMPILING GNNS FOR A SOUND EXPLAINER BENCHMARK

Anonymous authors
Paper under double-blind review

ABSTRACT

Explainers for Graph Neural Networks (GNNs) are commonly evaluated by their *plausibility*, i.e., how well their explanations recover a predefined ground truth, such as a motif planted in the data. This protocol implicitly assumes that a GNN trained on such data relies on the intended motif. Although prior work has questioned this assumption, plausibility remains widespread. First, we show that the assumption is violated on several widely used benchmarks, where, e.g., degree statistics alone suffice to solve the task. Then, we remove this confounder by replacing training with *compilation*. We achieve this by introducing Gracr, the first compiler translating graded modal logic formulas into GNN weights, yielding models that replicate the behaviour of the corresponding formulas. Since the behaviour of the model is now known by construction, we can define its ground truth explanation formally and compute it exactly. Building on this, we introduce GracrBench, a benchmark of compiled GNNs for the evaluation of explainers against this exact ground truth. Experiments on eleven explainers across six tasks show its effectiveness for fine-grained diagnostic evaluation: notably, we discover that most explainers are not robust to indirect influences or alternative implementations of the same formula. These results position GracrBench as a novel, rigorous evaluation setting for graph post-hoc explainability. Code is available in the supplement.

1 INTRODUCTION

Explaining why a Machine Learning (ML) model produces a particular output is a core challenge for deploying these models in the real world. Graph Neural Networks (GNNs) (Scarselli et al., 2008; Bronstein et al., 2021) are no exception, and a rich literature has emerged to develop methods for explaining such models (Yuan et al., 2022; Amara et al., 2022; Longa et al., 2025). One of the most widely used techniques for evaluating the *quality* of GNN explainers is ***plausibility***[1], i.e., how well the explanation recovers a "ground-truth" pattern. It follows this blueprint: *First*, identify or create a dataset with a known discriminative pattern, e.g., graphs containing a house-like motif versus graphs containing a cycle-like motif; *then*, train a GNN on this dataset; *last*, evaluate how well the explanations produced by the explainer under evaluation recover the known discriminative pattern.

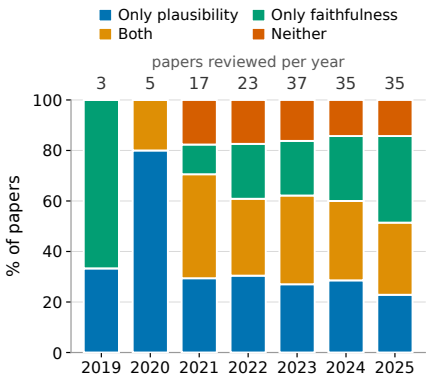


Figure 1: We manually reviewed 155 published GNN explainability papers (2019–2025), recording how they evaluate explanation quality. **Plausibility remains a core criterion**, used alone in at least 20% of papers each year. Numbers above bars indicate the number of papers analysed; details in Appendix G.

Previous work has already identified shortcomings of this blueprint (Faber et al., 2021; Himmelhuber et al., 2021; Agarwal et al., 2023; Fontanesi et al., 2024). Yet, despite these findings, our extensive literature review (see Fig. 1) shows that plausibility remains widely used: strikingly, roughly 25% of all 155 papers we surveyed adopt plausibility as the *only* criterion for evaluating explanations. The fundamental issue is that the trained GNN may learn other, potentially spurious

[1] A.k.a. explanation accuracy (Agarwal et al., 2023)

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

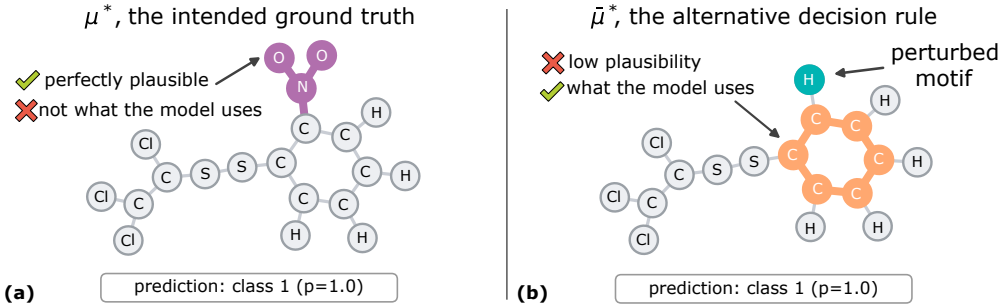


Figure 2: **Evaluating the plausibility of explanations is misleading when the model is not behaving as expected.** Example of a model trained on `Mutagenicity`, where plausibility penalizes well-behaved explainers. a) A correctly predicted mutagenic molecule, with the intended ground truth NO_2 pattern highlighted as the explanation μ^*. b) The same molecule, where we substituted the original μ^* with a single hydrogen atom. Crucially, the model prediction does not change, suggesting it has learned an alternative decision rule $\bar{\mu}^*$ (e.g., the carbon ring). As a result, a good explainer correctly providing $\bar{\mu}^*$as the explanation would be penalized in favour of a bad one providing μ^*. In this work, we fix this issue by building GNNs whose behaviour is known, as described in Section 4.1.

patterns present in the data that differ from the intended discriminative pattern, as illustrated by the example in Fig. 2. Consequently, we scrutinize the definition and usage of plausibility closely.

In our work, we first extend previous findings by identifying specific, widely used datasets where the above shortcoming is likely to occur, and then propose a new strategy for evaluating plausibility that replaces GNN training with a *compilation step*. We begin by identifying a pattern of interest, represented as a Boolean logic formula. Then, we introduce a compiler to transform this logic formula into a GNN that, by construction, shares the same behaviour and explanations as the formula. This raises the question of what the "ground truth" explanation of a formula actually is. Common practice answers with a designated motif, but already for simple patterns – such as *a node having more red than blue neighbours* – several subgraphs justify the prediction equally well. We therefore need a notion of ground truth that generalizes beyond simple motifs and accounts for alternative explanations. To do so, we borrow from formal XAI the definition of *prime implicant* (Darwiche & Hirth, 2020; Marques-Silva & Ignatiev, 2022) – i.e., a minimal subgraph that alone suffices to determine the prediction – and take the union of all prime implicants as the ground truth explanation. Finally, the explainer is evaluated based on how well its explanations recover such ground truth. Crucially, the compilation step allows us to control *exactly* the behaviour of the GNN, as it is equivalent to the logic formula from which it was derived: effects that are usually inextricably confounded on a trained model can be studied in controlled experiments, allowing specific hypotheses to be tested rigorously.

Summarizing, our contributions are as follows:

- We show that popular benchmarks are not a reliable test bed for evaluating plausibility, as the GNN over which explainers are evaluated may not behave as expected.
- We propose a compiler capable of translating a logic formula into the weights of a GNN. This yields GNNs where the exact behaviour is known and controllable, addressing the issue above.
- To derive a principled notion of what characterizes a ground truth explanation, we build on formal XAI and adopt *prime implicants*, and develop an efficient procedure to extract them.
- We exploit this compiler to propose a new, sound benchmark for evaluating plausibility. Our experiments recover previously known findings while uncovering new ones, exposing limitations of explainers in understudied scenarios beyond motif-based tasks.

2 BACKGROUND

Notation. A ***graph*** is a tuple $G = (\mathcal{V}, \mathcal{E})$ with nodes $\mathcal{V}$ and directed edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, where an edge e is an ordered pair of nodes $e = (u, v)$. Throughout, a ***subgraph*** $R \subseteq G$ is a *spanning* subgraph obtained from G by retaining all nodes in $\mathcal{V}$ and a subset of its edges. We write $E(u, v)$ for the predicate holding iff $(u, v) \in \mathcal{E}$, and $\mathcal{N}(v)$ for the neighbours of node v. We focus on binary classification tasks: a ***graph classifier*** is a function g which maps an input graph into $\{0, 1\}$; a ***node classifier***, instead, maps each node of a graph into $\{0, 1\}$.

Problem statement. A ***graph explainer*** is a function F mapping a classifier and an input graph to an ***explanation***, i.e., a vector of edge importance scores $F(g, G) = \mu \in \mathbb{R}^{|\mathcal{E}|}$, with μ_e the score of edge e. For node classification, F also takes the node to explain. The ***ground truth explanation*** is a binary mask $\mu^* \in \{0, 1\}^{|\mathcal{E}|}$. Finally, a ***plausibility*** score is any scoring function – like accuracy or variants thereof – between the predicted explanation μ and the ground truth μ^*, i.e., $\text{sim}(\mu, \mu^*) \in [0, 1]$ (Yuan et al., 2022; Agarwal et al., 2023). The choice of μ^* is crucial but often heuristic. Next, we describe cases in which an inappropriate choice can make plausibility misleading.

3 ISSUES WITH PREVIOUS EVALUATIONS OF PLAUSIBILITY

Evaluating the plausibility of an explainer typically boils down to the following blueprint (Agarwal et al., 2023). First, identify or create a dataset $\mathcal{D}$ of graphs containing a known discriminative pattern, which acts as the ground truth explanation μ^*. For example, in the `Mutagenicity` dataset (Kazius et al., 2005), the presence of a NO_2 group is usually associated with the mutagenicity that distinguishes the two classes and therefore serves as their ground-truth explanation (Ying et al., 2019; Luo et al., 2020). Then, the plausibility score between the post-hoc explanation μ and μ^* is computed.

As pointed out by previous work (Faber et al., 2021; Yuan et al., 2022; Agarwal et al., 2023; Monti et al., 2024), this blueprint makes the following *implicit* assumption, which may not hold in practice.

Assumption 1 (informal)**.** *A GNN trained to achieve high accuracy on a dataset with a known ground-truth explanation μ^* will learn only the discriminative pattern represented by μ^*.*

In fact, ML models are known to be susceptible to learning shortcuts in the data (Geirhos et al., 2020), which invalidates Assumption 1. Arguably, explainers are most useful exactly when they can reveal that the model is not behaving as expected (Lapuschkin et al., 2019; Teso et al., 2023). Nonetheless, ***evaluating plausibility when Assumption 1 does not hold can be misleading***, as illustrated in Fig. 2: If Assumption 1 held and the model had learned to distinguish molecule mutagenicity based solely on the ground truth NO_2 pattern, we would expect its prediction to change when this pattern is removed, provided that the resulting molecule remains sufficiently close to the data distribution. However, as exemplified in Fig. 2 and analysed in Appendix B.2, removing the NO_2 pattern often leaves the prediction of the model unchanged, suggesting that Assumption 1 does not hold, and that the model learned an alternative pattern. In this case, a well-behaved explainer outputting the alternative pattern would be penalized in favour of an explainer outputting μ^*.

Crucially, the shortcoming highlighted in Fig. 2 is likely to appear in both synthetic and real datasets. In Appendix B, we identify several of these cases on some of the most popular graph explainability benchmarks, extending Fontanesi et al. (2024)'s analysis to previously unexamined datasets. Specifically, we discover novel shortcuts in `BaShapes` (Ying et al., 2019) and `BaMultiShapes` (Azzolin et al., 2023), where the task can be solved by averaging node degrees rather than using the intended motif. Additionally, on molecular benchmarks with explicit motif rules like `Benzene`, `AlkaneCarbonyl`, and `FluorideCarbonyl` (Sanchez-Lengeling et al., 2020), we observe that GNNs consistently fail to learn the correct patterns exactly, resorting instead to spurious correlations. Notably, we argue that this issue cannot be solved by dataset design alone. Faber et al. (2021) and Agarwal et al. (2023) propose handcrafted datasets meant to be more reliable, e.g., by generating unambiguous ground truth and by requiring perfect model accuracy. Yet, they do not offer guarantees that a GNN trained on them implements a uniquely identified behaviour, and verifying a posteriori that the model relies on the intended pattern may itself be as hard as explaining the model. Hence, rather than devising better datasets, we provide guarantees on the model itself in Section 4.

Evaluating the faithfulness of explanations. An orthogonal and widely used (c.f. Fig. 1) approach to evaluating explanation quality is faithfulness, assessing how well an explanation reflects the model

behaviour (Agarwal et al., 2024). Nonetheless, faithfulness comes with its own limitations: there is no universal definition, available metrics often conflict (Azzolin et al., 2025a; Miró-Nicolau et al., 2025), and these metrics are frequently validated by their correlation with human-defined explanations (Fang et al., 2023; Zheng et al., 2023) – the same issue that plausibility suffers from. Despite these limitations, faithfulness remains the only viable option to assess a given explanation in downstream applications, and the evaluation setting we will propose in Section 5 offers a reliable testbed to validate faithfulness metrics as well, which we leave to future work.

In the remainder of the paper, we address the issue of plausibility by building GNNs whose behaviour is known by construction. This is achieved by replacing GNN training with *compilation*.

4 From logic formulas to ground truth explanations

4.1 Gracr: Compiling a logic formula into a GNN

Rather than hoping that training has induced the intended behaviour, we construct GNNs whose behaviour can be controlled exactly. We obtain it by *compiling*, in the spirit of Lindner et al. (2023), a target classifier expressed in logic directly into the weights of a standard message-passing architecture, so that the two agree on every graph. Our construction follows the constructive expressiveness results of Barceló et al. (2020, Prop. 4.1), which we extend to cover also graph-level tasks.

Preliminaries. We will consider node features drawn from a finite list $\mathcal{C}$ of colours, which act as atomic predicates over nodes: a node u is described by the indicator vector $\mathbf{c}_u \in \{0,1\}^{|\mathcal{C}|}$, and $\mathrm{Col}(u) = 1$ iff $(\mathbf{c}_u)_{\mathrm{Col}} = 1$, with Col any colour drawn from $\mathcal{C}$. We compile classifiers expressible in *graded modal logic* (De Rijke, 2000), the fragment of first-order logic generated by

$$\varphi(u) \;::=\; \top \mid \bot \mid \mathrm{Col}(u) \mid \neg\varphi(u) \mid \varphi(u) \wedge \psi(u) \mid \exists^{\geq N} v.\big(E(u,v) \wedge \varphi(v)\big), \qquad (1)$$

where φ and ψ are themselves graded modal logic formulas defined over a free variable. The last construct holds at u when at least N neighbours satisfy φ, and will be shortened to $\Diamond_{\geq N}\varphi$. We reserve Φ for the classifier we wish to compile, keeping φ and ψ for arbitrary formulas and, later, φ_ℓ for the subformulas of Φ. Throughout, we write $(G,u) \models \varphi$ to say that a formula φ is satisfied at the node u of G. Other connectives, like $\vee$ or $\Diamond_{=N}$, are obtained by manipulations of Eq. (1) as described in Appendix C.2. A node classifier is then a formula $\Phi(u)$ with one free variable, predicting $g(G,u) := \mathbb{1}\{(G,u) \models \Phi\}$ at each node u of G. A graph classifier, instead, closes a node predicate φ with a counting quantifier, $\Phi = \exists^{[\mathrm{lo},\mathrm{hi}]} u.\, \varphi(u)$, and thus asks how many nodes of G satisfy φ, predicting $g(G) := \mathbb{1}\{G \models \Phi\}$. For instance, $\exists^{[N,\inf]}$ holds when at least N of them do.

The GNN architecture. We compile into an aggregate-combine GNN (Barceló et al., 2020) with sum aggregation, per-layer weights, and truncated ReLU activation $\sigma(z) := \min(\max(0,z),1)$,

$$\mathbf{h}_u^{(\ell)} \;=\; \sigma\Big(\mathbf{h}_u^{(\ell-1)} C^{(\ell)} \;+\; \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(\ell-1)} A^{(\ell)} \;+\; b^{(\ell)}\Big), \qquad \ell = 1,\dots,L \qquad (2)$$

with $C^{(\ell)}, A^{(\ell)} \in \mathbb{R}^{d\times d}$ and $b^{(\ell)} \in \mathbb{R}^d$. Compilation is possible for this fragment precisely because every graded modal logic classifier is captured by a GNN (Barceló et al., 2020).

GRAph Neural Network CompileR (Gracr). Let $\mathrm{sub}(\Phi) = [\varphi_1,\dots,\varphi_d]$ be the distinct subformulas of Φ, ordered so that each one precedes those containing it and $\varphi_d = \Phi$, and let $\mathrm{depth}(\varphi)$ denote the number of nested connectives in φ, so that atoms have depth 0 and $\mathrm{depth}(\Phi) = L$. Our construction reserves one coordinate $i \in \{1,\dots,d\}$ of the hidden space to encode the truth value of each subformula φ_i, maintaining the invariant for successive layers, as follows:

$$\big(\mathbf{h}_u^{(\ell)}\big)_i = 1 \iff (G,u) \models \varphi_i, \qquad \text{for every } \mathrm{depth}(\varphi_i) \leq \ell \leq L. \qquad (3)$$

This is achieved via three components. *First,* we initialise $\mathbf{h}_u^{(0)}$ from the node features $\mathbf{c}_u$, setting coordinate i to 1 exactly when φ_i is an atom satisfied at u. *Second,* layer ℓ carries forward the subformulas already settled by setting $C_{ii}^{(\ell)} = 1$ whenever $\mathrm{depth}(\varphi_i) < \ell$, and evaluates each subformula φ_i with $\mathrm{depth}(\varphi_i) = \ell$ according to Table 1, all other entries being zero, with $j,k < i$. *Last,* after L layers a node classifier reads off coordinate d, while a graph classifier $\exists^{[\mathrm{lo},\mathrm{hi}]} u.\, \varphi(u)$ first sum-pools that coordinate into the count $m = |\{u \in \mathcal{V} : (G,u) \models \varphi\}|$, and then thresholds it e.g., $\sigma(m - N + 1)$ for $\exists^{[\mathrm{N},\infty]}$. The correctness of this construction follows from Barceló et al. (2020).

Further implementation details are provided in Appendix C, along with a practical example of a compiled GNN and its instantiation using our Python codebase in Appendix C.1. Taken together, the construction above allows us to build a GNN that encodes a graded modal logic statement. This allows practitioners to have full knowledge of the inner mechanisms of the GNN to be explained, constituting a powerful laboratory for research in graph explainability.

Table 1: Weights realising each connective.

Subformula	Compiled weights	
$\varphi_i = \neg\varphi_k$	$C^{(\ell)}_{ki} = -1$	$b^{(\ell)}_i = 1$
$\varphi_i = \varphi_j \wedge \varphi_k$	$C^{(\ell)}_{ji} = C^{(\ell)}_{ki} = 1$	$b^{(\ell)}_i = -1$
$\varphi_i = \Diamond_{\geq N}\varphi_k$	$A^{(\ell)}_{ki} = 1$	$b^{(\ell)}_i = 1 - N$

4.2 Defining ground truth explanations formally

Having obtained a GNN reliably implementing a target formula, the next stage is to define a suitable notion of ground truth explanations, i.e., what we consider as the *desired* explanation. In previous studies, this usually corresponds to the implanted target motif of interest, e.g., the house motif in `Ba2Motif` (Luo et al., 2020). Nonetheless, the case where ground truth explanations are not *unique*[2] is often overlooked, and a precise definition of what a ground truth explanation looks like is often left to task-specific heuristics (Schlichtkrull et al., 2021). In this work, we will build on the formal notion of graph explanation introduced in Azzolin et al. (2025b), described next:

Definition 1. *(Prime Implicant explanation) Let g be a classifier and G be an instance with predicted label $g(G)$, then $R \subseteq G$ is a Prime Implicant explanation (PI) for $g(G)$, written as $PI(g, G)$, if:*

1. *(sufficiency) For all R', such that $R \subseteq R' \subseteq G$, we have that $g(G) = g(R')$.*
2. *(minimality) No other $R' \subset R$ satisfies (1).*

PIs are not unique, and we denote the set containing the Union of all PIs (UPI*) for $g(G)$ as $\mathsf{UPI}(g, G) := \cup_i PI(g, G)_i$. We will shorten this to* UPI *when referring to a generic set of all PIs. For node-level tasks, we will write $PI(g, G, u)$ to indicate the PI explanation for $g(G, u)$.*

PIs are a reasonable target for the following reasons: *First,* they are a crisp, rigorous solution for explaining a wide range of models, like logic classifiers and tabular models (Shih et al., 2018; Marques-Silva & Ignatiev, 2022; Marques-Silva, 2023); *Second,* they were shown to encode a more informative and trustworthy notion of explanation for graph tasks (Weinbauer et al., 2025; Azzolin et al., 2026); *Third,* they naturally handle alternative explanations by collecting the set of all possible PIs (Darwiche & Hirth, 2020; 2023); *Last,* they were shown to recover the implanted target motif for motif-based graph tasks (Azzolin et al., 2025b). Then, our definition of ground truth explanation is:

Definition 2. *(Ground truth explanation) Let g be a classifier and G be an instance with predicted label $g(G)$, then the ground truth explanation μ^* for $g(G)$ is $\mathsf{UPI}(g, G)$, such that $\mu^*_e = \mathbb{1}\{e \in \mathsf{UPI}\}$.*

The definition above indicates the edges we consider to be relevant for a given task, and reconciles with existing definitions of feature relevancy from the formal explainability literature for tabular data (Huang et al., 2023). Having established a precise notion of ground truth explanation, we describe in the next section how UPI can be computed in practice.

4.3 Computing UPI for a compiled GNN

Computing PIs is a known computationally hard problem, with a few exceptions (Marques-Silva et al., 2020; Bassan et al., 2026). This complexity arises because for every candidate explanation R, one has to query the black-box model with *every* extension $R \subseteq R' \subseteq G$, and then iterate to find a subset-minimal candidate (cf. Def. 1). Thanks to our compilation step, however, we can avoid the cost of invoking the black-box model by relying on the symbolic representation from which the GNN was derived. Next, we will describe a knowledge compilation algorithm relying on Reduced Ordered Binary Decision Diagram (ROBDD) (Lee, 1959; Bryant, 1992) that allows an efficient computation of UPI. Although this is not a novel technique in itself (Shih et al., 2018; Darwiche & Hirth, 2023), it is to the best of our knowledge the first time it is applied in the context of graph explainability.

[2]E.g., counting tasks like *a node has more red than blue neighbours*: with n red and $m < n$ blue neighbours, any $m + 1$ red edges suffice and no fewer do, giving $\binom{n}{m+1}$ valid explanations.

Algorithm 1 UNIONPI

Require: classifier Φ; graph G
Ensure: $\mu^*_e = \mathbb{1}\{e \in \mathsf{UPI}\}$
1: $f \leftarrow \textsc{Build}(\Phi, G)$
2: $\hat{y} \leftarrow f(G)$
3: $f \leftarrow f$ **if** $\hat{y} = 1$ **else** $\neg f$
4: $f^{\uparrow} \leftarrow \textsc{Extend}(f)$
5: $\mu^* \leftarrow \textsc{Extract}(f^{\uparrow})$
6: **return** μ^*

Algorithm 2 EXTEND(f)

Require: f is an ROBDD
Ensure: $f^{\uparrow}$ maps R to 1 iff R is sufficient (Def. 1)
1: $f^{\uparrow} \leftarrow f$
2: **for** $e = 1, \ldots, |\mathcal{E}|$ **do**
3: $\quad f^{\uparrow} \leftarrow f^{\uparrow} \wedge f^{\uparrow}|_{x_e=1}$
4: **end for**
5: **return** $f^{\uparrow}$

Algorithm 3 EXTRACT$(f^{\uparrow})$

Require: $f^{\uparrow}$ as given by Alg. 2
Ensure: mask $\mu^* \in \{0,1\}^{|\mathcal{E}|}$
1: **for** $e = 1, \ldots, |\mathcal{E}|$ **do**
2: $\quad \kappa \leftarrow f^{\uparrow} \wedge x_e \wedge \neg f^{\uparrow}|_{x_e=0}$
3: $\quad \mu^*_e \leftarrow \mathbb{1}\{\kappa \neq 0\}$
4: **end for**
5: **return** μ^*

At a high level, an ROBDD is a tractable representation of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ over a set of Boolean variables $X = \{x_1, \ldots, x_n\}$. This is achieved by building a directed acyclic graph where nodes represent the Boolean variables, and edges represent Boolean assignments. This graph-based representation of a Boolean function supports efficient operations and queries on the function, including conjoining and disjoining ROBDDs, and this tractability will be critical for efficiently extracting UPI (see Meinel & Theobald (1998) for more on ROBDDs). Due to space constraints, we defer a more detailed introduction to ROBDDs to Appendix D.1.

Symbolic encoding of Boolean formulas over graphs. In our setting, given a target graph G and a Boolean function f representing a logic formula defined over G, we instantiate an ROBDD with one node for each directed edge of G, such that $X = \{x_e\}_{e \in \mathcal{E}}$. Variables are ordered by edge index, and node and edge features are kept constant. A subgraph $R \subseteq G$ is therefore identified with the assignment assigning $x_e = 1$ when $e \in R$ and $x_e = 0$ otherwise, representing a bijection between the $2^{|\mathcal{E}|}$ boolean assignments of X and the $2^{|\mathcal{E}|}$ subgraphs present in G. Thanks to this bijection, ***an*** ROBDD ***compactly encodes a Boolean function over all possible subgraphs of*** G. Hence, we will write the ROBDD as a function over Boolean variables X or subgraphs R interchangeably. We report an example of such a construction in Appendix D.1.

Having established how input graphs are encoded in Boolean variables, the construction of the ROBDD follows standard construction and reduction algorithms, for which we refer the reader to Bryant (1986) and Wegener (2000). We refer to this procedure as BUILD, described in Appendix D.2.

So far, f has been a generic function mapping subgraphs to $\{0,1\}$, hence a binary classifier. In our experiments, f is given by the modal logic sentence from which the GNN to be explained is derived, as discussed in Section 4.1. Consequently, the procedure described next for extracting UPI from f yields a valid ground truth explanation (cf. Def. 2) of the GNN we will explain.

Extracting UPI from an ROBDD. UNIONPI, described in Algorithm 1, presents the procedure for extracting UPI from an input graph G for the classifier encoded by an ROBDD. Once the ROBDD has been constructed, UNIONPI invokes two routines:

EXTEND (Algorithm 2) is responsible for finding those subgraphs that are sufficient (cf. Def. 1), albeit possibly not minimal. It does so by modifying the function f that the ROBDD encodes from

$$f : R \mapsto f(R) \qquad \text{to} \qquad f^{\uparrow} : R \mapsto f(R) \wedge \textstyle\bigwedge_{R \subset R' \subseteq G} f(R') \tag{4}$$

by recursively conjoining $f^{\uparrow}$ with $f^{\uparrow}|_{x_e=1}$, which corresponds to a copy of $f^{\uparrow}$ whose value at R is the value of $f^{\uparrow}$ at $R \cup \{e\}$. After unrolling the recursion, each subgraph R will be conjoined to every possible extension of itself inside G, and the truth value at R will be true iff R satisfies the sufficiency property. We show an example of how this recursion unrolls in Appendix D.3.

EXTRACT (Algorithm 3), instead, takes as input the ROBDD as given by EXTEND and returns a mask μ^* over the edges of G such that $\mu^*_e = \mathbb{1}\{e \in \mathsf{UPI}(f, G)\}$. For each edge, it asks whether there is a sufficient edge set that contains it ($f^{\uparrow} \wedge x_e$) and that stops being sufficient the moment the edge is removed ($\cdots \wedge \neg f^{\uparrow}|_{x_e=0}$). If the resulting formula admits a witness, i.e., $\kappa \neq 0$, then that edge is indispensable to at least one prime implicant, and therefore belongs in UPI. This is formalized next:

Theorem 1. *Let Φ be a classifier and G a graph such that $G \models \Phi$, let $f = \textsc{Build}(\Phi, G)$ be its* ROBDD *encoding, and let $f^{\uparrow}$ be the output of* EXTEND(f) *so that $f^{\uparrow}$ maps a subgraph R to 1 iff R is sufficient. Then, for any edge e in G, it holds that $(f^{\uparrow} \wedge x_e \wedge \neg f^{\uparrow}|_{x_e=0} \neq 0) \iff e \in \mathsf{UPI}(f, G)$.*

This result also applies to negative instances, as detailed in the proof; see Appendix I. Note that ad hoc, logic-specific knowledge compilation approaches are available (Ignatiev et al., 2019; Marques-Silva et al., 2020; Huang et al., 2021), but we wanted a general approach that generalizes easily beyond the logic fragment we are currently considering. For completeness, Appendix D.4 provides a computational complexity analysis, and Appendix D.5 compares the runtime of Algorithm 1 with that of the explainers evaluated next, showing that the runtime of our algorithm is negligible.

5 GracrBench: Benchmarking explainers on compiled GNNs

Having obtained a GNN implementing a target behaviour, we now use compiled models as a testbed for graph explainers. Our pipeline is as follows: we write a formula, compile it into the weights of a GNN using Gracr, generate a dataset of graphs whose label is consistent with the formula, compute the ground truth explanation UPI with Algorithm 1, and compute the plausibility score of explainers against the ground truth. No training is involved, so we are guaranteed the GNN reflects the ground truth we evaluate the explainer against, thus avoiding the confounder identified in Section 3. Formulas and datasets are chosen to highlight specific shortcomings of explainers, as will be detailed later.

Explainers. We benchmark eleven explainers spanning several methodological categories. These include four *gradient-based* attributions: Saliency (Pope et al., 2019), deconvolution (Deconv, Zeiler & Fergus, 2014), guided backpropagation (GuidedBP, Springenberg et al., 2014), and integrated gradients (IG, Sundararajan et al., 2017); three *perturbation-based* methods: GNNExplainer (Ying et al., 2019), GraphMask (Schlichtkrull et al., 2021) and SubgraphX (Yuan et al., 2021); two *surrogate-based* methods: PGMExplainer (Vu & Thai, 2020) and GraphSVX (Duval & Malliaros, 2021); and two *decomposition-based* methods: FlowX (Gui et al., 2024) and LRP (Baldassarre & Azizpour, 2019; Bach et al., 2015). We additionally include a random baseline (Rnd) that assigns each edge in the computational field a score drawn uniformly at random from $\{0, 1\}$.

Computing the plausibility score. We score an explainer by separately reporting the precision and recall of the predicted μ with respect to μ^*. Given that UPI is a union of alternative and individually valid explanations, this choice allows us to distinguish two cases: *i)* an explainer may precisely highlight truly relevant edges but fail to recover alternative explanations, resulting in high precision but low recall; and *ii)* an explainer may recover many relevant edges across alternative explanations while also including irrelevant edges, resulting in high recall but low precision. We provide additional motivation for this choice over alternatives in Appendix E. Operationally, for each target, we sweep over all thresholds induced by the scores, keep the one maximising F_1 between μ and μ^*, and report the corresponding precision and recall. To avoid empty explanations, we only explain positive targets.

5.1 Tasks and Datasets

We consider three graph-classification and three node-classification tasks. For each task, we generate graphs with random backgrounds and implant the target pattern in positive ones, while expressing the task as a logical formula that Gracr converts into a GNN. Details are in Appendix F.

Graph classification. *i)* In positive `RedBlueLeaf` instances, the graph contains a chain of four red nodes, each with exactly two blue leaves. This structure mirrors classical motif-based benchmarks in which the implanted motif constitutes the ground truth. *ii)* In positive `Redundancy` instances, the graph contains a black node with six red neighbours, any one of which is sufficient to satisfy the property. The task therefore admits six alternative PI explanations, while UPI comprises all six corresponding edges. *iii)* In positive `CompetingRedundancy` instances, the graph contains a black node satisfying at least one of three conditions: having one red, two blue, or three green neighbours. The task therefore admits three PI explanations, while UPI again comprises six edges.

Node classification. *i)* `RedBlueLeaf-NC` is a node-level variant of `RedBlueLeaf`. A target node is positive if it is red, has exactly two blue leaves, and is adjacent to another node with the same properties. Consequently, target nodes belonging to the same motif may have different explanations. *ii)* In `Majority1Hop`, a target node is positive if it has strictly more red neighbours than blue neighbours. *iii)* `Majority2Hop` applies this comparison recursively. A target node is positive if it has strictly more red-ish neighbours than blue-ish neighbours, where a node is red-ish according to the `Majority1Hop` rule. This corresponds to two rounds of label propagation (Nerini et al., 2025).

Table 2: **Plausibility on graph classification**, averaged over 5 seeds.

	RedBlueLeaf			Redundancy			CompetingRedundancy		
Explainer	precision	recall	F_1	precision	recall	F_1	precision	recall	F_1
SALIENCY	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	0.04± 0.00	1.00± 0.00	0.09± 0.00
IG	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	0.50± 0.00	0.67± 0.00
DECONV	0.22± 0.00	1.00± 0.00	0.37± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	0.04± 0.00	1.00± 0.00	0.09± 0.00
GUIDEDBP	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	0.04± 0.00	1.00± 0.00	0.09± 0.00
GNNEXPLAINER	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	0.04± 0.00	1.00± 0.00	0.09± 0.00
GRAPHMASK	0.25± 0.03	0.93± 0.15	0.38± 0.02	0.04± 0.00	1.00± 0.00	0.09± 0.00	0.04± 0.00	1.00± 0.00	0.09± 0.00
SUBGRAPHX	0.40± 0.01	0.84± 0.01	0.38± 0.00	0.19± 0.01	0.69± 0.01	0.15± 0.00	0.19± 0.01	0.69± 0.01	0.15± 0.00
PGMEXPLAINER	0.33± 0.16	0.89± 0.19	0.45± 0.15	0.15± 0.17	0.49± 0.16	0.19± 0.16	0.35± 0.16	0.56± 0.08	0.39± 0.15
GRAPHSVX	1.00± 0.00	1.00± 0.00	1.00± 0.00	0.45± 0.05	0.89± 0.07	0.59± 0.06	0.47± 0.03	0.74± 0.16	0.56± 0.06
FLOWX	0.90± 0.03	0.99± 0.01	0.94± 0.01	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	0.97± 0.07	0.98± 0.04
LRP	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	0.04± 0.00	1.00± 0.00	0.09± 0.00
RND	0.23± 0.02	0.93± 0.16	0.37± 0.00	0.06± 0.01	0.73± 0.15	0.10± 0.01	0.06± 0.01	0.73± 0.15	0.10± 0.01

Table 3: **Plausibility on node classification,** averaged over 5 seeds. A '*OOT*' marks an explainer that exceeded the per-explanation time budget (see Appendix H) on every seed; a † marks one whose mean uses only seeds completed within the budget.

	RedBlueLeaf-NC			Majority1Hop			Majority2Hop		
Explainer	precision	recall	F_1	precision	recall	F_1	precision	recall	F_1
SALIENCY	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	0.87± 0.02	0.92± 0.02
IG	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	0.96± 0.01	1.00± 0.00	0.97± 0.01
DECONV	0.87± 0.00	0.58± 0.00	0.69± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	0.94± 0.02	1.00± 0.00	0.96± 0.01
GUIDEDBP	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	0.87± 0.02	0.92± 0.02
GNNEXPLAINER	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	0.96± 0.01	0.99± 0.00	0.97± 0.01
GRAPHMASK	0.01± 0.00	0.92± 0.16	0.01± 0.00	0.00± 0.00	1.00± 0.00	0.00± 0.00	0.02± 0.01	0.78± 0.12	0.03± 0.01
SUBGRAPHX	0.19± 0.00†	0.74± 0.01†	0.28± 0.00†	0.36± 0.00	0.88± 0.01	0.49± 0.01		*OOT*	
PGMEXPLAINER	0.82± 0.02	0.97± 0.01	0.88± 0.01	0.43± 0.01	0.99± 0.00	0.59± 0.01	0.30± 0.02	0.83± 0.01	0.41± 0.02
GRAPHSVX	0.89± 0.00	1.00± 0.00	0.94± 0.00	0.50± 0.00	1.00± 0.00	0.66± 0.00	0.56± 0.02	0.74± 0.02	0.61± 0.02
FLOWX	0.99± 0.00	1.00± 0.00	0.99± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	0.91± 0.01	0.95± 0.01	0.92± 0.01
LRP	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.01	0.89± 0.02	0.93± 0.01
RND	0.19± 0.01	0.49± 0.03	0.24± 0.01	0.33± 0.17	0.81± 0.06	0.41± 0.18	0.22± 0.01	0.51± 0.04	0.29± 0.01

5.2 RESULTS & KEY TAKEAWAYS

The results are reported in Table 2 and Table 3 for graph and node classification, respectively. Results highlight that we can recover previous insights while uncovering new ones, described next.

5.2.1 RECOVERING EXISTING INSIGHTS

GNNEXPLAINER *does not sparsify redundant edges.* Among the well-performing explainers, GNNEXPLAINER is the only one specifically designed to provide sparse explanations, due to its L_1 penalty on explanation scores (Ying et al., 2019). Our results on `Redundancy`, however, reveal the opposite: GNNEXPLAINER achieves perfect recall across a wide range of L_1 strengths (see Fig. 3 in the Appendix), highlighting *all* alternative explanations rather than a sparse subset, a contradiction already observed empirically (Funke et al., 2022) and explained theoretically (He et al., 2025).

DECONV *can highlight irrelevant-by-design edges.* DECONV is the only gradient-based explainer faring poorly on the motif-based `RedBlueLeaf` task. After a careful investigation, we found that DECONV can pathologically assign spurious relevance to irrelevant-by-design edges. In fact, we can vary its precision by recolouring background nodes with a colour that does not constitute a valid explanation, as shown in Fig. 4 in the Appendix. This recovers the finding of Nie et al. (2018) that DECONV can be insensitive to what the model actually computes.

5.2.2 UNCOVERING NEW LIMITATIONS OF EXPLAINERS

Only* FLOWX, GRAPHSVX, *and* IG *are reliable across tasks, while others fluctuate. Reading Tables 2 and 3 jointly, FLOWX, GRAPHSVX, and IG are the only explainers that never fall to chance level; e.g., FLOWX keeps an F1 of at least 0.92 on all six tasks, and IG never scores below 0.96 in precision. Every other explainer is either uniformly poor or strongly task-dependent. We

stress that this distinction is invisible on the motif-based tasks alone – where SALIENCY, LRP, and GNNEXPLAINER perform on par with FLOWX and IG – and that it is only on settings departing from well-known motif-based tasks that the difference emerges, such as the two discussed next.

Most explainers are not robust to the implementation of formulas. A desirable property of explainers is implementation invariance (Sundararajan et al., 2017; Nauta et al., 2023), i.e., explainers should yield the same – or similar thereof – explanations for equivalent models. Since Gracr enables us to compile GNNs with a controllable behaviour, it provides a controlled setting in which to systematically evaluate this property. As shown in Table 2, several explainers that achieve perfect scores on the other tasks perform at chance level on `CompetingRedundancy`. To investigate this discrepancy, we evaluate an alternative formulation of the same task; see Appendix C.5. Only IG, FLOWX, and GRAPHSVX maintain consistent performance across the two formulations, suggesting greater robustness to these implementation changes. The discrepancy is associated with the original implementation of the "$\vee$" operator, which introduces double negations through De Morgan's laws. Similar effects arise from other formula rewritings, as illustrated by `RedBlueLeaf` in Appendix C.5.

Explainers struggle to recover indirect influences. While most explainers achieve perfect UPI recovery on `Majority1Hop`, none attain perfect results on `Majority2Hop`. We first observe that gradient-based explainers, with the exception of DECONV and IG, maintain high precision (> 0.9) at the expense of recall. This is because they consistently miss edges whose removal would increase the number of bluish neighbours, indicating that they fail to capture indirect, multi-hop effects even in simple settings: although removing such edges does not directly change the target nodes number of red neighbours, they can change the state of an intermediate grey node, which indirectly affects the targets prediction. In contrast, on `Majority2Hop` GNNEXPLAINER, DECONV, and FLOWX attain a recall ≥ 0.95 but at the expense of precision, meaning they cover most indirect interactions whilst producing noisier explanations. See Fig. 5 in Appendix for visual examples of these problems. This insight is robust across implementation choices for most explainers; see Section C.3.

6 CONCLUSION

We formalized a common failure mode when evaluating the plausibility of GNN explainers: the model being explained may not behave as expected. First, we showed that several popular datasets used in the literature suffer from this (Section 3). Then, we proposed Gracr, the first compiler translating graded modal logic sentences into the weights of a GNN (Section 4). Gracr yields compiled GNNs whose behaviour is known exactly, constituting a rigorous testbed for evaluating explainers. To define a reliable ground truth, we adopt prime implicants and define our ground truth explanation to be the union of all prime implicants, for which we developed an exact, efficient extraction algorithm. Finally, we proposed GracrBench, which evaluates graph explainers over compiled GNNs based on how well they recover the respective ground truth explanation (Section 5). Results on graph and node classification recover previously known findings while uncovering new ones, demonstrating how GracrBench enables a sound, fine-grained analysis of the behaviour of post-hoc GNN explainers.

Limitations. Compiled GNNs produced by Gracr rely on binary embeddings and noiseless aggregations, and therefore have limitations in terms of realism compared to GNNs trained from data. This is, however, a deliberate design choice allowing us to reason *exactly* about the internal workings of both the model and the explainer, at a level of detail previously not possible. Nonetheless, experiments conducted on compiled models should be interpreted carefully, and evaluations based on them should be intended as a minimum bar rather than a proxy of in-production evaluations. Furthermore, just like for trained models, our results are to be interpreted in conjunction with the model, as different model configurations can yield different results. Nevertheless, GracrBench allows us to rigorously control these configurations – which are otherwise inextricably confounded on trained models – and assess the robustness of explainers across these. Last, our current implementation is restricted to binary classification and categorical node features.

Future work. Since evaluating all available explainers was infeasible due to their large number, we exclusively tested popular instance-level explainers. Nonetheless, our analysis can be extended to include counterfactual and model-level explainers, especially those aiming to explain GNNs by extracting a logic formula; see Appendix A for an overview. Also, we could make compiled GNNs more realistic, e.g., by fine-tuning after compilation. This is challenging, as fine-tuning may invalidate their guarantees, and ensuring that the expected behaviour is preserved is left to future work.

AI USE STATEMENT

In this work, we used generative AI tools for providing feedback on research methodology or experiments, implementing methods, formulating mathematical claims, assisting in the writing of proofs, interpreting results, and helping develop theoretical models or conceptual frameworks. We have not used generative AI tools for generating synthetic data sets, providing critical ingredients for proving mathematical claims, or proposing or refining hypotheses, and assisting with translation, cleaning and reformatting datasets, and supporting qualitative and thematic data analysis are not applicable to this work. Additionally, we used generative AI tools for brainstorming, identifying relevant literature, drafting parts of a research paper, editing a research paper to improve readability, creating or editing software code, creating or modifying scientific figures or images, and the creation of artifacts. We have reviewed all AI-assisted work. Any LLM output for coding and mathematical proofs was always double-checked by at least two authors, while text was reviewed by all of them. Specifically, LLM-generated code used to implement our synthetic data generation scripts was manually verified and tested. Key ideas were always generated by authors, and then checked, validated, and cross-referenced with the help of LLMs. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI in the manuscript.

ETHICS STATEMENT

All authors have read and approved the ICLR Code of Ethics. As for societal consequences, the aim of this work is to shed light on misleading practices in the evaluation of the quality of explainers in the graph post-hoc explainability literature, and to provide a remedy to it via logic-compiled GNNs. It can therefore contribute to the development of more trustworthy evaluation procedures for graph explainability tools.

REPRODUCIBILITY STATEMENT

Implementation details about Gracr are available in Appendix C, while we provide details and further introductory material to ROBDDs in Appendix D. We also report further details about datasets in Appendix F, about our literature review in Appendix G, and details about explainer implementation and explainer hyperparameter tuning in Appendix H. The proofs of our theoretical results are provided in Appendix I. Our source code is available in the supplemental material.

REFERENCES

Chirag Agarwal, Owen Queen, Himabindu Lakkaraju, and Marinka Zitnik. Evaluating explainability for graph neural networks. *Scientific Data*, 10(1):144, 2023.

Chirag Agarwal, Sree Harsha Tanneru, and Himabindu Lakkaraju. Faithfulness vs. plausibility: On the (un)reliability of explanations from large language models, 2024.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

Kenza Amara, Zhitao Ying, Zitao Zhang, Zhichao Han, Yang Zhao, Yinan Shan, Ulrik Brandes, Sebastian Schemm, and Ce Zhang. Graphframex: Towards systematic evaluation of explainability methods for graph neural networks. In *Learning on Graphs Conference*, pp. 44–1. PMLR, 2022.

Burouj Armgaan, Manthan Dalmia, Sourav Medya, and Sayan Ranu. Graphtrail: Translating GNN predictions into human-interpretable logical rules. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=fzlMza6dRZ.

Artur S Avila Garcez and Gerson Zaverucha. The connectionist inductive learning and logic programming system. *Applied Intelligence*, 11(1):59–77, 1999.

Steve Azzolin, Antonio Longa, Pietro Barbiero, Pietro Lio, and Andrea Passerini. Global explainability of gnns via logic combination of learned concepts. In *The Eleventh International Conference on Learning Representations*, 2023.

Steve Azzolin, Antonio Longa, Stefano Teso, and Andrea Passerini. Reconsidering faithfulness in regular, self-explainable and domain invariant GNNs. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=kiOxNsrpQy.

Steve Azzolin, Sagar Malhotra, Andrea Passerini, and Stefano Teso. Beyond topological self-explainable gnns: A formal explainability perspective. In *Forty-second International Conference on Machine Learning*, 2025b.

Steve Azzolin, Stefano Teso, Bruno Lepri, Andrea Passerini, and Sagar Malhotra. GNN explanations that do not explain and how to find them. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=HBcgLe6NZD.

Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

Federico Baldassarre and Hossein Azizpour. Explainability techniques for graph convolutional networks. *arXiv preprint arXiv:1905.13686*, 2019.

Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286 (5439):509–512, 1999.

Pablo Barceló, Egor V Kostylev, Mikael Monet, Jorge Pérez, Juan Reutter, and Juan-Pablo Silva. The logical expressiveness of graph neural networks. In *ICLR*, 2020.

Shahaf Bassan, Ron Eliav, and Shlomit Gur. Explain yourself, briefly! self-explaining neural networks with concise sufficient reasons. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=8nuzsfiQfS.

Shahaf Bassan, Xuanxiang Huang, and Guy Katz. Unifying formal explanations: A complexity-theoretic perspective. In *The Fourteenth International Conference on Learning Representations*, 2026.

Michael Benedikt, Chia-Hsuan Lu, and Tony Tan. Decidability of graph neural networks via logical characterizations. *ACM Transactions on Computational Logic*, 27(2), April 2026. ISSN 1529-3785. doi: 10.1145/3800944. URL https://doi.org/10.1145/3800944.

Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

Randal E Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986. doi: 10.1109/TC.1986.1676819.

Randal E Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, 24(3):293–318, 1992.

Ngoc Bui, Hieu Trung Nguyen, Viet Anh Nguyen, and Rex Ying. Explaining graph neural networks via structure-aware interaction index. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 4854–4883. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/bui24b.html.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in neural information processing systems*, 33: 13260–13271, 2020.

Miles Cranmer, Alvaro Sanchez Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering symbolic models from deep learning with inductive biases. *Advances in neural information processing systems*, 33:17429–17442, 2020.

Adnan Darwiche and Auguste Hirth. On the reasons behind decisions. In *ECAI 2020*, pp. 712–720. IOS Press, 2020.

Adnan Darwiche and Auguste Hirth. On the (complete) reasons behind decisions. *Journal of Logic, Language and Information*, 32(1):63–88, 2023.

Maarten De Rijke. A note on graded modal logic. *Studia Logica*, 64(2):271–283, 2000.

Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34 (2):786–797, 1991.

Alexandre Duval and Fragkiskos D Malliaros. Graphsvx: Shapley value explanations for graph neural networks. In *Joint European conference on machine learning and knowledge discovery in databases*, pp. 302–318. Springer, 2021.

Lukas Faber, Amin K. Moghaddam, and Roger Wattenhofer. When comparing to ground truth is wrong: On evaluating gnn explanation methods. In *KDD*, 2021.

Junfeng Fang, Wei Liu, Yuan Gao, Zemin Liu, An Zhang, Xiang Wang, and Xiangnan He. Evaluating post-hoc explanations for graph neural networks via robustness analysis. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Matthias Fey, Jinu Sunil, Akihiro Nitta, Rishi Puri, Manan Shah, Blaž Stojanovič, Ramona Bendias, Alexandria Barghi, Vid Kocijan, Zecheng Zhang, et al. Pyg 2.0: Scalable learning on real world graphs. *arXiv preprint arXiv:2507.16991*, 2025.

Ioannis Filippidis and contrib. dd: Binary decision diagrams in pure Python and Cython wrappers of CUDD, Sylvan and BuDDy. https://github.com/tulip-control/dd, 2023. Version 0.6.0.

Michele Fontanesi, Alessio Micheli, and Marco Podda. Xai and bias of deep graph networks. ESANN, 2024.

Thorben Funke, Megha Khosla, Mandeep Rathee, and Avishek Anand. Zorro: Valid, sparse, and stable explanations in graph neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 35(8):8687–8698, 2022.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.

Chuqin Geng, Ziyu Zhao, Zhaoyue Wang, Haolin Ye, Yuhe Jiang, and Xujie Si. LogicXGNN: Grounded logical rules for explaining graph neural networks. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=nqZe43tRY9.

Bernardo Cuenca Grau, Eva Feng, and Przemysław Andrzej Wałęga. The correspondence between bounded graph neural networks and fragments of first-order logic. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 19135–19142, 2026.

Shurui Gui, Hao Yuan, Jie Wang, Qicheng Lao, Kang Li, and Shuiwang Ji. Flowx: Towards explainable graph neural networks via message flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(7):4567–4578, 2024. doi: 10.1109/TPAMI.2023.3347470.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Jesse He, Akbar Rafiey, Gal Mishne, and Yusu Wang. Explaining gnn explanations with edge gradients. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 884–895, 2025.

Anna Himmelhuber, Mitchell Joblin, Martin Ringsquandl, and Thomas Runkler. Demystifying graph neural network explanations. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 67–75. Springer, 2021.

Xuanxiang Huang, Yacine Izza, Alexey Ignatiev, Martin C Cooper, Nicholas Asher, and Joao Marques-Silva. Efficient explanations for knowledge compilation languages. *arXiv preprint arXiv:2107.01654*, 2021.

Xuanxiang Huang, Martin C Cooper, António Morgado, Jordi Planes, and Joao Marques-Silva. Feature necessity & relevancy in ml classifier explanations. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 167–186. Springer, 2023.

Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. Abduction-based explanations for machine learning models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 1511–1519, 2019.

Jeroen Kazius, Ross McGuire, and Roberta Bursi. Derivation and validation of toxicophores for mutagenicity prediction. *Journal of medicinal chemistry*, 48(1):312–320, 2005.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Dominik Köhler and Stefan Heindorf. Utilizing description logics for global explanations of heterogeneous graph neural networks. *arXiv preprint arXiv:2405.12654*, 2024.

Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, et al. Captum: A unified and generic model interpretability library for pytorch. *arXiv preprint arXiv:2009.07896*, 2020.

Sebastian Lapuschkin, Stephan Wäldchen, Alexander Binder, Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Unmasking clever hans predictors and assessing what machines really learn. *Nature communications*, 10(1):1–8, 2019.

Chang-Yeong Lee. Representation of switching circuits by binary-decision programs. *The Bell System Technical Journal*, 38(4):985–999, 1959.

David Lindner, János Kramár, Sebastian Farquhar, Matthew Rahtz, Thomas McGrath, and Vladimir Mikulik. Tracr: Compiled transformers as a laboratory for interpretability. In *NeurIPS*, 2023.

Antonio Longa, Steve Azzolin, Gabriele Santin, Giulia Cencetti, Pietro Liò, Bruno Lepri, and Andrea Passerini. Explaining the explainers in graph neural networks: a comparative study. *ACM Computing Surveys*, 2025.

Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. Parameterized explainer for graph neural network. *Advances in neural information processing systems*, 33:19620–19631, 2020.

Joao Marques-Silva. Logic-based explainability in machine learning. In *Reasoning Web. Causality, Explanations and Declarative Knowledge: 18th International Summer School 2022, Berlin, Germany, September 27–30, 2022, Tutorial Lectures*, pp. 24–104. Springer, 2023.

Joao Marques-Silva and Alexey Ignatiev. Delivering trustworthy ai through formal xai. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 12342–12350, 2022.

Joao Marques-Silva, Thomas Gerspacher, Martin C. Cooper, Alexey Ignatiev, and Nina Narodytska. Explaining naive bayes and other linear classifiers with polynomial time and delay. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, 2020. ISBN 9781713829546.

Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VLSI Design: OBDD-foundations and applications*. Springer Science & Business Media, 1998.

Miquel Miró-Nicolau, Antoni Jaume-i Capó, and Gabriel Moyà-Alcover. A comprehensive study on fidelity metrics for xai. *Information Processing & Management*, 62(1):103900, 2025.

Corrado Monti, Paolo Bajardi, Francesco Bonchi, André Panisson, and Alan Perotti. A true-to-the-model axiomatic benchmark for graph-based explainers. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=HSQTv3R8Iz.

Matthew Morris and Ian Horrocks. Sound logical explanations for mean aggregation graph neural networks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=7TY89cqLfE.

Maximilian Muschalik, Fabian Fumagalli, Paolo Frazzetto, Janine Strotherm, Luca Hermes, Alessandro Sperduti, Eyke Hüllermeier, and Barbara E Hammer. Exact computation of any-order shapley interactions for graph neural networks. In *International Conference on Learning Representations*, volume 2025, pp. 36295–36337, 2025.

Meike Nauta, Jan Trienes, Shreyasi Pathak, Elisa Nguyen, Michelle Peters, Yasmin Schmitt, Jörg Schlötterer, Maurice Van Keulen, and Christin Seifert. From anecdotal evidence to quantitative evaluation methods: A systematic review on evaluating explainable ai. *ACM Computing Surveys*, 55(13s):1–42, 2023.

Francesco Paolo Nerini, Francesco Bonchi, and André Panisson. A true-to-the-model benchmark for edge-level attributions of gnn explainers. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 78–93. Springer, 2025.

Weili Nie, Yang Zhang, and Ankit Patel. A theoretical explanation for perplexing behaviors of backpropagation-based visualizations. In *International conference on machine learning*, pp. 3809–3818. Pmlr, 2018.

Pierre Nunn, Marco Sälzer, François Schwarzentruber, and Nicolas Troquard. A logic for reasoning about aggregate-combine graph neural networks. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, IJCAI '24, 2024. ISBN 978-1-956792-04-1. doi: 10.24963/ijcai.2024/391. URL https://doi.org/10.24963/ijcai.2024/391.

Michael Pazzani and Dennis Kibler. The utility of knowledge in inductive learning. *Machine learning*, 9(1):57–94, 1992.

Alexander Pluska, Pascal Welke, Thomas Gärtner, and Sagar Malhotra. Logical distillation of graph neural networks. In *ICML 2024 Workshop on Mechanistic Interpretability*, 2024.

Phillip E Pope, Soheil Kolouri, Mohammad Rostami, Charles E Martin, and Heiko Hoffmann. Explainability methods for graph convolutional neural networks. In *CVPR*, 2019.

Alessio Ragno, Marc Plantevit, and Céline Robardet. Faithful explanations for graph classification using logic. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 113–129. Springer, 2025a.

Alessio Ragno, Marc Plantevit, and Céline Robardet. On logic-based self-explainable graph neural networks. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 116138–116166. Curran Associates, Inc., 2025b. doi: 10.52202/085713-3873. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/a8c893712cb7858e49631fb03c941f8d-Paper-Conference.pdf.

Benjamin Sanchez-Lengeling, Jennifer Wei, Brian Lee, Emily Reif, Peter Wang, Wesley Qian, Kevin McCloskey, Lucy Colwell, and Alexander Wiltschko. Evaluating attribution for graph neural networks. *Advances in neural information processing systems*, 33:5898–5910, 2020.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Michael Sejr Schlichtkrull, Nicola De Cao, and Ivan Titov. Interpreting graph neural networks for {nlp} with differentiable edge masking. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=WznmQa42ZAx.

Andy Shih, Arthur Choi, and Adnan Darwiche. A symbolic approach to explaining bayesian network classifiers. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18. AAAI Press, 2018. ISBN 9780999241127.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pp. 3319–3328. PMLR, 2017.

Juntao Tan, Shijie Geng, Zuohui Fu, Yingqiang Ge, Shuyuan Xu, Yunqi Li, and Yongfeng Zhang. Learning and evaluating graph neural network explanations based on counterfactual and factual reasoning. In *Proceedings of the ACM Web Conference 2022*, WWW '22, pp. 10181027, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450390965. doi: 10.1145/3485447.3511948.

Stefano Teso, Öznur Alkan, Wolfang Stammer, and Elizabeth Daly. Leveraging explanations in interactive machine learning: An overview. *Frontiers in Artificial Intelligence*, 2023.

Geoffrey G. Towell and Jude W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1):119–165, 1994. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(94)90105-8. URL https://www.sciencedirect.com/science/article/pii/0004370294901058.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

Minh Vu and My T Thai. Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. *Advances in neural information processing systems*, 33:12225–12235, 2020.

Xiaoqi Wang and Han Wei Shen. GNNBoundary: Towards explaining graph neural networks through the lens of decision boundaries. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=WIzzXCVYiH.

Ingo Wegener. *Branching programs and binary decision diagrams: theory and applications*. SIAM, 2000.

Klaus Weinbauer, Tieu-Long Phan, Peter F. Stadler, Thomas Gärtner, and Sagar Malhotra. Prime implicant explanations for reaction feasibility prediction. *arXiv preprint arXiv:2510.09226*, 2025.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32, 2019.

Hao Yuan, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji. On explainability of graph neural networks via subgraph explorations. In *International conference on machine learning*, pp. 12241–12252. PMLR, 2021.

Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. Explainability in graph neural networks: A taxonomic survey. *IEEE transactions on pattern analysis and machine intelligence*, 45(5): 5782–5799, 2022.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.

Xu Zheng, Farhad Shirani, Tianchun Wang, Wei Cheng, Zhuomin Chen, Haifeng Chen, Hua Wei, and Dongsheng Luo. Towards robust fidelity for evaluating explainability of graph neural networks, 2023.

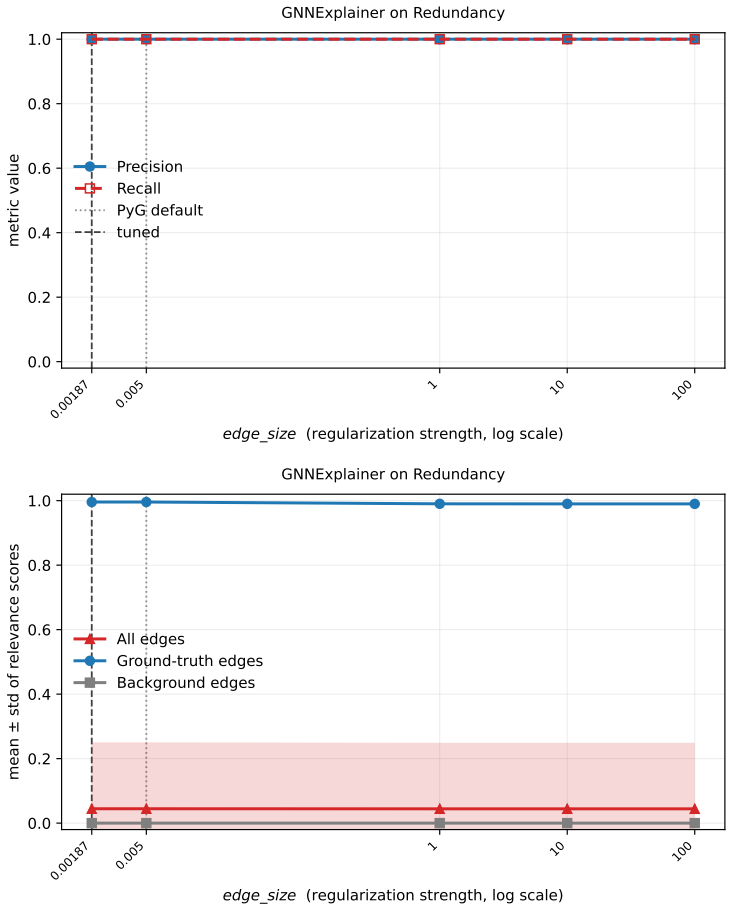


Figure 3: Ablation study on GNNEXPLAINER over `Redundancy`. The plots report: **Top:** metric values for different values of the L_1 penalty regularization term controlling the sparsity strength; across all strength values, GNNEXPLAINER maintains perfect precision and recall. **Bottom:** average edge-wise relevance scores for different values of the L_1 penalty regularization term, confirming that despite the penalty strength getting stronger, GNNEXPLAINER does not sparsify redundant edges.

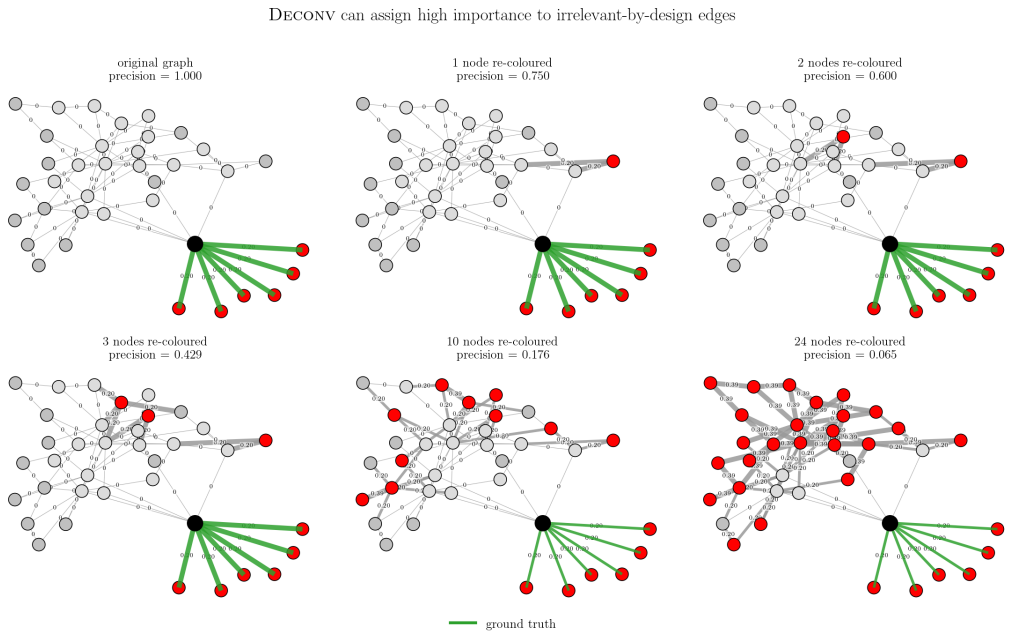


Figure 4: **DECONV can assign high importance to irrelevant-by-design edges**. To show this, we take a graph from `Redundancy` and progressively re-colour background nodes into red. Recall the underlying formula is $\exists u.\text{black}(u) \wedge \Diamond_{\geq 1} \text{red}$. In the original graph, grey-like colours play no role in the formula, and are correctly assigned zero importance. Hence, DECONV achieves perfect precision, as indicated also in Table 2. However, as red nodes are added to the graph, precision steadily drops. This is because red nodes, although they do not contribute to the formula when present alone, do appear in the formula, which is enough for DECONV to highlight them as important. This is the same mechanisms that brings DECONV to perform at chance-level on `RedBlueLeaf`, as the background contains nodes firing intermediate sub-formulas (and intermediate sub-formulas only).

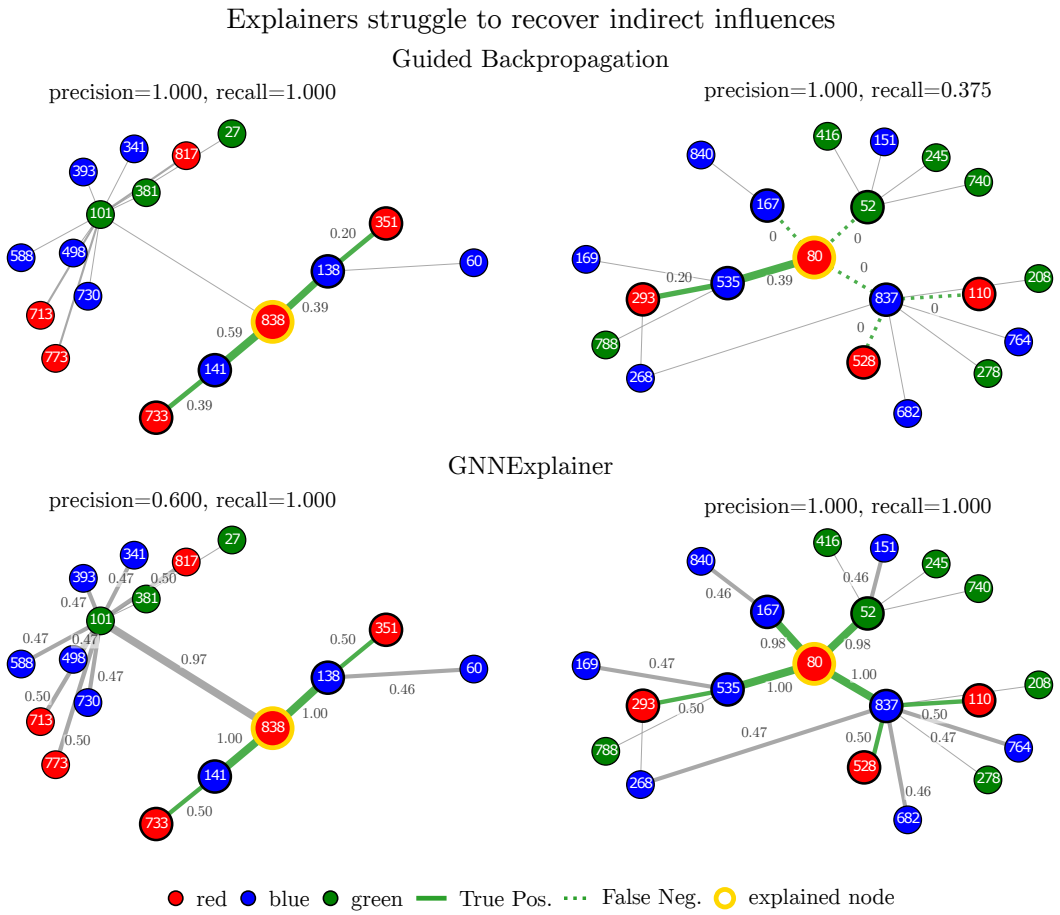


Figure 5: **Explainers struggle to recover indirect influences.** We illustrate this limitation using two representative examples from `Majority2Hop` evaluated across GUIDEDBP and GNNEXPLAINER. `Majority2Hop` requires the target node to have strictly more 1-hop neighbors with a majority of red neighbors than neighbors with a majority of blue neighbors, with node colors reflecting their original features. On the left example graph, GNNEXPLAINER assigns high importance (≥ 0.5) to edges $(101, 713)$, $(101, 817)$, and $(101, 773)$; however, because node 101 has a blue-majority neighborhood, it does not contribute to the positive prediction of the target. On the right example graph, instead, GUIDEDBP assigns zero importance to edges $(80, 167)$, $(80, 52)$, $(80, 837)$, $(110, 837)$, and $(528, 837)$. Yet, these edges ensure that nodes 167, 52, and 837 maintain an equal balance of red and blue neighbors, which prevents a higher number of blue-ish nodes in the neighborhood of node 80: if they were removed, the prediction at target node 80 would flip. In both scenarios, the misattributed edges correspond to messages aggregated two hops away from the target. Notably, the true importance of $(80, 52)$, $(80, 837)$, and $(80, 167)$ is directional (from node 80 outward): while baseline explainers and our UPI algorithm evaluate directed edges, we display undirected attributions here for visual clarity.

A Extended related work

From Logic to Neural Networks. Integrating logical rules into machine learning models has been a long-standing line of research (e.g., Pazzani & Kibler, 1992; Towell & Shavlik, 1994; Avila Garcez & Zaverucha, 1999). In this domain, researchers typically investigated methods to translate input propositional logic rules into neural networks, which could subsequently be refined through backpropagation. More recently, starting with the work of Barceló et al. (2020), the equivalence between formal logic and GNNs has been deeply analysed to characterize the expressivity of the latter. Specifically, Barceló et al. (2020) established a fundamental correspondence between Aggregate-Combine GNNs and graded modal logic. Nunn et al. (2024) extended these results by proposing the logic $K^{\#}$, an expansion of graded modal logic that accounts for linear combinations of counting modalities, enabling bidirectional translation between specific GNNs and logical formulas. These results were later expanded by Grau et al. (2026) and Benedikt et al. (2026): the former systematically maps different classes of bounded GNNs to corresponding fragments of modal and first-order logic, whereas the latter broadens the class of GNNs translatable into logical formulas to analyse their expressiveness and decidability. Closest to our work are Monti et al. (2024) and Nerini et al. (2025), which likewise replace trained black boxes with white-box GNNs of known logic. However, unlike our approach, they do not support general graded modal logic formulas. Monti et al. (2024) evaluates only node features, leaving edge importance entirely undefined. While Nerini et al. (2025) does target edges using a fixed label-propagation GNN (whose tasks we reuse as `Majority1Hop` and `Majority2Hop`), their manual derivation of the ground-truth edges is flawed, as it crucially misses edges responsible for "indirect influences" in the classification. In contrast, Gracr compiles *any* graded modal logic formula and automatically extracts the exact, complete UPI (Algorithm 1) for both node and graph tasks.

From GNN to Logic. A growing line of work explains GNNs post hoc with logic formulas rather than subgraphs. Global explainers fit Boolean formulas over learned or mined concepts to the predictions of a trained model (Azzolin et al., 2023; Armgaan et al., 2024; Geng et al., 2026; Ragno et al., 2025a), or learn description logic class expressions for heterogeneous graphs (Köhler & Heindorf, 2024). Similarly, Pluska et al. (2024) distil GNNs into decision trees, and Morris & Horrocks (2025) extract rules that are provably sound, at the price of restricting the GNN to non-negative weights. A complementary line of work designs self-explainable GNN architectures that expose logic rules by construction rather than post hoc (Cranmer et al., 2020; Azzolin et al., 2025b; Ragno et al., 2025b). Rather than extracting a logic formula out of a GNN, we go in the opposite direction: the formula is fixed first and compiled into the GNN, so the two coincide by construction. Hence, GracrBench offers a reliable testbed for testing also logic-based explainability methods, which we plan to investigate in future work.

B Analyses of popular datasets for evaluating plausibility

We provide an extended analysis of Section 3. In short, for the dataset below, we investigate whether alternative solutions exist, or whether training a GNN on the dataset may yield a model that does not reliably implement the intended ground truth.[3] Since all of these datasets require identifying the correct motifs, the alignment between models and the intended ground truth is only possible if the models achieve nearly perfect accuracy, as discussed in Faber et al. (2021). For datasets already covered in Section 3, we only provide the experimental details and additional results. In total, our analysis covers 74 out of 94 of the plausibility-related papers we considered.

B.1 Synthetic datasets

We start our analysis by looking at synthetic motif datasets. For all of them, we explicitly check the presence of a "shortcut", i.e., if there exists a valid alternative explanation $\bar{\mu}^*$ to the assumed ground truth μ^*, invalidating Assumption 1. On these datasets, we use GIN (Xu et al., 2018), as a provably powerful GNN.

[3] By ground truth here we mean the expected ground truth that the authors of these datasets use to evaluate plausibility in their setting.

We observe that some of these "shortcuts" are already known in the literature: Fontanesi et al. (2024) noticed that the datasets `Ba2Motif` (Luo et al., 2020), `Grid` (Longa et al., 2025), and `GridHouse` (Longa et al., 2025) all contain alternative ground truth explanations that can be used to solve the task. We do not further comment on these datasets, and focus on novel shortcuts instead.

`BaShapes` (Ying et al., 2019). As described in the main text, it is possible to obtain an accuracy of 100% from a 2-layer GIN (Xu et al., 2018) with 64 hidden dimensions, trained for 1000 epochs with an Adam optimizer (learning rate of 0.001). Each GIN layer has an MLP of 2 layers, and between layers we use ReLU activations. This suggests that an alternative ground truth explanation is present in the data, as the original implanted house-like motif requires 3 hops in order to cover all of its edges. In fact, the shortcut can be identified by plotting the degree distribution of each node (Fig. 6, left panel) and of its neighbours (Fig. 6, right panel), showing that all classes are perfectly separable by node degrees without the need to learn the house-like motif.

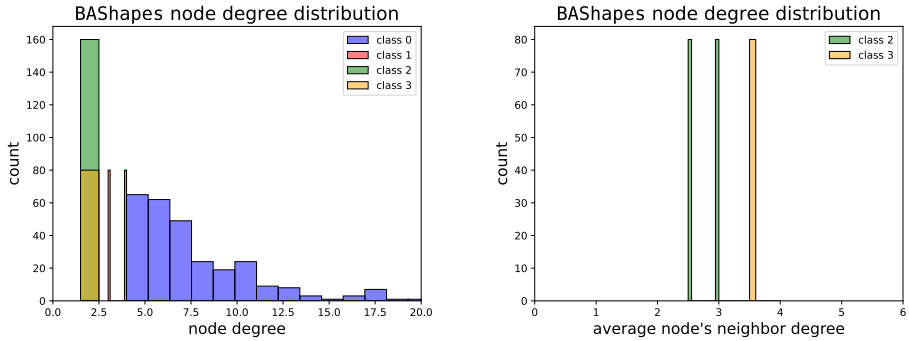


Figure 6: **Left:** Degree distribution for each node class in `BaShapes`. Classes 0 and 1 are fully distinguishable, whereas classes 2 and 3 overlap. **Right:** Distribution of average neighbour degrees for classes 2 and 3, showing that 2-hop degree statistics cleanly separate the remaining two classes.

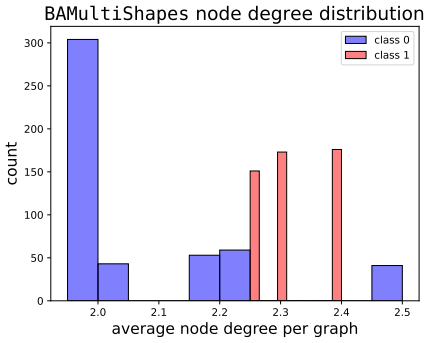


Figure 7: Distribution of average node degree in `BaMultiShapes`, showing a clear separation between graphs in class 0 and class 1.

`BaMultiShapes` (Azzolin et al., 2023). Similarly to `BaShapes`, we notice that it is possible to separate the two classes perfectly just by computing the average node degree of graphs, plotted in Fig. 7. In fact, training a 1-layer GIN with a 2-layer MLP for 200 epochs can reach 100% test accuracy. The GIN model was trained with a learning rate of 0.001, mean global readout, 128 hidden size, and an MLP classifier head. This, again, indicates that the task can be solved by learning an alternative ground truth explanation to the one originally intended, providing a counterexample to Assumption 1.

`Tree-Cycles` (Ying et al., 2019) is a node classification dataset where the task is to distinguish between nodes belonging to a 6-cycle (the ground truth explanation) and those in the background tree. To verify whether GNNs must learn the intended ground truth explanation, we trained a GIN (Xu et al., 2018) for a different number of layers. Each model is trained with a 2-layer MLP, a ReLU

activation function, and a linear classifier. Also, each model was trained over 5 random seeds for 1500 epochs under identical conditions, using the Adam optimizer with a learning rate of 0.005. As shown in Table 4, however, no configuration could reach a test accuracy of 100% robustly across random seeds, indicating that it may be hard for GINs to reliably learn the intended motif (Faber et al., 2021). We then analyze the only model to achieve a test accuracy of 100%, although exclusively on few random seeds: a 4-layer GIN. Supposedly, this is the only candidate GIN that may have learned the intended ground truth explanation. Nonetheless, by performing local perturbations in the background tree, we observe a prediction change for a node *inside* the cycle; see Fig. 8 for more details. This indicates that, despite the perfect accuracy, the model is still attending to unexpected information outside of the intended ground truth explanation.

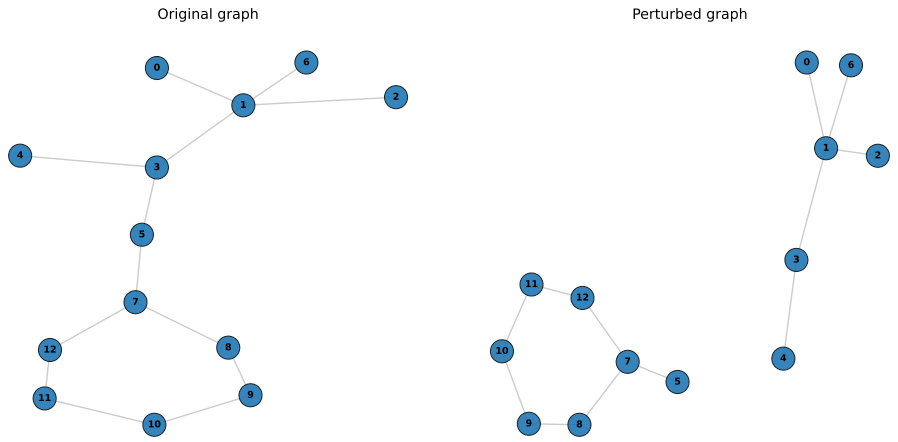


Figure 8: 4-hop neighbourhood of node 7 in `Tree-Cycles`. Original graph has all nodes predicted correctly. However, removing an edge outside of the ground truth explanation yields a prediction change for node 7.

Table 4: Test and validation accuracy averaged across five random seeds for `Tree-Cycles`.

Layers	Test Accuracy	Validation Accuracy
2	0.9720 ± 0.0045	0.9818 ± 0.0045
3	0.9460 ± 0.0141	0.9939 ± 0.0045
4	0.9600 ± 0.0245	0.9859 ± 0.0090

`Tree-Grid` (Ying et al., 2019). Analogous to `Tree-Cycles`, the task requires classifying nodes as belonging either to the background tree or to a 3×3 grid, which serves as the ground-truth motif. Interestingly, 3-layer GINs achieve the best accuracy on this task despite having a receptive field too small to cover the entire ground truth (Table 5), suggesting that the model is learning some shortcut in the node classification. The model was configured identically to `Tree-Cycles` (64 hidden channels, 2-layer MLPs, ReLU activations, linear classifier) and trained across 5 seeds for 1500 epochs with Adam (learning rate$= 0.005$). Crucially, removing edges strictly outside the ground-truth motif causes the model to flip its predictions on some target nodes, providing further evidence of a possible shortcut; see Fig. 9 for more details.

B.2 Molecular Datasets

We consider four widely adopted molecular datasets: `Mutagenicity`, `Benzene`, `AlkaneCarbonyl`, and `FluorideCarbonyl` (Kazius et al., 2005; Sanchez-Lengeling et al.,

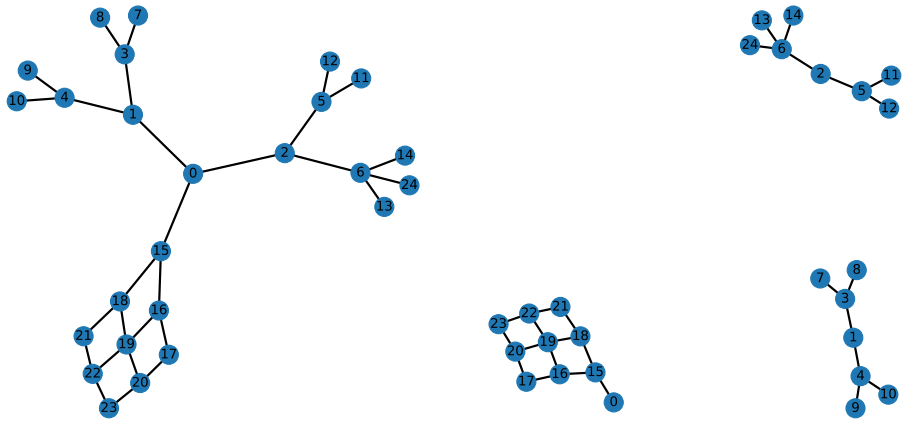


Figure 9: 4-hop neighbourhood of node 15 on `Tree-Grid`. Given a high-accuracy model, all nodes are predicted correctly in the original graph. Yet, removing edges outside of the ground truth explanation (right) yields a prediction change for node 15.

Table 5: Test and validation accuracy averaged across five random seeds for `Tree-Grid`.

Layers	Test Accuracy	Validation Accuracy
2	0.8774 ± 0.0060	0.9073 ± 0.0040
3	0.9774 ± 0.0094	0.9870 ± 0.0040
4	0.9694 ± 0.0236	0.9740 ± 0.0130
5	0.9581 ± 0.0139	0.9707 ± 0.0151

2020)[4]. `Mutagenicity` is a graph-classification dataset annotated according to the mutagenic effect of several organic molecules. Previous studies have identified the NO_2 and NH_2 functional groups as the ground truth explanation μ^* (Ying et al., 2019; Luo et al., 2020)[5]. `Benzene`, `FluorideCarbonyl`, and `AlkaneCarbonyl`, instead, are datasets where the task is to classify graphs based on the presence of specific chemical substructures that serve as the ground-truth motifs defining the positive class: a six-carbon aromatic ring for `Benzene`; a fluoro group (F) paired with a carbonyl group (C=O) for `FluorideCarbonyl`; and a carbonyl group alongside a three-carbon single-bonded chain for `AlkaneCarbonyl`.

For each dataset, we test whether it is possible to train a model completely aligned with the intended ground truth. If that was the case, the model would never fail in the task of recognising the motif. Therefore, we check that it obtains near-perfect accuracy (≥ 0.99) on the task at hand, a test previously justified in Faber et al. (2021). For all datasets, we performed a grid search over five backbones (GCN (Kipf & Welling, 2016), GIN (Xu et al., 2018), GraphSAGE (Hamilton et al., 2017), PNA (Corso et al., 2020), and GAT (Veličković et al., 2018)) across 1 to 4 layers and hidden dimensions in $\{16, 32, 64, 128, 256\}$. Every model employed global add pooling and was optimized with Adam (learning rate $= 10^{-3}$, weight decay $= 10^{-5}$, batch size $= 128$) for up to 500 epochs with cross-entropy loss. We evaluated each configuration using 5-fold cross-validation, selecting the epoch with the lowest validation loss in each fold. Each dataset had both node features (atom type) and edge features (chemical bond type). We used the latter whenever the architecture could use them. We report the average test accuracy of the best configurations for all datasets in Table 6.

[4]For our analysis, we used the dataset implementations provided by GraphXAI (Agarwal et al., 2023), https://github.com/mims-harvard/GraphXAI. Note that in the public repository, the folder names for `FluorideCarbonyl` and `AlkaneCarbonyl` are inverted; we corrected this mismatch in our pipeline.

[5]In the literature, two separate datasets are referred to by this name although they originate from different studies, `MUTAG` (Debnath et al., 1991) and `Mutagenicity` (Kazius et al., 2005). Their primary difference lies in dataset size: `MUTAG` consists of 188 graphs, whereas `Mutagenicity` contains 4337. Since these names are frequently used interchangeably, we specify that all our reported results correspond to the larger dataset.

Table 6: Mean 5-fold cross-validation test performance (accuracy for `Mutagenicity` and `Benzene`; F_1 score for `AlkaneCarbonyl` and `FluorideCarbonyl`) for the best hyperparameter configuration across each architecture and dataset for the additional analysis provided in Appendix B.2.

	Accuracy		F_1 score	
	`Mutagenicity`	`Benzene`	`AlkaneCarbonyl`	`FluorideCarbonyl`
GCN	0.77 ± 0.05	0.85 ± 0.07	0.42 ± 0.31	0.61 ± 0.14
GIN	0.80 ± 0.02	0.91 ± 0.03	0.70 ± 0.17	0.71 ± 0.10
GraphSAGE	0.77 ± 0.06	0.85 ± 0.04	0.43 ± 0.28	0.63 ± 0.12
PNA	0.80 ± 0.02	0.99 ± 0.02	0.71 ± 0.18	0.77 ± 0.11
GAT	0.77 ± 0.04	0.93 ± 0.06	0.49 ± 0.27	0.71 ± 0.09

Almost none of the model-dataset pairs achieve near-perfect accuracy. This suggests that these models are often unable to reliably recognise the ground-truth motifs, indicating that Assumption 1 may not hold; otherwise, the models would be expected to achieve near-perfect discriminative performance. We provide additional discussions on why this does not happen for each of the datasets in the following paragraphs.

`Mutagenicity`. We verified the model behaviour through an additional experiment. We first retrained the best-performing model configurations from scratch under identical settings on a $70\%/15\%/15\%$ train/val/test split; then, we selected all test mutagenic graphs with a NO_2 or NH_2 group and substituted them with a single hydrogen H. This targeted perturbation is specifically designed to generate realistic organic molecules, avoiding out-of-distribution graphs (Zheng et al., 2023). If the mutagenicity of these molecules is exclusively linked to the presence of the NO_2 or NH_2 groups, we expect most perturbed graphs to have a different prediction than the original one. However, as shown in Table 7, more than *half* of the graphs did not yield a prediction change, meaning that the NO_2 or NH_2 groups are not the exclusive factor influencing predictions. Reinforcing this result, Tan et al. (2022) points out that the original `MUTAG` paper (Debnath et al., 1991) explicitly excluded the NH_2 functional group as a cause of mutagenicity, as it is active only under specific conditions. Thus, NH_2 is likely a spurious signal, whereas NO_2 is only one of multiple possible drivers of mutagenicity in the dataset, as shown by our analysis.

Table 7: Percentage of prediction changes on `Mutagenicity` across perturbed graphs for positive, mutagenic graphs containing NO_2 or NH_2 groups.

Model	**Pred. change (%)**
GCN	39%
GIN	43%
GraphSAGE	46%
PNA	37%
GAT	41%

`Benzene`. Our experiments reported in Table 6 show that the GNNs achieving perfect classification is PNA (Corso et al., 2020), which utilizes edge features. However, if all atoms in the datasets were explicitly represented as nodes, aromatic bonds such as those in the benzene rings could be identified purely from node degrees, as aromatic carbons connect to fewer neighbour atoms. Unfortunately, because hydrogen atoms are not explicitly represented in this dataset, graph connectivity alone cannot capture these differences, and aromaticity is encoded via edge features. This directly implies that GNNs that do not process edge features, like GCN (Kipf & Welling, 2016), GIN (Xu et al., 2018), and GraphSAGE (Hamilton et al., 2017), need to rely on alternative signals outside the true ground-truth six-carbon aromatic ring subgraph, increasing their reliance on spurious correlations. Consequently, even an accuracy ≥ 0.90 does not indicate a model aligned with the intended ground truth (Faber et al., 2021). Models without edge features were used with this dataset in the past (Sanchez-Lengeling et al., 2020; Muschalik et al., 2025; Bui et al., 2024; Agarwal et al., 2023), but they could not achieve perfect accuracy.

`FluorideCarbonyl` and `AlkaneCarbonyl`. Although the ground truth motifs are fully discriminative and the datasets contain no label noise, no GNN could reach perfect performance despite hyperparameter optimization (see Table 6).

C Additional details on compiling GNNs from logic

C.1 Practical example of a compiled GNN

In this section, we provide a practical example of how a graded modal logic sentence can be compiled into a GNN, described next.

Example 1. *Let* $\Phi(u) = \mathrm{red}(u) \wedge \Diamond_{\geq 2}\,\mathrm{blue}$*, i.e., a red node with at least two blue neighbours. Its subformulas are* $\mathrm{sub}(\Phi) = [\,\mathrm{red},\ \mathrm{blue},\ \Diamond_{\geq 2}\,\mathrm{blue},\ \Phi\,]$*, of depths* $0, 0, 1, 2$*, so the compiled network has width* $d = 4$ *and* $L = 2$ *layers. Initialisation copies the two colours into coordinates* 1 *and* 2*. Layer* 1 *carries them forward (*$C^{(1)}_{11} = C^{(1)}_{22} = 1$*) and evaluates the diamond (*$A^{(1)}_{23} = 1$*,* $b^{(1)}_{3} = -1$*), while layer* 2 *carries all three forward and evaluates the conjunction (*$C^{(2)}_{14} = C^{(2)}_{34} = 1$*,* $b^{(2)}_{4} = -1$*), all remaining entries being zero. Unrolling Eq. (2) at coordinate* 4 *gives*

$$\big(\mathbf{h}_u^{(2)}\big)_4 \;=\; \sigma\Big(\underbrace{\big(\mathbf{h}_u^{(0)}\big)_1}_{\mathrm{red}(u)} + \sigma\Big(\sum_{v\in\mathcal{N}(u)} \underbrace{\big(\mathbf{h}_v^{(0)}\big)_2}_{\mathrm{blue}(v)} - 1\Big) \;-\; 1\Big), \tag{5}$$

which equals 1 *exactly when* u *is red and at least two of its neighbours are blue.*

The same classifier is obtained in our library by parsing its string representation and compiling it:

```
from compiler import parse, compile_formula_heterogeneous as compile
from compiler.pyg_module import from_compiled_heterogeneous_pyg as to_pyg
matrices = compile(parse("Col(red) & <2> Col(blue)"))  # {C^(ℓ)}_L, {A^(ℓ)}_L, {b^(ℓ)}_L
gnn = to_pyg(matrices)                                 # torch.nn.Module
```

C.2 Desugaring the derived connectives

Eq. (1) lists only the core operators the compiler writes into weights. Every other connective we use is rewritten into that core before compilation. Propositional connectives are standard, $\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$, $\varphi \to \psi \equiv \neg(\varphi \wedge \neg\psi)$ and $\varphi \leftrightarrow \psi \equiv (\varphi \to \psi) \wedge (\psi \to \varphi)$, and so is the dual modality $\Box\varphi \equiv \neg\Diamond_{\geq 1}\neg\varphi$. The counting modalities are obtained from the graded diamond alone:

$$\begin{aligned}
\Diamond_{\leq n}\varphi &\equiv \neg\,\Diamond_{\geq n+1}\varphi,\\
\Diamond_{=n}\varphi &\equiv \Diamond_{\geq n}\varphi \wedge \neg\,\Diamond_{\geq n+1}\varphi \quad (\text{if } n > 0),\\
\Diamond_{=0}\varphi &\equiv \neg\,\Diamond_{\geq 1}\varphi,\\
\Diamond_{[n,m]}\varphi &\equiv \Diamond_{\geq n}\varphi \wedge \neg\,\Diamond_{\geq m+1}\varphi.
\end{aligned}$$

The comparative operator $\Diamond_{>}(\varphi, \psi)$, reading *"more neighbours satisfy* φ *than* ψ*"*, is the one construct that is not a simple rewriting, since a comparison between two unbounded counts is not expressible in graded modal logic, the fragment of FOC_2 used in Barceló et al. (2020). We therefore fix a bound n on the counts, which in our datasets is the maximum degree, and write

$$\Diamond_{>}(\varphi, \psi) \;\equiv\; \bigvee_{m=1}^{n} \Big(\Diamond_{\geq m}\varphi \wedge \neg\,\Diamond_{\geq m}\psi\Big), \tag{6}$$

i.e., there is a threshold m met by the φ-neighbours and missed by the ψ-neighbours. The variant $\Diamond_{=}(\varphi, \psi)$ replaces the clause in parenthesis with $\Diamond_{=m}\varphi \wedge \Diamond_{=m}\psi$ and starts the disjuction from $m = 0$. Written with binary connectives, each disjunction costs three layers, since $\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$, so Eq. (6) would compile to a GNN of depth $3n$. We avoid this by compiling flat n-ary conjunction and disjunction gates directly, in a single column each,

$$\bigwedge_{k=1}^{n} \varphi_k :\ C^{(\ell)}_{ki} = 1\ \ \forall k,\ \ b^{(\ell)}_i = -(n-1), \qquad \bigvee_{k=1}^{n} \varphi_k :\ C^{(\ell)}_{ki} = 1\ \ \forall k,\ \ b^{(\ell)}_i = 0, \tag{7}$$

which are correct because the incoming coordinates are Boolean and σ saturates at 1. The comparative operator then compiles to a GNN of depth 4 regardless of n: one layer aggregates all the thresholds $\Diamond_{\geq m}\varphi$ and $\Diamond_{\geq m}\psi$ in parallel, one negates, one forms the clauses, and one disjoins them with the flat gate of Eq. (7). The n thresholds are thus absorbed into the width of a layer rather than the depth of the compiled model. Note that the thresholds can share a single layer because each occupies its own column of $A^{(\ell)}$: these columns all read the same neighbour sum and differ only in their bias entry $b^{(\ell)}_i = 1 - m$.

C.3 Ablation on comparative operator implementation

The comparative operator, as previously mentioned, is not expressible in the original fragment of graded modal logic used in Barceló et al. (2020). However, during the writing of this manuscript, we found out that Nunn et al. (2024) extended the results of Barceló et al. (2020) to express comparisons and, more generally, linear combinations of counting modalities. Therefore, we ablated our original implementation (which is exact only when the bound n is equal to or higher than the maximum degree of the graph) with one derived directly from Nunn et al. (2024):

$$\Diamond_{>}(\varphi_j, \psi_k): \; A^{(\ell)}_{ji} = 1, \;\; A^{(\ell)}_{ki} = -1, \;\; b^{(\ell)}_{i} = 0. \tag{8}$$

We report the ablation results in Table 8.

Results & Discussion. Most explainers in Table 8 are unaffected by the change. GNNExplainer shows minor variations on `Majority2Hop`, and while GraphMask improves in recall, its overall F_1 score remains within the margin of noise. These fluctuations are due to the fact that explainers rely on the underlying model: GraphMask initialization depends on the input/output dimensions of each layer; GNNExplainer, instead, is affected by the internal gradients of the model and the output of the layers. Saliency and LRP, instead, are noticeably affected by the change because the alternative implementation introduces negative gradients. Because Saliency returns absolute gradient values, edges with large negative gradients (e.g., from a blue node to the target in `Majority1Hop`) are incorrectly assigned high importance, reducing precision in both tasks. For LRP, the final relevance of an edge is computed as the product of its subsequent edges' relevancies. An edge from a red node to a blue-ish intermediary has a negative gradient for that intermediary; however, because the intermediary itself has a negative gradient with respect to the final target, their product results in a false positive relevance. This compounding effect explains why the performance drops slightly on `Majority2Hop` but remains identical on `Majority1Hop`.

C.4 Keeping the backward pass of a compiled model informative

A compiled network computes with integers, as by Eq. (3) every coordinate is Boolean, and every weight in Table 1 is an integer. Consequently σ is always evaluated on the boundaries of the activations ($z \in \{0, 1\}$) or on a saturated flat ($z \leq -1$ or $z \geq 2$), where the gradient $\sigma'(z)$ is 0, and this may create problems for gradient-based explainers. To avoid issues with meaningless gradients, we therefore replace the derivative of σ with

$$\widetilde{\sigma}'(z) \;:=\; \mathbb{1}\{z > 0\}\,, \tag{9}$$

and leave the forward pass untouched. This is applied for every explainer, except for IG for the reasons described below. Note that Eq. (9) is the derivative of a ReLU rather than of a truncated one, and provides the backward gradient that a conventional architecture would provide at that unit. Eq. (9) is also semantically meaningful, since for every pattern in Table 1 the condition $z > 0$ coincides with the truth of the subformula the unit computes: $z = 1 - \alpha_k$ for $\neg\varphi_k$, $z = \alpha_j + \alpha_k - 1$ for $\varphi_j \wedge \varphi_k$, $z = \nu - N + 1$ for $\Diamond_{\geq N}\varphi_k$, and $z = \alpha_k$ for a carried-forward unit, where α_k is the incoming value of coordinate k and ν the number of neighbours satisfying φ_k. Gradient therefore flows along the subformulas that are satisfied.

IG is evaluated with the exact gradient. IG is the only explainer for which we keep the unmodified derivative of σ. This is because, otherwise, the change may violate the Completeness axiom (Sundararajan et al., 2017), as attributions may not sum up to the difference in the output between the original samples and the baseline of IG. Our ablation study in Table 9 confirms that IG is nonetheless robust to this choice: the two backward passes yield identical scores on five tasks out of six, and on `Majority2Hop` the surrogate gradient gains 0.04 precision in exchange for 0.13 in recall, the final F_1 being 0.92 instead of the original 0.97 reported in Table 2.

C.5 Ablating how the formula is written

In standard training, we cannot control how a model internally parametrizes a learned function. Therefore, when functionally equivalent models exist for a given task, a reliable explainer should ideally perform consistently regardless of the underlying architecture (Sundararajan et al., 2017; Nauta et al., 2023). Because Gracr models are compiled rather than trained, we have full control

Table 8: **Plausibility under the ablation of the comparative operator**, averaged over 5 seeds. The original columns compile $\Diamond_{>}(\varphi, \psi)$ as the capped disjunction of Eq. (6), the alternative ones as a single aggregation layer defined in Eq. (8). The graphs, the labels, and the UPI are identical in both cases and equal those of Table 3; only the compiled GNN changes, from 4 to 1 layers on `Majority1Hop` and from 8 to 2 on `Majority2Hop`, and from 203 to 3 and 505 to 5 hidden dimensions respectively. A 'OOT' marks an explainer that exceeded the per-explanation time budget on every seed.

	Original ($\Diamond_{>}$ expanded)			Alternative ($\Diamond_{>}$ direct)		
Explainer	precision	recall	F_1	precision	recall	F_1
			`Majority1Hop`			
SALIENCY	1.00± 0.00	1.00± 0.00	1.00± 0.00	0.90± 0.01	1.00± 0.00	0.94± 0.00
IG	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00
DECONV	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00
GUIDEDBP	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00
GNNEXPLAINER	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00
GRAPHMASK	0.00± 0.00	1.00± 0.00	0.00± 0.00	0.00± 0.00	0.99± 0.01	0.00± 0.00
SUBGRAPHX	0.36± 0.00	0.88± 0.01	0.49± 0.01	0.36± 0.00	0.88± 0.01	0.49± 0.01
PGMEXPLAINER	0.43± 0.01	0.99± 0.00	0.59± 0.01	0.43± 0.01	0.99± 0.00	0.59± 0.01
GRAPHSVX	0.50± 0.00	1.00± 0.00	0.66± 0.00	0.50± 0.00	1.00± 0.00	0.66± 0.00
FLOWX	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00
LRP	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00
RND	0.33± 0.17	0.81± 0.06	0.41± 0.18	0.33± 0.17	0.81± 0.06	0.41± 0.18
			`Majority2Hop`			
SALIENCY	1.00± 0.00	0.87± 0.02	0.92± 0.02	0.67± 0.01	0.87± 0.02	0.73± 0.01
IG	0.96± 0.01	1.00± 0.00	0.97± 0.01	0.96± 0.01	1.00± 0.00	0.97± 0.01
DECONV	0.94± 0.02	1.00± 0.00	0.96± 0.01	0.94± 0.02	1.00± 0.00	0.96± 0.01
GUIDEDBP	1.00± 0.00	0.87± 0.02	0.92± 0.02	1.00± 0.00	0.87± 0.02	0.92± 0.02
GNNEXPLAINER	0.96± 0.01	0.99± 0.00	0.97± 0.01	0.94± 0.02	1.00± 0.00	0.96± 0.01
GRAPHMASK	0.02± 0.01	0.78± 0.12	0.03± 0.01	0.01± 0.00	0.96± 0.06	0.02± 0.00
SUBGRAPHX		OOT			OOT	
PGMEXPLAINER	0.30± 0.02	0.83± 0.01	0.41± 0.02	0.30± 0.02	0.83± 0.01	0.41± 0.02
GRAPHSVX	0.56± 0.02	0.74± 0.02	0.61± 0.02	0.56± 0.02	0.74± 0.02	0.61± 0.02
FLOWX	0.91± 0.01	0.95± 0.01	0.92± 0.01	0.91± 0.01	0.95± 0.01	0.92± 0.01
LRP	1.00± 0.01	0.89± 0.02	0.93± 0.01	0.96± 0.01	0.87± 0.02	0.90± 0.02
RND	0.22± 0.01	0.51± 0.04	0.29± 0.01	0.22± 0.01	0.51± 0.04	0.29± 0.01

Table 9: **IG under the surrogate backward pass** of Eq. (9), averaged over 5 seeds. The graphs, the labels, UPI, and the forward pass are identical to those in Table 2 and Table 3.

Task	precision	recall	F_1
`RedBlueLeaf`	1.00± 0.00	1.00± 0.00	1.00± 0.00
`Redundancy`	1.00± 0.00	1.00± 0.00	1.00± 0.00
`CompetingRedundancy`	1.00± 0.00	0.50± 0.00	0.67± 0.00
`RedBlueLeaf-NC`	1.00± 0.00	1.00± 0.00	1.00± 0.00
`Majority1Hop`	1.00± 0.00	1.00± 0.00	1.00± 0.00
`Majority2Hop`	1.00± 0.00	0.87± 0.02	0.92± 0.02

over the underlying implementation. This allows us to turn conditions that are usually inextricably confounded in trained models – such as vanishing gradients, layer saturation, or specific weight configurations – into controlled experiments. In this section, we test the robustness of explainers by observing how they react when they are evaluated on different but logically equivalent compiled GNNs. In the following, we test an alternative choice to compile the disjunction operator for `CompetingRedundancy`, and an alternative logic formula for the `RedBlueLeaf` task.

Alternative disjunction operator for `CompetingRedundancy`. Our original implementation of `CompetingRedundancy` encodes the "$\vee$" operator via De Morgan's laws ($\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$, see Appendix C.2), which requires intermediate subformulas with nested "$\wedge$" and "$\neg$" operations. To evaluate the sensitivity of explainers to alternative compilation choices, we compare the original implementation reported in the main text against an alternative formulation using the n-ary, flat definition of "$\vee$", described in Eq. (7). While input graphs, labels, and UPI remain identical to those

Table 10: **Plausibility on `CompetingRedundancy` over 5 seeds, for two compilations of the same disjunction**, see Appendix C.5. The original columns report the implementation of the disjunction "$\varphi \vee \psi$" with De Morgan's law "$\neg(\neg\varphi \wedge \neg\psi)$". The alternative ones use the flat n-ary gate (Eq. (7)), which puts no negation on any path.

	Original ($\vee$ via De Morgan)			Alternative ($\vee$ flat gate)		
Explainer	precision	recall	F_1	precision	recall	F_1
SALIENCY	0.04± 0.00	1.00± 0.00	0.09± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00
IG	1.00± 0.00	0.50± 0.00	0.67± 0.00	1.00± 0.00	0.50± 0.00	0.67± 0.00
DECONV	0.04± 0.00	1.00± 0.00	0.09± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00
GUIDEDBP	0.04± 0.00	1.00± 0.00	0.09± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00
GNNEXPLAINER	0.04± 0.00	1.00± 0.00	0.09± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00
GRAPHMASK	0.04± 0.00	1.00± 0.00	0.09± 0.00	0.05± 0.00	1.00± 0.00	0.09± 0.01
SUBGRAPHX	0.19± 0.01	0.69± 0.01	0.15± 0.00	0.18± 0.01	0.72± 0.02	0.14± 0.00
PGMEXPLAINER	0.35± 0.16	0.56± 0.08	0.39± 0.15	0.25± 0.18	0.49± 0.35	0.25± 0.18
GRAPHSVX	0.47± 0.03	0.74± 0.16	0.56± 0.06	0.47± 0.03	0.74± 0.16	0.56± 0.06
FLOWX	1.00± 0.00	0.97± 0.07	0.98± 0.04	1.00± 0.00	0.97± 0.07	0.98± 0.04
LRP	0.04± 0.00	1.00± 0.00	0.09± 0.00	1.00± 0.00	1.00± 0.00	1.00± 0.00
RND	0.06± 0.01	0.73± 0.15	0.10± 0.01	0.06± 0.01	0.73± 0.15	0.10± 0.01

in Table 2, this alternative compilation shrinks the GNN from 8 to 3 layers and from 16 to 9 hidden dimensions, as it does not require nested "$\wedge$" and "$\neg$" operations.

As Table 10 shows, the two implementations yield drastically different results: five explainers (SALIENCY, DECONV, GUIDEDBP, GNNEXPLAINER, and LRP) jump from random-level to perfect performance. This is because, by Eq. (9), the gradient can only travel along satisfied subformulas, while a negation is satisfied exactly when its argument is not, and the two are never satisfied together. Crucially, De Morgan's disjunction puts a negation on every path from the output to its branches, so no gradient ever reaches them.[6] Conversely, the flat gate disjunction of Eq. (7) contains no negation at all, and explainers may benefit from this. Only IG, FLOWX, and GRAPHSVX maintain consistent results across both formulations, indicating a more robust behaviour.

Alternative formula for `RedBlueLeaf`. To further investigate how the syntactical structure of a formula impacts explanations, we compare two distinct compiled models for the `RedBlueLeaf` task (see Appendix F). In the original formulation reported in the main text, the node predicate is defined simply as:

$$\varphi_{\wedge}(u) \;=\; \mathrm{midend}(u) \wedge \Diamond_{\geq 1}\, \mathrm{midend}, \qquad \Phi \;=\; \exists u.\, \varphi_{\wedge}(u), \tag{10}$$

which fires on the two interior nodes of the chain; see Appendix F for the full formula. We compare it against a second, more complex predicate for the same motif, built from the same vocabulary:

$$\begin{aligned}
\mathrm{leaf}(u) &= \mathrm{blue}(u) \wedge \Diamond_{=1} \top && \text{a blue node of degree one,}\\
\mathrm{core}(u) &= \mathrm{red}(u) \wedge \Diamond_{=2}\, \mathrm{leaf} && \text{a red node with exactly two blue leaves,}\\
\mathrm{end}(u) &= \mathrm{core}(u) \wedge \Diamond_{=1}\, \mathrm{core} && \text{endpoint of the chain,}\\
\mathrm{mid}(u) &= \mathrm{core}(u) \wedge \Diamond_{=2}\, \mathrm{core} && \text{interior node of the chain,}\\
\mathrm{midend}(u) &= \mathrm{mid}(u) \wedge \Diamond_{\geq 1}\, \mathrm{end} && \text{interior node touching an endpoint,}\\
\mathrm{midmid}(u) &= \mathrm{mid}(u) \wedge \Diamond_{\geq 1}\, \mathrm{midend} && \text{interior node beside such an interior node,}\\
\varphi_{\mathrm{end}}(u) &= \mathrm{end}(u) \wedge \Diamond_{\geq 1}\, \mathrm{midmid} && \text{the two endpoints, along the whole chain,}\\
\varphi_{\mathrm{int}}(u) &= \mathrm{mid}(u) \wedge \Diamond_{\geq 1}\, \mathrm{end} \wedge \Diamond_{\geq 1}\, \mathrm{midend} && \text{the two interior nodes}
\end{aligned}$$

and closed with the same graph-level quantifier,

$$\varphi_{\vee}(u) \;=\; \varphi_{\mathrm{end}}(u) \vee \varphi_{\mathrm{int}}(u), \qquad \Phi \;=\; \exists u.\, \varphi_{\vee}(u), \tag{11}$$

[6] The same absence of signal would arise with the exact derivative of σ in Eq. (2), but for both compilations alike, since for positive instances of `CompetingRedundancy` all three colour-group clauses hold and therefore gates saturate, yielding zero gradient.

Table 11: **Plausibility on `RedBlueLeaf` over 5 seeds, for two different formulas**, see Appendix C.5. The graphs, the labels, and UPI are identical for both formulas and equal to those evaluated in Table 2. Only the compiled GNN changes. Given that the alternative, more complex formula yields a GNN of greater depth, execution is slower and FLOWX exceeds the time budget.

	Original formula ($\varphi_\wedge$)			Alternative formula ($\varphi_\vee$)		
Explainer	precision	recall	F_1	precision	recall	F_1
SALIENCY	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$0.22_{\pm 0.00}$	$1.00_{\pm 0.00}$	$0.37_{\pm 0.00}$
IG	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$
DECONV	$0.22_{\pm 0.00}$	$1.00_{\pm 0.00}$	$0.37_{\pm 0.00}$	$0.22_{\pm 0.00}$	$1.00_{\pm 0.00}$	$0.37_{\pm 0.00}$
GUIDEDBP	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$0.22_{\pm 0.00}$	$1.00_{\pm 0.00}$	$0.37_{\pm 0.00}$
GNNEXPLAINER	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$
GRAPHMASK	$0.25_{\pm 0.03}$	$0.93_{\pm 0.15}$	$0.38_{\pm 0.02}$	$0.25_{\pm 0.03}$	$0.89_{\pm 0.19}$	$0.38_{\pm 0.02}$
SUBGRAPHX	$0.40_{\pm 0.01}$	$0.84_{\pm 0.01}$	$0.38_{\pm 0.00}$	$0.40_{\pm 0.01}$	$0.84_{\pm 0.01}$	$0.38_{\pm 0.00}$
PGMEXPLAINER	$0.33_{\pm 0.16}$	$0.89_{\pm 0.19}$	$0.45_{\pm 0.15}$	$0.33_{\pm 0.16}$	$0.89_{\pm 0.19}$	$0.45_{\pm 0.15}$
GRAPHSVX	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$
FLOWX	$0.90_{\pm 0.03}$	$0.99_{\pm 0.01}$	$0.94_{\pm 0.01}$		OOT	
LRP	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$1.00_{\pm 0.00}$	$0.22_{\pm 0.00}$	$1.00_{\pm 0.00}$	$0.37_{\pm 0.00}$
RND	$0.23_{\pm 0.02}$	$0.93_{\pm 0.16}$	$0.37_{\pm 0.00}$	$0.23_{\pm 0.02}$	$0.93_{\pm 0.16}$	$0.37_{\pm 0.00}$

which instead fires on all four nodes of the chain. Despite having different depths, the two models agree with the intended label on all graphs of the dataset, and re-deriving the ground truth from either of them with Algorithm 1 returns the same UPI on every graph.

When evaluating this second ablation (Table 11), we observe a phenomenon remarkably similar to the disjunction ablation above. SALIENCY, GUIDEDBP, and LRP drop to chance-level performance under the deeper $\varphi_\vee$ formulation, meaning they are not robust to different model implementations. Conversely, IG, GRAPHSVX, and GNNEXPLAINER remain unaffected. FLOWX, instead, runs out of time due to the deeper model architecture.

Discussion. Taken together, these two ablations highlight a fundamental vulnerability in many popular explainers: simple gradient-based and LRP are highly susceptible to different implementations of the model being explained, meaning they do not fulfill the implementation invariant (Sundararajan et al., 2017; Nauta et al., 2023). While they may achieve perfect scores on a specific formulation (e.g., the flat $\vee$ operator or the shallow $\varphi_\wedge$ predicate), they fail when the same logic is parametrized differently. In contrast, IG and GRAPHSVX remain stable across implementations, indicating more robust behaviours. Because our models are compiled rather than trained, we have full control over what is being explained. This turns conditions – like vanishing gradients – that are usually inextricably confounded on a trained model into controlled experiments, and lets one state and test a specific hypothesis rigorously. For instance, one can test how an explainer copes with saturated activations by testing equivalent models compiled at different depths, or with logically equivalent but syntactically different connectives, while ensuring the same ground truth holds. We leave a systematic ablation over these experimental conditions to future work.

D ADDITIONAL DETAILS ABOUT ROBDDS AND HOW TO EXTRACT UPI

D.1 DEFINITION OF ROBDDS AND A PRACTICAL EXAMPLE

In this section, we aim to provide a compact overview of how the ROBDDs used in Section 4.3 are defined, and then provide a practical example of an ROBDD over graphs. We start by reviewing what ROBDDs are.

Definition 3 (OBDD and ROBDD). *Let $X = \{x_1, \ldots, x_n\}$ be a set of binary variables and π an ordering of the elements of X. An Ordered Binary Decision Diagram (OBDD) over X with respect to π is a rooted, directed acyclic graph, with two sinks called the 1-sink and 0-sink. Every node u except the sinks is labelled with a tuple $(\mathrm{var}(u), \mathrm{lo}(u), \mathrm{hi}(u))$, where $\mathrm{var}(u) \in X$ indicates the node's variable name, and $\mathrm{lo}(u)$ and $\mathrm{hi}(u)$ indicate the two successors of u in the diagram. If there is an edge from a node labelled x_i to a node labelled x_j, then $i <_\pi j$. The OBDD is Reduced (ROBDD) if in addition:*

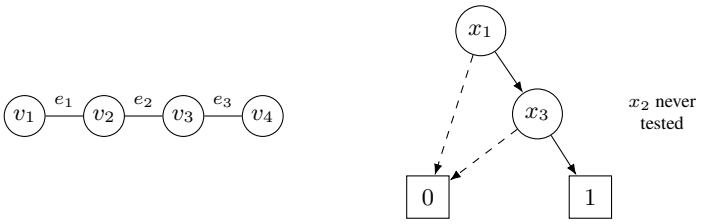


Figure 10: The path graph of Example 2 (left) and the ROBDD of its edge-cover function $f = x_1 \wedge x_3$ under the order $x_1 < x_2 < x_3$ (right). Dashed arcs are lo (the variable is 0, the edge is deleted), solid arcs are hi (the variable is 1, the edge is kept).

(no irrelevant branches) $\mathrm{lo}(u) \neq \mathrm{hi}(u)$ *for every internal node* u*, and*

(no duplicated sub-diagram) no distinct internal nodes carry the same triple $\big(\mathrm{var}(u), \mathrm{lo}(u), \mathrm{hi}(u)\big)$.

At a high level, an ROBDD is a tractable representation of a Boolean function over a set of Boolean variables, allowing efficient operations and queries on the function. In our setting, in particular, we use ROBDDs to efficiently encode a Boolean function – i.e., the graded modal logic formula we wish to convert into a GNN; see Section 4.1 – over subgraphs of G, allowing us to extract the ground truth explanation UPI efficiently. Below, we report an example of an ROBDD defined using the graph symbolic encodings described in Section 4.3. For more on ROBDDs and its applications, see Lee (1959); Bryant (1986; 1992); Meinel & Theobald (1998); Wegener (2000).

Example 2 (Edge cover on a path)**.** *Let G be the path on four nodes, $\mathcal{V} = \{v_1, \ldots, v_4\}$ and $\mathcal{E} = \{e_1, e_2, e_3\}$ with $e_i = (v_i, v_{i+1})$, and let f be the Boolean function holding on R iff R is an edge cover of G, i.e., every node is incident to at least one edge of R. Writing one clause per node,*

$$f = \bigwedge_{v \in \mathcal{V}} \bigvee_{e \ni v} x_e = \underbrace{x_1}_{v_1} \wedge \underbrace{(x_1 \vee x_2)}_{v_2} \wedge \underbrace{(x_2 \vee x_3)}_{v_3} \wedge \underbrace{x_3}_{v_4} = x_1 \wedge x_3, \tag{12}$$

because v_1 and v_4 have degree one, so their clauses are unit and need to be assigned 1 for f to be evaluated to 1, resulting in the simplification above. The resulting ROBDD *for f over G encodes the following Boolean function, and is shown in Fig. 10:*

R	$\varnothing$	$\{e_1\}$	$\{e_2\}$	$\{e_3\}$	$\{e_1, e_2\}$	$\{e_1, e_3\}$	$\{e_2, e_3\}$	$\mathcal{E}$
$f(R)$	0	0	0	0	0	1	0	1

(13)

The possible edge covers are two, but the minimal one is $\{e_1, e_3\}$. Hence, $\mathsf{UPI}(f, G) = \{e_1, e_3\}$.

To keep the example above compact, we considered the undirected edge $e = (u, v)$ as a single variable boolean x_e, but in our implementation we instantiate a single variable per directionality.

D.2 The BUILD algorithm for ROBDDs

In Section 4.3, we introduced the method BUILD for constructing an ROBDD for each formula and graph. In this section, we discuss this procedure in more detail. In practice, we rely on the dd Python package (Filippidis & contrib., 2023) for ROBDD construction and inference.

Four fundamental operations are used to construct ROBDDs (Bryant, 1986): *Reduce*, *Apply*, *Restrict*, and *Composition*. Any other operation on ROBDDs can be expressed as a combination of these procedures (Bryant, 1986). *Reduce* transforms an arbitrary Binary Decision Diagram (BDD) into a canonical ROBDD by eliminating duplicate subgraphs and redundant tests. Given two Boolean functions f and h implemented as ROBDDs and a binary Boolean operator $\langle op \rangle$, *Apply* generates a BDD implementing $f \langle op \rangle h$ (and can also be used to implement logical negation and induction, as detailed in Bryant (1986)). *Restrict* simplifies an input BDD into another BDD by fixing a specific variable x to a constant value b. Finally, *Composition* replaces a variable x in a ROBDD f with another function implemented by a ROBDD h. Because *Apply*, *Restrict*, and *Composition* yield general BDDs, *Reduce* must always be applied post-operation. In our implementation, this workflow is handled automatically by the dd package (Filippidis & contrib., 2023). In our paper, we treat

Boolean operations directly as operations on the ROBDDs themselves, using standard logical notation (e.g., $\wedge, \neg, |_{x=1}$) to implicitly denote the execution of these underlying graph-construction algorithms. We denote the constant true and false functions by 1 and 0, respectively.

There are two variations of the BUILD algorithm: one for node tasks (BUILDNODE in Algorithm 4), and one for graph tasks (BUILDGRAPH in Algorithm 5). In the latter case, we take as input only Φ and the graph G, while in the former we also need the target node v. Since BUILDGRAPH internally uses BUILDNODE, we start our description from BUILDNODE.

BUILDNODE (Algorithm 4) is recursive: when a formula Φ is the conjunction of two subformulas φ_j and φ_k, the procedure recursively constructs the ROBDDs for φ_j and φ_k separately, then conjoins them. Analogous recursive steps handle disjunction and negation. For base cases, i.e., the constants $\top$, $\bot$, and the atomic predicates $\varphi(v) = \mathrm{Col}(v)$, the ROBDD is initialized directly as 1, 0, or the constant $\mathrm{Col}(v)$. The most nuanced step is handling the counting quantifier $\Diamond_{\geq N}\,\psi$, which requires constructing a Boolean function C_N that evaluates to true if and only if at least N neighbours of v satisfy ψ. For each incident edge $e = (v, u)$, the algorithm computes $t = x_e \wedge \textsc{BuildNode}(\psi, u, G)$, indicating edge existence and that u satisfies ψ, and then iteratively updates the intermediate counting functions C_k, $k \in \{1, \ldots, N\}$ (line 20). After evaluating all incident edges, it then returns the counting function C_N, which now indicates whether node v has at least N neighbours satisfying ψ.

BUILDGRAPH (Algorithm 5) builds directly upon Algorithm 4. The graph-level classifier is framed as a counting quantifier $\exists^{[\mathrm{lo},\mathrm{hi}]}$ that verifies whether a node-level formula φ is satisfied by at least lo and at most hi nodes, i.e. it express the formula: $\Phi = \left(\exists^{\geq \mathrm{lo}}\, u.\, \varphi(u)\right) \wedge \neg\left(\exists^{\geq \mathrm{hi}+1}\, u.\, \varphi(u)\right)$. The case where at least N nodes are fired is then defined by setting $\mathrm{lo} = N$ and $\mathrm{hi} = \infty$ (symmetrically, the case where maximum N nodes are fired is defined by setting $\mathrm{lo} = 0$ and $\mathrm{hi} = N$). Therefore, we iterate over each node $v \in \mathcal{V}$, construct its corresponding ROBDD for formula φ via BUILDNODE, and iteratively update the global threshold functions $C_1, \ldots, C_{|\mathcal{V}|}$ (Algorithm 5), analogously to the counting quantifier case of Algorithm 4 (which instead iterates over the edges incident to a node). The graph-level ROBDD is finally computed as $f \leftarrow C_{\mathrm{lo}} \wedge \neg C_{\mathrm{hi}+1}$, which returns true exactly if between lo and hi nodes in the input graph satisfy φ.

In both cases, to speed up the algorithms, we implement a lookup table, where we save each of the ROBDD previously constructed, such that a given function is never implemented twice.

D.3 UNROLLING THE RECURSIVE RULE IN ALGORITHM 2

Consider a generic ROBDD over a graph G, i.e., a Boolean function mapping each subgraph R of G to a truth value, as described in Section 4.3. Algorithm 2 initialises $f^{\uparrow}$ to this function and then updates it in place, executing one conjunction per edge, each time against a copy of $f^{\uparrow}$ shifted by that edge. Writing $f^{\uparrow}$ for the accumulated function and f for the one the ROBDD encodes before the first iteration, unfolding the assignments in order gives, at an arbitrary row R:

$$
\begin{aligned}
\text{start}: \quad & f^{\uparrow}(R) \;=\; f(R) \\
\text{add } e_1: \quad & f^{\uparrow}(R) \;\leftarrow\; f^{\uparrow}(R) \wedge f^{\uparrow}(R \cup \{e_1\}) \;=\; f(R) \wedge f(R \cup \{e_1\}) \\
\text{add } e_2: \quad & f^{\uparrow}(R) \;\leftarrow\; f^{\uparrow}(R) \wedge f^{\uparrow}(R \cup \{e_2\}) \\
& \qquad\qquad = \big[f(R) \wedge f(R \cup \{e_1\})\big] \wedge \big[f(R \cup \{e_2\}) \wedge f(R \cup \{e_1\} \cup \{e_2\})\big] \\
\text{add } e_3: \quad & f^{\uparrow}(R) \;\leftarrow\; f^{\uparrow}(R) \wedge f^{\uparrow}(R \cup \{e_3\}) \\
& \qquad\qquad = \big[f(R) \wedge f(R \cup \{e_1\}) \wedge f(R \cup \{e_2\}) \wedge f(R \cup \{e_1\} \cup \{e_2\})\big] \\
& \qquad\qquad\quad \wedge\; \big[f(R \cup \{e_3\}) \wedge f(R \cup \{e_1\} \cup \{e_3\}) \\
& \qquad\qquad\qquad \wedge f(R \cup \{e_2\} \cup \{e_3\}) \wedge f(R \cup \{e_1\} \cup \{e_2\} \cup \{e_3\})\big] \\
& \ldots
\end{aligned}
$$

Each step conjoins the *accumulated* $f^{\uparrow}$ with a shifted copy of itself, so every term already present is duplicated with the new edge added and the clause doubles. After all $|\mathcal{E}|$ steps, the accumulated

Algorithm 4 BUILDNODE(φ, v, G)

Require: formula φ; node $v \in \mathcal{V}$; graph $G = (\mathcal{V}, \mathcal{E})$;
Require: global variable B indicating an ROBDD object;
Ensure: ROBDD f with $f(R) = \mathbb{1}\{(R, v) \models \varphi\}$ for any $R \subseteq G$

1: **if** $\varphi = \top$ **then**
2: $f \leftarrow 1$
3: **else if** $\varphi = \bot$ **then**
4: $f \leftarrow 0$
5: **else if** $\varphi = \mathrm{Col}$, a colour **then**
6: $f \leftarrow \mathrm{Col}(v)$
7: **else if** $\varphi = \neg\psi$ **then**
8: $f \leftarrow \neg$ BUILDNODE(ψ, v, G)
9: **else if** $\varphi = \bigwedge_i \psi_i$ **then**
10: $f \leftarrow \bigwedge_i$ BUILDNODE(ψ_i, v, G)
11: **else if** $\varphi = \bigvee_i \psi_i$ **then**
12: $f \leftarrow \bigvee_i$ BUILDNODE(ψ_i, v, G)
13: **else if** $\varphi = \Diamond_{\geq N}\, \psi$ **then**
14: $C_0 \leftarrow 1$
15: $C_k \leftarrow 0$ for $k = 1, \ldots, N$
16: **for** $u \in \mathcal{N}(v)$ **do**
17: $e \leftarrow (v, u)$
18: $t \leftarrow x_e \wedge$ BUILDNODE(ψ, u, G)
19: **for** $k = N, \ldots, 1$ **do**
20: $C_k \leftarrow C_k \vee (C_{k-1} \wedge t)$
21: **end for**
22: **end for**
23: $f \leftarrow C_N$
24: **end if**
25: **return** f

Algorithm 5 BUILDGRAPH(Φ, G)

Require: formula $\Phi = \exists^{[\mathrm{lo},\mathrm{hi}]} u.\, \varphi(u)$, with the counting quantifier standing for the readout bounds $\mathrm{lo} \leq \mathrm{hi}$, so that $G \models \Phi$ iff $\mathrm{lo} \leq |\{u : (G, u) \models \varphi\}| \leq \mathrm{hi}$; graph $G = (\mathcal{V}, \mathcal{E})$,
Ensure: ROBDD f with $f(R) = \mathbb{1}\{R \models \Phi\}$ for any $R \subseteq G$

1: $C_0 \leftarrow 1$
2: $C_k \leftarrow 0$ for $k = 1, \ldots, |\mathcal{V}|$
3: **for** $v \in \mathcal{V}$ **do**
4: $f_v \leftarrow$ BUILDNODE(φ, v, G)
5: **for** $k = |\mathcal{V}|, \ldots, 1$ **do**
6: $C_k \leftarrow C_k \vee (C_{k-1} \wedge f_v)$
7: **end for**
8: **end for**
9: $f \leftarrow C_{\mathrm{lo}}$
10: **if** $\mathrm{hi} + 1 \leq |\mathcal{V}|$ **then**
11: $f \leftarrow f \wedge \neg\, C_{\mathrm{hi}+1}$
12: **end if**
13: **return** f

function is $f^{\uparrow}(R) = \bigwedge_{T \subseteq \mathcal{E}} f(R \cup T)$; as T ranges over the subsets of $\mathcal{E}$, the set $R \cup T$ ranges over every superset of R, so $f^{\uparrow}(R) = 1$ exactly when R is sufficient in the sense of Def. 1.

D.4 COMPUTATIONAL COMPLEXITY OF ALGORITHM 1

We proceed to analyse the time complexity of Algorithm 1 in terms of the number of symbolic operations on the ROBDD. Given a graph G and a formula Φ, we first perform the construction of the ROBDD. Consider a node-level formula Φ and a target node v, for which we are building the corresponding ROBDD. While $\wedge$, $\vee$, and $\neg$ all require a single operation on the ROBDD to be built (lines 7-12 of Algorithm 4), the existential quantifier $\Diamond_{\geq N}$ can require up to $N|\mathcal{E}_T|$ operations, with $\mathcal{E}_T$ the receptive field of target node v for formula Φ (lines 13-23). Calling N_{max} the largest N between counting quantifiers in Φ, the upper bound for the computational cost for building a node-level formula is then $O(|\mathcal{E}| \cdot |\Phi| \cdot N_{max})$ symbolic operations on the ROBDD, where $|\Phi| := |\mathrm{sub}(\Phi)|$ is the number of subformulas of Φ, as defined in Section 4.1. For graph-level formulas (Section 4.1), the node counting quantifier $\exists^{[\mathrm{lo},\mathrm{hi}]}$ additionally requires $O(|\mathcal{V}|^2)$ symbolic operations (lines 3-8 of Algorithm 5) to be constructed.

Then, Algorithm 2 requires $O(|\mathcal{E}|)$ symbolic operations to construct the function $f^{\uparrow}$: as shown in Appendix D.3, for each edge in $\mathcal{E}$ we modify $f^{\uparrow}$ iteratively by performing a symbolic operation with the ROBDD ($f^{\uparrow} \leftarrow f^{\uparrow} \wedge f^{\uparrow}|_{x_e=1}$). Algorithm 3 similarly performs $O(|\mathcal{E}|)$ symbolic operations to examine if each $e \in \mathcal{E}$ is also in UPI (for each edge, it always performs a fixed number of symbolic operations on the ROBDD, and then checks its satisfiability, which is an $O(1)$ operation when using ROBDD). The final time complexity, for a single target of a graph classifier, is then $O(|\mathcal{E}| \cdot |\Phi| \cdot N_{max} + |\mathcal{V}|^2)$ symbolic operations.

The real bottleneck of our approach is that the time complexity of a symbolic operation on a ROBDD depends on its size, which is in the worst case exponential in the number of variables (Bryant, 1986) which, in our case, is the number of edges $|\mathcal{E}|$ of the input graph. However, in practice, the tasks and datasets we have chosen rarely fall into this worst case, and the `dd` Python package (Filippidis & contrib., 2023) provides an efficient implementation of the ROBDD.

We complement the analysis with a measurement of the actual cost of Algorithm 1 over random graphs at varying size.

Setup. For a given number of nodes N, we take G to be the complete directed graph on N nodes, self-loops included, so that the number of candidate subgraphs is 2^{N^2}. The classifier is the directed edge-cover function f of Example 2, i.e. $f(R) = 1$ iff every node of G retains at least one incoming edge in R. We pick this family for two main reasons: its ground truth is $\mathsf{UPI}(f, G) = \mathcal{E}$, so no instance degenerates into a trivial ROBDD; and both f and the sufficiency-extended function of Algorithm 2 keep growing with $|\mathcal{E}|$.

We report the wall-clock time of *one* complete run of Algorithm 1 for three ROBDD engines exposed by the `dd` package (Filippidis & contrib., 2023): the Cython-accelerated CUDD backend with a static variable order, the same backend with dynamic variable reordering enabled, and the pure-Python `autoref` backend (no Cython/CUDD acceleration). Alongside each time, we report the number of nodes in the ROBDD to rack memory consumption. The last column is the brute-force baseline, namely enumerating all $2^{|\mathcal{E}|}$ subgraphs of G, querying the compiled GNN on each of them, representing a naive implementation of Def. 1. All measurements are taken on a single core of an AMD Ryzen 9 7950X. An engine exceeding a budget of 1 second on a given size is not run on the larger ones and is marked OOT. All engines are also asserted to return the same edge mask.

Results. As reported in Table 12, the brute-force baseline is already at 48 seconds on $N = 5$ nodes and is out of reach immediately after, whereas Algorithm 1 stays in the millisecond range over the whole range we tested, up to $|\mathcal{E}| = 144$ edges, i.e. 2^{144} subgraphs, with a diagram of a few tens of thousands of nodes. The comparison between the two CUDD columns is also informative: dynamic reordering never pays off here, and costs up to two orders of magnitude at the larger sizes, which indicates that our default ordering π based on the edge index is already good enough. Given the results, we adopt CUDD with a static variable order as the default implementation in all our experiments.

Table 12: Cost of extracting UPI with Algorithm 1 on complete graphs on N nodes for the edge-cover classifier of Example 2, against a brute-force baseline implementing Def. 1. Times are for one full extraction, *nodes* is the number of nodes in the ROBDD, and OOT marks a configuration that was dropped after exceeding the budget of 1 second on a smaller size.

		CUDD, static π		CUDD, dynamic π		`autoref`		brute force
N	N^2	time (s)	nodes	time (s)	nodes	time (s)	nodes	time (s)
3	9	0.006	<1k	0.006	<1k	0.001	<1k	0.004
4	16	0.006	<1k	0.006	<1k	0.001	<1k	0.050
5	25	0.007	1k	0.007	1k	0.003	1k	48.041
6	36	0.002	2k	0.002	2k	0.006	2k	OOT
8	64	0.002	7k	0.022	7k	0.020	7k	OOT
10	100	0.003	18k	0.198	18k	0.044	18k	OOT
12	144	0.006	38k	0.502	36k	0.096	38k	OOT

D.5 Cost of the ground truth against the cost of the explainers

In this section, we compare the execution time of the ground truth extraction in Algorithm 1 with the cost of running explainers. As the results shall show, the total runtime is largely dominated by the explainers.

Setup. We break the wall clock into the three stages of Algorithm 1: the construction of the ROBDD, the sufficiency extension of Algorithm 2, and the final extraction of Algorithm 3. We report average and standard deviations over the first 200 instances of each dataset, on a single core of an AMD Ryzen 9 7950X. For explainers, we report the time that each explainer takes on average to explain a

single explanation. In this analysis, we only report the timing for the most expensive explainers, as the others run at negligible cost.

Results. As reported in Table 13, one extraction takes up to 0.15 seconds across the whole benchmark. Note that this cost is paid once and then the ground truth explanations are stored together with the dataset. For motif-based tasks, such as `RedBlueLeaf`, or for simple dataset designs like `Redundancy` and `CompetingRedundancy`, the algorithm can, in principle, be replaced by manual annotation of the target motif, therefore further reducing the cost. This is possible because the formula is designed to recognize the target motif exclusively, with every relevant edge belonging to that motif by construction. Table 14 reports the numbers for explanation extraction, showing how the overall runtime of our experiments is largely dominated by the explainers.

Table 13: Cost of extracting UPI with Algorithm 1 on the benchmark datasets. Times are wall-clock seconds for one complete extraction on a single explained instance (one graph for graph classification, one target node for node classification), split into the three stages of the algorithm, as mean ± standard deviation over the first 200 explained instances of each dataset. $|\mathcal{V}|$ and $|\mathcal{E}|$ are the size of the graph, and $|\mathcal{E}_T|$ is the average size of the receptive field of the target. The '*Num. nodes*' column indicates the mean number of nodes in the ROBDD, with the maximum reported in parentheses.

					time (s)			
Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{E}_T\rvert$	Num. nodes	BUILD	Algorithm 2	Algorithm 3	total
graph classification								
`RedBlueLeaf`	32	98	98.0	4.6k (155k)	0.0124 ± 0.0185	0.0002 ± 0.0000	0.0003 ± 0.0000	0.0130 ± 0.0185
`Redundancy`	37	134	134.0	338 (338)	0.0009 ± 0.0001	0.0004 ± 0.0000	0.0005 ± 0.0000	0.0018 ± 0.0002
`CompetingRedundancy`	37	134	134.0	342 (342)	0.0022 ± 0.0047	0.0004 ± 0.0000	0.0005 ± 0.0000	0.0030 ± 0.0047
node classification								
`RedBlueLeaf-NC`	750	2192	56.0	14k (1.2M)	0.0041 ± 0.0306	0.0002 ± 0.0003	0.0003 ± 0.0004	0.0045 ± 0.0311
`Majority1Hop`	500	1992	6.0	620 (43k)	0.0057 ± 0.0099	0.0000 ± 0.0003	0.0001 ± 0.0004	0.0058 ± 0.0105
`Majority2Hop`	500	1992	73.7	109k (2.9M)	0.1374 ± 0.1625	0.0062 ± 0.0212	0.0082 ± 0.0235	0.1517 ± 0.2024

Table 14: Wall-clock seconds to explain *one* instance, per explainer and per dataset, averaged over random seeds. A † marks an explainer that did not finish every instance within the per-explanation time-out, so that its mean is computed on the instances that did finish.

	time (s)		
Dataset	GNNEXPLAINER	GRAPHMASK	SUBGRAPHX
graph classification			
`RedBlueLeaf`	0.554 ± 0.380	0.609 ± 0.292	1.523 ± 0.409
`Redundancy`	0.066 ± 0.002	0.019 ± 0.002	42.464 ± 1.846
`CompetingRedundancy`	1.001 ± 0.029	0.023 ± 0.001	44.986 ± 1.890
node classification			
`RedBlueLeaf-NC`	0.323 ± 0.070	0.109 ± 0.012	4.502 ± 12.769†
`Majority1Hop`	0.380 ± 0.027	0.678 ± 0.037	0.104 ± 0.555
`Majority2Hop`	1.203 ± 0.101	4.113 ± 0.206	1904.211 ± 1805.311†

E CHOOSING THE RIGHT EVALUATION METRIC

In this section we motivate our choice to report precision and recall separately, and scoring against the *union* of the PIs rather than against each PI individually. We do this considering previous approaches:

Why not plain Accuracy (Ying et al., 2019). On the tasks we consider, UPI covers a small minority of the edges of a graph (c.f. Table 15). Accuracy would therefore be dominated by the negative class: the constant explanation that declares every edge irrelevant would score very high, while carrying no information whatsoever.

Why not AUROC (Luo et al., 2020) and Precision-Recall Area Under the Curve (PRAUC) (Nerini et al., 2025). These two metrics measure different things, but they share one key pitfall: each compresses the precision/recall trade-off into a single scalar, and because UPI is a union of alternative,

individually valid explanations, that trade-off is informative. Consider an explainer that returns one single PI, correctly and in full. This is alone a desirable, high-quality explanation (Darwiche & Hirth, 2020; Marques-Silva & Ignatiev, 2022; Bassan et al., 2025; Azzolin et al., 2026), as everything it selects is relevant and minimal. Its recall, however, is $|PI|/|\mathsf{UPI}|$, which can be arbitrarily small. Hence, the PRAUC will also be small. Since such an explainer has settled on one explanation, it has no reason to rank the edges of the *other* PIs any higher than the irrelevant ones, and therefore AUROC also drops. In every case, the fact that each and every edge the explainer selected is correct may get overwhelmed by the fact that alternative valid explanations exist when the explainer preferred to focus on just one.

Why not a weighted intersection over union (WIoU) (Azzolin et al., 2025a). A further alternative is to compare the two masks with a weighted intersection over union,

$$\mathrm{WIoU}(\mu, \mu^*) = \frac{\sum_{e \in \mathcal{E}} \min(\mu_e, \mu_e^*)}{\sum_{e \in \mathcal{E}} \max(\mu_e, \mu_e^*)},$$

which has the appeal of requiring no threshold. The problem is that WIoU may charge the explanation for the total score mass placed on irrelevant edges, which however accumulates with the size of the graph even when the ranking of the edges is perfect. To see this, consider a task whose ground truth is a single edge per graph, and an explainer that assigns importance 1.0 to that edge and a negligible 0.001 to all others, i.e., a perfect ordering. On a graph with $|\mathcal{E}| = 10$ we get $\mathrm{WIoU} = 1/(1 + 9 \cdot 0.001) = 0.991$, whereas on a graph with $|\mathcal{E}| = 5000$ we get $1/(1 + 4999 \cdot 0.001) = 0.167$. This shows that the same explainer, with the same ranking, is scored an order of magnitude apart, meaning scores are not comparable across graph size.

Why the union of the PIs, and not the disjoint union (Agarwal et al., 2023). Evaluating against every single PI individually would require retaining one explanatory mask per PI and scoring the explanation against the best matching one, similarly to what is proposed in Agarwal et al. (2023). We avoid this for two reasons: First, the number of PIs can grow exponentially in $|\mathcal{E}|$ so both storing the masks and matching against them may become intractable; Second, a best-matching-PI score is easy to inflate, as the explainer returning a single PI is awarded a perfect score while telling the user nothing about how well the explainer covers all the other alternative explanations.

In summary, our design choice of tracking precision and recall separately allows us to measure: *i)* how well the explainer recovers exactly the relevant elements, via the precision score; *ii)* how well it can cover the range of all possible alternative explanations, via the recall score.

Two additional caveats are of interest: *First,* we note that the F_1 reported in Table 2 and Table 3 does not match the precision and recall columns; this is because the F_1 column is computed as the average of sample-wise precisions and recalls, and the average of F_1 scores does not necessarily match the F_1 of the averages. *Second,* by sweeping over all possible thresholds, we are providing an *optimistic* estimate of precision, recall, and F_1.

F Dataset details

Throughout, $\Diamond_{\geq N}\,\varphi$ reads "at least N neighbours satisfy φ" and $\Diamond_{=N}\,\varphi$ "exactly N of them", $\top$ is the always-true predicate (so $\Diamond_{=1}\top$ is "degree exactly one"), and $\mathrm{Col}(u)$ is true when node u carries colour Col. For graph classification tasks, every task is specified by a node predicate φ together with the graph-level quantifier $\Phi = \exists u.\, \varphi(u)$ that acts as the graph-level readout. We call *palette* the ordered list of colours $\mathcal{C}$ a task uses. It plays two roles at once: it is the vocabulary out of which the colour predicates $\mathrm{Col}(u)$ are drawn, and it is the node feature space, as a node carrying colour Col is encoded by the indicator vector $\mathbf{c}_v$, here one-hot, selecting the position of Col in the palette.

Common protocol. Every dataset contains 1000 graphs with a class ratio of 0.5, built on a Barabási–Albert background with attachment parameter 2 (Barabási & Albert, 1999), of 20 nodes for `RedBlueLeaf` and of 30 nodes for `Redundancy` and `CompetingRedundancy`. Every background node receives a colour drawn uniformly from the colours of the palette that are reserved for the background in the task. At generation time, we apply rejection sampling against φ, so that a candidate graph is kept only when φ fires exactly where the annotation says it should, i.e., only inside the motif for a positive graph, nowhere at all for a negative one. This guarantees that the annotation is unambiguous.

RedBlueLeaf. The palette is $[\text{red}, \text{blue}, \text{green}]$ and the motif is a chain of four red nodes, each carrying exactly two blue leaves, for 12 nodes and 11 edges in total. The motif is attached to the background by linking the two chain endpoints to two random background nodes, which are recoloured green first. This is done because a green anchor is neither a red core nor a blue leaf, so the attachment cannot perturb the local counts of the motif. The two attachment edges are not part of the ground truth, which is exactly the 11 motif edges. If the motif is attached, the graph is annotated as of positive class; otherwise, of negative one. The underlying formula is built bottom-up from the local structure of the motif,

$$\begin{aligned}
\text{leaf}(u) &= \text{blue}(u) \wedge \Diamond_{=1} \top && \text{a blue node of degree one,} \\
\text{core}(u) &= \text{red}(u) \wedge \Diamond_{=2}\, \text{leaf} && \text{a red node with exactly two blue leaves,} \\
\text{end}(u) &= \text{core}(u) \wedge \Diamond_{=1}\, \text{core} && \text{endpoint of the chain,} \\
\text{mid}(u) &= \text{core}(u) \wedge \Diamond_{=2}\, \text{core} && \text{interior node of the chain,} \\
\text{midend}(u) &= \text{mid}(u) \wedge \Diamond_{\geq 1}\, \text{end} && \text{interior node touching an end point,}
\end{aligned}$$

and closed with the following node predicate and graph-level quantifier,

$$\varphi(u) \;=\; \text{midend}(u) \wedge \Diamond_{\geq 1}\, \text{midend}, \qquad \Phi \;=\; \exists u.\, \varphi(u), \tag{14}$$

which fires on the two interior nodes of the chain. The reason for this construction is to avoid counts by backtracking: the naive "four red cores next to each other" is satisfied even by a chain of length 2, since a walk can simply bounce between two adjacent red cores. We further provide in Appendix C.5 an ablation study on the impact of how the formula above is written on the plausibility evaluation.

Redundancy. The palette is $[\text{black}, \text{red}, \text{grey}, \text{silver}]$, where the last two shades are reserved for the background. The node predicate, and the graph-level quantifier closing it, are

$$\varphi(u) \;=\; \text{black}(u) \wedge \Diamond_{\geq 1}\, \text{red}, \qquad \Phi \;=\; \exists u.\, \varphi(u), \tag{15}$$

i.e., 'a black node with at least one neighbour of colour red'. A positive graph carries exactly one black node, together with 6 red neighbours and further connected to the background by 5 edges to random background nodes, which are the irrelevant edges an explainer has to reject. Since no background node carries the anchor colour, the planted anchor is the only node that can ever satisfy φ. Note that any single one of the 6 edges between black and red nodes constitutes a valid PI explanation. Hence, their union UPI is composed of the whole group of 6 edges.

CompetingRedundancy. The palette is $[\text{black}, \text{red}, \text{blue}, \text{green}, \text{grey}, \text{silver}]$ and the node predicate, with the graph-level quantifier closing it, is

$$\varphi(u) \;=\; \text{black}(u) \wedge \big(\Diamond_{\geq 1}\, \text{red} \;\vee\; \Diamond_{\geq 2}\, \text{blue} \;\vee\; \Diamond_{\geq 3}\, \text{green}\big), \qquad \Phi \;=\; \exists u.\, \varphi(u). \tag{16}$$

The construction follows the one above, except that the black node now contains three groups of candidate neighbours in three different colours, planted with 1, 2 and 3 members respectively, and each group has to be complete for its disjunct to be true. A negative graph starves every group in the same way as above, planting one member fewer than its threshold. Differently from `Redundancy`, there are exactly 3 alternative explanations, one per colour-group. UPI is their union, meaning 6 edges in total, the same as in `Redundancy`.

Common protocol for node classification. A node-classification task has no graph-level quantifier: the formula $\Phi(u)$ itself labels the target node, so what the explainer scores is the receptive field of the target rather than the whole graph. Each of these datasets is composed of several Barabási–Albert graphs with attachment parameter 2 (Barabási & Albert, 1999) and palette $[\text{red}, \text{blue}, \text{green}]$; we randomly generate and accumulate these graphs until we collect at least 500 nodes on which φ fires. We do so in order to limit the size and the density of the receptive field, and therefore the time required to extract the ground truth for each target node. Below, $\Diamond_{>}(\varphi, \psi)$ is the comparative operator of Appendix C.2, and it reads 'strictly more neighbours satisfy φ than ψ' (see Appendix C.2 for the details regarding its implementation).

Majority1Hop. A single graph is 500 nodes, each coloured uniformly at random from the palette $[\text{red}, \text{blue}, \text{green}]$, and the node predicate is

$$\Phi(u) \;=\; \Diamond_{>}(\text{red}, \text{blue}), \tag{17}$$

i.e., 'strictly more red neighbours than blue ones'. A node can be assigned no colour, in which case we refer to it as of grey colour.

`Majority2Hop`. Same graph and same construction as `Majority1Hop`, with the predicate applied twice (Nerini et al., 2025),

$$\text{red-ish}(u) \;=\; \Diamond_{>}(\text{red}, \text{blue}),\;\; \text{blue-ish}(u) \;=\; \Diamond_{>}(\text{blue}, \text{red}),\;\; \Phi(u) \;=\; \Diamond_{>}(\text{red-ish}, \text{blue-ish}). \tag{18}$$

The inner predicates give every node a derived label, and the outer one compares those derived labels across the neighbourhood, which is exactly two rounds of label propagation. The receptive field grows accordingly, from one hop to two, and with it the cost of extracting UPI (Table 13).

`RedBlueLeaf-NC`. This is the node-level counterpart of `RedBlueLeaf`, built from the same vocabulary,

$$\begin{aligned} \text{leaf}(u) \;&=\; \text{blue}(u) \wedge \Diamond_{=1} \top && \text{a blue node of degree one,} \\ \text{core}(u) \;&=\; \text{red}(u) \wedge \Diamond_{=2}\, \text{leaf} && \text{a red node with exactly two blue leaves,} \end{aligned}$$

and closed into the classifier

$$\Phi(u) \;=\; \text{core}(u) \wedge \Diamond_{\geq 1}\, \text{core}, \tag{19}$$

which fires on a red core adjacent to another red core, and therefore needs three hops to be decided: the neighbour's colour, that neighbour's blue leaves, and the degree of each leaf. This represents a minor modification with respect to `RedBlueLeaf`: instead of distinguishing between end, mid or midend nodes, we simplified the formula in order to have multiple nodes firing for each planted motif. On a background of 300 nodes we plant 50 units, each a path of three red cores carrying two blue leaves apiece, whose two endpoints are tied to two random background nodes recoloured green first, for 750 nodes in total. Because the model uses a 3-hop receptive field, even the central core node integrates multiple background edges outside the motif due to the pair of anchor points. As in `RedBlueLeaf`, green is inert. Note that the planted unit and the formula deliberately do not coincide: in a path of three cores, the middle core and the two endpoint cores fire for structurally different reasons, so targets inside the same unit have different explanations.

Table 15: Size of the datasets, averaged over the positive samples of the 5 evaluation seeds. Nodes and edges are counted per graph, and edges are counted as directed, i.e., an undirected edge contributes two. For node classification we keep sampling graphs until the dataset holds at least 500 explainable targets, so the number of graphs is a range over the seeds rather than a fixed size. The last column is the fraction of edges that the explainers are asked to recover. For node classification, the denominator is the receptive field of the target.

Dataset	graphs	nodes	edges	UPI edges	UPI fraction
`RedBlueLeaf`	1000	32	98.0	22.0	0.22
`Redundancy`	1000	37	134.0	6.0	0.04
`CompetingRedundancy`	1000	37	134.0	6.0	0.04
`RedBlueLeaf-NC`	4	750	2192.0	10.7	0.29
`Majority1Hop`	3–4	500	1992.0	2.3	0.60
`Majority2Hop`	3–4	500	1992.0	17.1	0.43

G Additional Details on the Survey

In this section, we describe the procedure by which we collected the relevant information from previous literature summarized in Fig. 1. Our literature search was restricted to peer-reviewed venues, specifically CORE A^* conferences, leading journals, and key domain-specific conferences. The complete set of targeted venues comprises:

- *Conferences*: AAAI, ACL, AISTATS, CVPR, FAccT, ICDM, ICDE, ICLR, ICML, IJCAI, KDD, LOG, NeurIPS, VLDB, and WWW.
- *Journals*: ACM Computing Surveys, ACM Transactions on Intelligent Systems and Technology (ACM TIST), Proceedings of the ACM on Programming Languages (PACMPL), Nature Communications, Scientific Data, Transactions on Machine Learning Research (TMLR), and IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)

We intentionally included LOG and FAccT given their direct domain relevance to graph machine learning and algorithmic transparency.

We discarded works targeting non-standard explanation modalities (e.g., global decision boundaries (Wang & Shen, 2024)) where faithfulness and plausibility metrics cannot be meaningfully applied. We divided each entry according to the type of evaluation used in the paper: plausibility, faithfulness, both, or none of them (a minority of papers used ad-hoc metrics or entirely different objectives). For each paper, we annotate the following information: type of paper (whether it is a methodological paper, a survey, a benchmark, a critique, applied, or a mix thereof), whether the authors make sure the underlying model has learned the intended ground truth, the type of explanation provided (global vs local, post-hoc vs ante-hoc), the target tasks (node classification, graph classification, etc.), and the datasets used in the evaluation.

H Implementation details

Table 16: **Hyper-parameter search space of each explainer**, shared across all tasks. $[a, b]_{\log}$ denotes a log-uniform range, $[a, b]$ a uniform one, and $\{\cdot\}$ a categorical choice. The other explainers are run in their default configuration.

Explainer	Hyper-parameters and ranges
GNNExplainer	`epochs` $\in \{50, 100, 1000\}$; `lr` $\in [10^{-4}, 10^{-1}]_{\log}$; `edge_size` $\in [10^{-4}, 10^{-1}]_{\log}$; `edge_ent` $\in [10^{-4}, 10^{-1}]_{\log}$
GraphMask	`epochs` $\in \{5, 10, 25, 50\}$; `lr` $\in [10^{-5}, 10^{-2}]_{\log}$; `penalty_scaling` $\in \{1, \dots, 10\}$; `allowance` $\in [0, 0.3]$
SubgraphX	`max_nodes` $\in \{5, \dots, 20\}$; `min_atoms` $\in \{3, \dots,$ `max_nodes`$\}$; `sample_num` $\in \{10, 20, 50\}$; `rollout` $\in \{3, 5, 10, 20\}$; `expand_atoms` $\in \{2, \dots, 10\}$; `local_radius` $\in \{1, 2, 3\}$
PGMExplainer	`num_samples` $\in \{50, 100, 200, 500\}$; `perturbation_mode` $\in \{$`randint`, `zero`, `mean`, `max`$\}$; `significance_threshold` $\in [10^{-3}, 10^{-1}]_{\log}$; `pred_threshold` $\in [0, 0.5]$; `is_perturbation_scaled` $\in \{$`false`, `true`$\}$; `max_subgraph_size` $\in \{1, \dots, 50\}$ (graph tasks only)
GraphSVX	`num_samples` $\in \{100, 200, 400, 800\}$; `S` $\in \{1, \dots, 4\}$; `args_coal` $\in \{$`Smarter`, `Smart`, `Random`$\}$; `args_g` $\in \{$`WLS`, `WLR_sklearn`$\}$; `use_abs` $\in \{$`true`, `false`$\}$

Hyperparameters tuning of explainers: Hyperparameters are tuned for every task with Optuna (Akiba et al., 2019) maximising the mean F_1 on a dataset generated with a held-out seed, so that no explainer is tuned on the instances it is later scored on. We run 50 trials per explainer and task, each scored on 50 instances, and report the searched ranges in Table 16.

Explainer time budget: Given that some explainers are particularly computationally expensive, we set a time budget of 180 seconds for explaining individual instances. During the evaluation of explainers in Section 5, whenever an instance takes more than 180 seconds to be explained, the process is halted, and no further samples are explained for that specific seed. When an explainer does not finish any of the random seeds, we report '*OOT*' in the corresponding table entry.

H.1 Implementation details of explainers

For some explainers, we performed some modifications to their implementation in order to adapt them to our evaluation setting. We detail each of the modifications in the following paragraphs, along with the corresponding justification. In every case, these modifications are applied so as not to change the original scope or functionality of the explainer. Implementations of explainers are taken from PyG (Fey et al., 2025), except when otherwise specified.

SubgraphX. (Yuan et al., 2021) We adopt the implementation provided in Agarwal et al. (2023)[7]. For node tasks, after computing the explanation, this implementation applies a filter and return only the

[7] https://github.com/mims-harvard/GraphXAI

attribution on the computational graph of the node. Since it is the only explainer to do so, we removed this filter and compared the explanation on the full graph instead, to make the evaluation identical for all explainers. Because underlying computation of the explainer is inherently restricted to the computation graph, edges falling outside of it are simply zero-padded. Note that this modification does not affect execution time, and the explainer infers the depth of the computation graph directly from the number of message-passing layers in the target model.

GNNExplainer. (Ying et al., 2019) GNNExplainer scales every message between node u and v by $\sigma(m_{u,v})$ for a learned logit $m_{u,v}$, and the default initialisation sets $m_{u,v} \approx 0$, so that each message is halved. We noticed that this can create instabilities for some datasets (like `RedBlueLeaf`) in the training of the logits, as halving the count of incoming messages can greatly change the total count of the neighbours satisfying a given sub-formula, even at initialization time. To reduce this effect, we use the bias term introduced in the original implementation[8] provided by Ying et al. (2019), such that logits are initialized at values around $\sigma(m_{u,v}) \sim 1$. In this way, the count of satisfying neighbours is roughly preserved at initialisation time. Contrary to the original implementation, this bias term is only used for initializing the logits, and is not injected at every forward step. Then, during training, GNNExplainer will attempt to sparsify the logits as much as possible, following the original implementation. We verified that the value of this bias does not drive our insight regarding the sparsification ability of GNNEXPLAINER described in Section 5.2.1 by sweeping the additional bias term over $\{0, 1, 2.5, 5\}$ on `Redundancy`, where 0 recovers the unbiased initialisation. Precision, recall and F_1 all remain equal to 1.00 ± 0.00 over the same 5 seeds, exactly the value reported in Table 2.

FLOWX. (Gui et al., 2024) We adopt the original FLOWX implementation[9]. FLOWX is a decomposition explainer, which decomposes the contribution of each part of the graphs through *flows*. Besides the standard method, which we adopt, the authors of the paper propose variations that have an additional training step to refine the original explanations. We observed that the original implementation of the baseline method had a training loop of a single iteration, inherited from its variations, which did not perform any actual training. Moreover, inside the loop, there were additional operations which produced OOM errors when running FLOWX on models with more than three message-passing layers. As such, we modified these operations and removed the calculation of the gradients in order to avoid OOM issues. We compared the original and the new version of the explainer on a single seed of the tasks `Majority2Hop` and `RedBlueLeaf-NC` in Table 17, showing no distinguishable behaviour besides a small rounding error on the recall in `Majority2Hop`.

Table 17: Comparison in performance of the new implementation of FLOWX, reporting mean $\pm$ standard deviation for explanations across sample nodes, for a single seed.

	`Majority2Hop`		`RedBlueLeaf-NC`	
Explainer	precision	recall	precision	recall
FLOWX	$0.888_{\pm 0.150}$	$0.942_{\pm 0.103}$	$0.994_{\pm 0.028}$	$0.998_{\pm 0.014}$
FLOWX (original)	$0.888_{\pm 0.150}$	$0.943_{\pm 0.103}$	$0.994_{\pm 0.028}$	$0.998_{\pm 0.014}$

PGMEXPLAINER. (Vu & Thai, 2020) We adapted the implementation of PGMEXPLAINER provided in PYG (Fey et al., 2025). Originally, PGMEXPLAINER provides as the explanation the p-value scores associated with each node for its statistical relevance, together with a binary mask on the k nodes with *lowest* p-value (where k is a parameter of the explainer). Since in this work we evaluate exclusively edge-level explanations, we assigned to each edge $e = (u, v)$ an importance score $1 - \max(p_u, p_v)$: in this way, an edge is important only as much as its less important endpoint.

GRAPHSVX. (Duval & Malliaros, 2021) We adopt the original GRAPHSVX implementation[10]. GRAPHSVX is a perturbation-based explainer that performs attribution for each node in the k-hop neighbourhood of the explained node and for each of its features. Since in this work we evaluate exclusively edge-level explanations, we disable the feature players so that the whole coalition budget is spent on the graph structure, and, as for PGMEXPLAINER, we assign to each edge $e = (u, v)$ the score

[8] We note that this bias term is only present in the original codebase available at https://github.com/RexYing/gnn-model-explainer, and not in the implementation provided in PyG (Fey et al., 2025).

[9] https://github.com/CM-BF/FlowX

[10] https://github.com/AlexDuvalinho/GraphSVX

$\min(|\phi_u|, |\phi_v|)$, where ϕ are the estimated Shapley values; the explained node, which is not a player, receives the maximum score of its neighbourhood. Moreover, the original implementation assumes dense adjacency matrices for graph classification tasks, incompatible with the sparse encoding of PyG; we adapted the code to work with sparse matrices as well, keeping the original coalition sampling and regression. Because it assigns equal importance to both edge directions, its maximum achievable precision is bounded at 50% on `Majority1Hop`, `Redundancy`, and `CompetingRedundancy`, as relevant edges in these datasets contribute exclusively in one direction. Nonetheless, across all these tasks, the explainer reaches a precision of $\approx 50\%$, implying that it successfully localizes relevant nodes, but its inherent design limitations prevent it from distinguishing fine-grained edge directionality. Similar reasoning applies to `Majority2Hop`, where the explainer reaches 56% precision: although both directions of an edge can be important, typically only one contributes, meaning its theoretical maximum is only marginally higher. The same theoretical cap applies to PGMExplainer and SubgraphX as well, though both achieve significantly lower precision than GraphSVX in practice.

LRP. (Bach et al., 2015) We adopt the LRP implementation provided in Captum (Kokhlikyan et al., 2020), through the Captum interface of PyG. LRP propagates the model output backwards through the network according to layer-specific rules. We adopted the baseline epsilon rule for all the layers of our compiled models, the same rule Captum applies by default.

I Proof of Theorem 1

Theorem 1. *Let Φ be a classifier and G a graph such that $G \models \Phi$, let $f = \textsc{Build}(\Phi, G)$ be its* ROBDD *encoding, and let $f^{\uparrow}$ be the output of $\textsc{Extend}(f)$ so that $f^{\uparrow}$ maps a subgraph R to 1 iff R is sufficient. Then, for any edge e in G, it holds that $(f^{\uparrow} \wedge x_e \wedge \neg f^{\uparrow}|_{x_e=0} \neq 0) \iff e \in \mathsf{UPI}(f, G)$.*

Proof. We will analyze the two directions separately:

($\leftarrow$)

We aim to show that if $e \in \mathsf{UPI}$, then $f^{\uparrow} \wedge x_e \wedge \neg f^{\uparrow}|_{x_e=0} \neq 0$. Then, by assumption, e belongs to a PI, say M. We note that M is by itself a witness of $f^{\uparrow} \wedge x_e \wedge \neg f^{\uparrow}|_{x_e=0} \neq 0$:

- M is sufficient by definition, hence $f^{\uparrow}(M) = 1$
- $e \in M$ by assumption, hence $x_e = 1$
- $M \setminus \{e\}$ is not sufficient, since by definition of PI there exists no subset of M that is also sufficient, hence $f^{\uparrow}|_{x_e=0}$ evaluates to 0 at M, and $\neg f^{\uparrow}|_{x_e=0}$ to 1

Hence, $f^{\uparrow} \wedge x_e \wedge \neg f^{\uparrow}|_{x_e=0} \neq 0$ since M is a valid witness.

($\rightarrow$)

We aim to show that if $\kappa := (f^{\uparrow} \wedge x_e \wedge \neg f^{\uparrow}|_{x_e=0}) \neq 0$, then $e \in \mathsf{UPI}$. The fact that $\kappa \neq 0$ means there must exist a subgraph R such that the following hold together:

(H1) R is sufficient,

(H2) edge e is part of R,

(H3) $R \setminus \{e\}$ is not sufficient.

Let us define $D := \{S : S \subseteq R \wedge f^{\uparrow}(S) = 1\}$. Note that $D \neq \varnothing$ as $R \in D$. Since D is finite, there must be an element R' with the smallest cardinality. Since R' has the smallest cardinality, it is also the smallest sufficient subgraph in D, hence R' is a PI.

We now want to show that e must be in R', meaning $e \in \mathsf{UPI}$. Let us assume to the contrary that $e \notin R'$. Then, $R' \subseteq R \setminus \{e\}$. However, R' is sufficient since it is a PI, and therefore any of its supergraphs must also be sufficient, including $R \setminus \{e\}$. This contradicts the hypothesis (H3), meaning it was absurd to assume $e \notin R'$. Hence, $e \in \mathsf{UPI}$. □

Note that the restriction to positive graphs is without loss of generality. For a negative graph, UNIONPI runs EXTEND on $\neg f$ (line 3 of Algorithm 1), for which G is positive; since Def. 1 compares $g(R')$ with $g(G)$, a subgraph is sufficient for $\neg f$ iff it is for f, hence $\mathsf{UPI}(\neg f, G) = \mathsf{UPI}(f, G)$. An interesting edge case of Theorem 1 is whenever UPI is empty. Note that this can only happen when the prediction on graph G is invariant to the removal of any subset of edges, meaning $\varnothing$ is a PI. Cases where this happens are negative instances of `Redundancy`, or tasks like *the graph has an isolated node*. The following Corollary shows that Algorithm 3 will return precisely an empty mask in this case:

Corollary 1. *Under the same assumptions of Theorem 1:*

$$\left(\forall e \in \mathcal{E}.\, f^{\uparrow} \wedge x_e \wedge \neg f^{\uparrow}|_{x_e=0} = 0\right) \iff \mathsf{UPI}(f, G) = \varnothing. \tag{20}$$

Proof. Both directions are immediately proved by contradiction with Theorem 1. On one hand, let us assume $\mathsf{UPI}(f, G) = \varnothing$ to be true. Assume for the sake of contradiction that Eq. (20) is false, meaning that, $\exists e \in \mathcal{E}. f^{\uparrow} \wedge x_e \wedge \neg f^{\uparrow}|_{x_e=0} \neq 0$, then by Theorem 1 $\mathsf{UPI}(f, G)$ must be non empty, leading to a contradiction. On the other hand, let us assume the left side of Eq. (20) to be true. Assume for the sake of contradiction that the right side is false, meaning that $\mathsf{UPI}(f, G) \neq \varnothing$, then there must exist an $e \in \mathsf{UPI}(f, G)$: but by Theorem 1, this implies $f^{\uparrow} \wedge x_e \wedge \neg f^{\uparrow}|_{x_e=0} \neq 0$, which represents a contradiction. □