Explaining GNN Negatives Globally and Locally

Kehan Pang[1], Wenfei Fan[1,2,3], Min Xie[2], Dandan Lin[2]
[1]Beihang University [2]Shenzhen Institute of Computing Sciences [3]University of Edinburgh
pangkehan@buaa.edu.cn, wenfei@inf.ed.ac.uk, {xiemin, lindandan}@sics.ac.cn

***Abstract*—This paper studies explanations for graph neural network (GNN) classifiers $\mathcal{M}$ when $\mathcal{M}$ makes negative predictions, such as loan denials, paper rejections, or job application turn-downs. The objective is to (a) provide global explanations of $\mathcal{M}$'s behavior and (b) generate *counterfactual explanations* locally at a vertex u, suggesting the minimal changes to features or topology around u needed to flip $\mathcal{M}$'s prediction. We propose a class of rules that treat negative predictions as their consequences, and employ the 1-dimensional Weisfeiler–Leman (1-WL) test as a predicate, which is at least as expressive as GNN classifiers. We develop algorithms to (a) learn such rules for global explanations and (b) apply them to compute local counterfactuals. Extensive experiments on real-world graphs show that our approach outperforms state-of-the-art methods across major metrics.**

I. Introduction

When an ML model $\mathcal{M}$ makes negative predictions, one naturally seeks explanations, since understanding the reasons behind refusal is often more critical than for positive outcomes. Why was my paper rejected, my loan denied, or my job application turned down? Moreover, what minimal changes would allow me to pass the model's check?

Explanations for $\mathcal{M}$'s predictions fall into two main types.

- *Global explanations* capture the *input-independent* behavior of $\mathcal{M}$, identifying structures and features behind negative predictions, to provide insights and build trust.
- *Local explanations* (*a.k.a. counterfactual explanations*) suggest minimal changes to a specific *input instance* to flip the prediction, such as paper acceptance or loan approval.

This paper studies both global and local explanations when Graph Neural Networks (GNNs) make negative predictions. We focus on GNNs, widely-used in real-world applications such as recommendation systems [1], [2], fraud detection [3], [4] and drug discovery [5], [6], [7], [8], among others.

Prior work on explaining GNN negatives includes local [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19] (surveyed in [20]), global [21], [22], [23], [24], [25], and hybrid approaches [26], [27]. However, these methods usually suggest subgraph modifications without clarifying which features drive predictions or under what conditions they hold. Moreover, they often propose making immutable changes, which is infeasible. Furthermore, none of them provides a unified logical framework for both global and local explanations.

Example 1: Consider a real job application case [28] (Figure 1), where an applicant u (ID 2445) is turned down by a GNN model $\mathcal{M}$. For clarity, we show only representative perturbations within the 2-hop neighborhood graph G_u of u.

Prior methods craft local counterfactuals by altering sub-

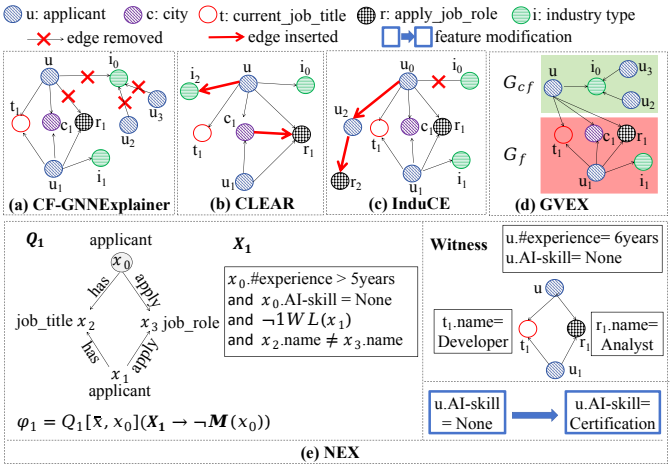


Fig. 1: A case study to swap prediction at u's job application

graph topology, often ignoring features and feasibility.

- CF-GNNExplainer [11] (Figure 1(a)) removes multiple edges from G_u, *e.g.*, (u_2, works_in, i_0). However, since vertex features are ignored, the prediction is not flipped.
- CLEAR [14] (Figure 1(b)) adds new edges, such as (c_1, r_1). While this changes the prediction, it violates domain constraints (*e.g.*, a city c_1 cannot connect to a job-role r_1).
- InduCE [13] (Figure 1(c)) inserts an edge between two applicants and alters the job role of applicant u_1, which is unrealistic; CFGraph [27] suffers from the same issue.
- GVEX [26] (Figure 1(d)) deletes a factual subgraph G_f from G_u to get G_{cf}, but G_{cf} cannot swap the prediction.

Overall, these methods only perturb edges but ignore feasible feature updates, leading to less effective explanations.

For global explanations, existing methods are limited primarily because they overlook vertex features. Specifically, (a) XGNN [21] and CFGraph [27] generate patterns with fake edges; (b) GNNInterpreter [22] and GVEX [26] produce overly complex patterns that obscure the most discriminative substructures; and (c) GVEX also suffers from poor efficiency. □

To address these issues, we present NEX, a unified logic-based framework for explaining negative GNN predictions both *globally* and *locally*. NEX introduces RxGNNs (Rules for eXplaining GNN classifiers), a new class of logic rules to explain model behavior for negative decisions (see Section II).

Given a GNN $\mathcal{M}$ and a vertex variable x_0 with $\mathcal{M}(x_0)$ = false, denoted as $\neg\mathcal{M}(x_0)$, NEX discovers RxGNNs of the form $Q(X \rightarrow \neg\mathcal{M}(x_0))$, where Q is a graph pattern pivoted at x_0, and X is a set of predicates on vertex features. Intuitively, Q captures topological structures around x_0, and X specifies

features conditions. These rules both yield global explanations by revealing the structures and features most responsible for $\neg\mathcal{M}(x_0)$, and derive local explanations for vertices u in graph G by identifying specific matches of Q and X as witnesses; counterfactuals are deduced by perturbing these witnesses.

Example 2: Continuing Example 1, Figure 1(e) shows a mined RxGNN φ_1 and its witness at applicant u in graph G [28]. Rule φ_1 consists of a graph pattern Q_1 and a precondition X_1 on vertex features. The witness explains why $\neg\mathcal{M}_1(u)$: (a) u's job title is manager while applying for analyst, indicating a career change; (b) another applicant u_1 shares the same title and role as u, and u_1 has already been denied (by the 1-WL test [29], [30], see Section II-A); and (c) although u has sufficient experience (> 5 years), u lacks an AI-skill certification.

By perturbing this witness, NEX suggests u get an AI-skill certification. This counterfactual explanation is feasible for u to implement and more importantly, it flips $\mathcal{M}_1$'s prediction.

Unlike prior methods (Example 1), φ_1 pinpoints which structures and features drive the negative prediction and under what conditions. It then derives a feasible counterfactual (*e.g.,* by updating mutable features) rather than simply altering graph topology. It also makes less perturbations than prior methods.

Unlike prior global methods, RxGNN φ_1 not only accounts for individual features, but also highlights the most discriminative substructure, containing only real edges in G. □

Contributions & organization. We propose a uniform framework for both global and local explanations of GNN classifiers.

(1) Rules for eXplaining GNN (RxGNNs) (Section II). We propose RxGNNs for explaining negative GNN decisions. Unlike prior graph rules, RxGNNs (a) support rich predicates, including negation and the 1-WL test [29], [30] (as expressive as GNNs [31], [32]), to capture diverse behaviors of GNNs; (b) use general graph patterns to encode graph topology; and (c) treat negative predictions as consequences, enabling counterfactual explanations via actionable input perturbations.

(2) NEX: Negative prediction EXplanations (Section III). We develop NEX, a unified framework for explaining GNN negatives. Given a classifier $\mathcal{M}$, NEX discovers a set Σ of RxGNNs as global explanations. For a given vertex u in graph G where $\mathcal{M}$ predicts negative, NEX derives a counterfactual explanation, the necessary changes for flipping $\mathcal{M}$'s decision.

(3) Model-guided discovery (Section IV). We develop an efficient, parallelly scalable [33] algorithm for discovering RxGNNs, guided by a given GNN $\mathcal{M}$. It first identifies motifs most responsible for $\mathcal{M}$'s predictions, and then composes them into graph patterns via a deep Q-learning policy [34] to avoid exponential search. Preconditions are subsequently generated with a novel predicate ordering to reduce computational cost.

(4) Counterfactual explanations (Section V). For a vertex u in graph G with $\mathcal{M}(u) = \mathsf{false}$, we seek a set of perturbations that invalidates all RxGNNs in Σ at u, *i.e.,* the minimal changes needed to flip $\mathcal{M}(u)$ from negative to positive. Although the problem is NP-hard, we develop a cost-effective strategy that, guided by $\mathcal{M}$, selects edge removals and feature modifications while minimizing perturbations to G.

(5) Experimental study (Section VI). Using real-world graphs, we find (a) RxGNNs learned by NEX provide accurate global explanations, with recognizability (measuring how consistently the rules reflect model behavior) and reliability (measuring how well the rules cover testing vertices) 70.16% and 189.68% higher than baselines on average. (b) NEX yields effective counterfactuals, with fidelity (success in flipping predictions) and sparsity (minimality of perturbations) improved by 33.60% and 3.25×, respectively, over prior methods. (c) NEX is efficient: achieving 3.06× parallel speedup (16 vs. 4 processors) in rule discovery, and generating a counterfactual in 1.38s on a graph with 15.4M vertices and 12.7M edges.

The source code and data are available in [35], and the full version is [36], providing proofs and algorithm details.

Related work. We categorize the related work as follows.

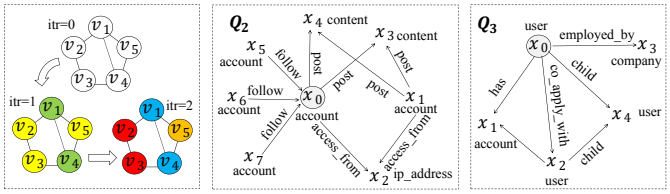
Explanation methods. For the lack of space, we focus on counterfactual explanation methods for GNN-based models.

(1) Local counterfactual explanation. Existing methods [9], [10], [11], [12], [13], [14], [15], [16], [8], [17], [18], [19] (surveyed in [20]) generate counterfactuals by identifying minimal perturbations to the input graph that flip GNN predictions. For example, CF-GNNExplainer [11] uses optimization on edge removals; RCExplainer [12] removes resilient edge subsets; CLEAR [14] employs a variational autoencoder; InduCE [13] leverages reinforcement learning; and LocalCE [16] finds pairs of similar but differently classified vertices as counterfactuals.

(2) Global explanation. Global methods generate candidate counterfactual graph patterns, including XGNN [21], which uses reinforcement learning; GNNInterpreter [22], which is based on numerical optimization; DAG [23], which applies randomized greedy search; and GLGExplainer [37], which employs prototype learning. GCFExplainer [24] performs global counterfactual reasoning via vertex-reinforced random walks, and ComRecGC [25] generates counterfactual graphs relevant to all inputs. Although some methods (*e.g.,* XGNN) do not take specific graphs as input, they still rely on pre-trained GNNs and thus implicitly depend on the training data. In contrast, NEX is designed to generate both global and local explanations and thus, naturally operates on a specific graph.

(3) Hybrid global and local. GVEX [26] uses a two-tier structure, combining global graph patterns with local subgraph explanations. CFGraph [27] derives counterfactuals by adding or removing edges and aggregates them into global explanations. The most related work, Makex [38], employs rule-based explanations for GNN link predictions but with less expressive rules, limited star patterns, and restricted predicates. It supports only factual explanations by identifying sufficient substructures, whereas counterfactuals require graph modifications and thus rules with negated predicates (see [36] for details).

(4) Non-GNN explanations. Logic-based explanation methods [39], [40], [41], [42], [43], [44] have been studied across



(a) An example of 1-WL test (b) Examples of graph patterns

Fig. 2: Running examples

various data modalities (see surveys [45], [46]). Systems such as Geco [47], FACET [48], Mace [49] and Dice [50] identify plausible feature changes to flip ML predictions. Alternative approaches employ declarative programming (*e.g.,* answer-set programming [44]) or automated reasoning with SAT/SMT solvers [43]. In contrast, GNN explanations must account for graph topology, which these methods cannot capture.

Discussion. This work differs from prior work in several key aspects: (a) RxGNNs combine graph patterns with preconditions, revealing not only which structures are decisive but also under what feature conditions predictions are made, beyond simple meta-paths or subgraphs used earlier; (b) we emphasize counterfactual explanations that flip predictions, unlike factual explanations that require no graph changes, general graph patterns, or negated predicates such as [38]; (c) we provide a unified framework for both global and local explanations applicable to any GNN classifier; and (d) unlike prior logic-based methods [39], [40], [41], [44], [43], which rely on constant predicates or manual preference settings, RxGNNs support diverse predicates (including negation and 1-WL test), enabling expressive reasoning for GNNs with automatic setup.

II. Rules for Explaining GNN Negatives

This section first reviews 1-WL test (Section II-A), and then defines rules for explaining negative predictions (Section II-B).

A. Graph Patterns and 1-WL Test

Assume two countably infinite sets of symbols Λ and Υ, for labels and vertex attributes (*i.e.,* vertex features), respectively.

Graphs. We consider directed labeled graphs $G = (V, E, L_G, F_A)$, where (a) V is a finite set of vertices; (b) $E \subseteq V \times \Lambda \times V$ is a finite set of labeled edges, with $e = (v, l, v')$ denoting an edge from v to v' labeled $l \in \Lambda$; (c) each vertex $v \in V$ has a label $L_G(v) \in \Lambda$ and a tuple of attributes $F_A(v) = (A_1 = a_1, \ldots, A_n = a_n)$, where $A_i \in \Upsilon$, a_i is a constant, and $v.A_i = a_i$ (with $A_i \neq A_j$ if $i \neq j$). Different vertices may carry different attributes, while edges carry only labels.

Patterns. A (connected) *graph pattern* is $Q[\bar{x}, x_0] = (V_Q, E_Q, L_Q)$, where (1) V_Q and E_Q are finite sets of pattern vertices and edges, respectively; (2) L_Q assigns a label $L_Q(v) \in \Lambda$ to each vertex $v \in V_Q$; (3) $\bar{x}$ is a list of distinct variables, each $x \in \bar{x}$ denoting a vertex $v \in V_Q$ [51], [52]; and (4) x_0 is a designated variable in $\bar{x}$, representing the entity of interest.

We denote variables in Q by x, y, and vertices in G by u, v.

Example 3: The graph pattern $Q_1[\bar{x}, x_0]$ in Figure 1(e) models entities and relations in the graph of real-life job applications [28]. It has four vertex variables $\{x_0, x_1, x_2, x_3\}$ labeled as applicant, job_title and job_role, and four edges; *e.g.,* $(x_0, \mathsf{apply}, x_3)$ denotes that applicant x_0 applies for job x_3. □

Notations	Definitions
G, $Q[\bar{x}, x_0]$	graph, graph pattern
$\neg\mathcal{M}(x_0)$	a GNN model that predicts false at a vertex x_0
$\neg\mathsf{1WL}(x)$	the (negated) predicate for 1-WL test
p, X	a predicate, precondition (collection of predicates)
φ	an RxGNN $\varphi = Q[\bar{x}, x_0](X \rightarrow \neg\mathcal{M}(x_0))$
Σ	a set of RxGNNs
σ, δ, N	thresholds of support and confidence, max pattern size
$\mathsf{supp}()$, $\mathsf{conf}()$	the support and confidence of an RxGNN φ on graph G
$\mathbb{O}$, $G \ominus \mathbb{O}$	a set of perturbations in G, the graph G perturbed by $\mathbb{O}$
$\mathsf{cost}(v)$, $\mathsf{eff}(v)$	the cost and effectiveness of a vertex v

TABLE I: Notations

Pattern matching. Following [53], [54], a *match* of a pattern $Q[\bar{x}, x_0]$ in a graph G is a homomorphism $h : Q \rightarrow G$ such that: (a) for each vertex $v \in V_Q$, $L_Q(v) = L_G(h(v))$; and (b) for each edge $e = (v, l, v') \in E_Q$, $(h(v), l, h(v')) \in E_G$.

GNN. Graph neural networks (GNNs) [55], [56], [57], [58] leverage graph topology and vertex features to learn embeddings for vertex classification. Let $\mathbf{e}_v$ denote the initial feature vector of vertex v obtained from its attributes. An L-layer GNN iteratively refines $\mathbf{e}_v$ by aggregating embeddings of its L-hop neighbors. At layer t, the embedding of v is:

$$\mathbf{e}_v^t = \mathsf{Update}(\mathbf{e}_v^{t-1}, \mathsf{Aggregate}(\{\mathbf{e}_{v'}^{t-1}\} \mid v' \in \mathcal{N}(v))).$$

Here Aggregate and Update denote the aggregation and update functions, respectively, and $\mathcal{N}(v)$ is the 1-hop neighborhood of v. After L iterations, the model outputs the final embedding $\mathbf{e}_v^L$; its classification probability is given by function $f(\mathbf{e}_v^L)$ learned *w.r.t.* the training classes. For simplicity, we follow [58], [40] and study binary vertex classification: given a threshold λ, $\mathcal{M}(v)$ returns true if $f(\mathbf{e}_v^L) \geq \lambda$ and false otherwise.

1-WL test. We review the 1-dimensional Weisfeiler-Leman (1-WL) test [29]. Given a graph G where each vertex is initialized with a color (from its feature vector), 1-WL iteratively refines colors [32]: for vertices v and v' of the same current color, they receive different new colors if they differ in the number of neighbors of some color d. The process iterates until stable, and vertices sharing the same final color form a class.

Example 4: Consider Figure 2(a), an example borrowed from [59]. All vertices start white, and colors are refined iteratively. In the first iteration, v_1 and v_5 receive different colors as they have 3 and 2 white children, respectively. After two iterations the process stabilizes, with v_2 and v_3 sharing the same final color and thus belonging to the same class. □

Notations. We will use the following notations (see Table I).

(1) We use $\neg\mathcal{M}(x)$ to denote a negative prediction (false) of model $\mathcal{M}$ at a vertex, *i.e.,* $f(\mathbf{e}_v^L) < \lambda$ for threshold λ. Treated as a Boolean predicate, $\neg\mathcal{M}(x)$ captures negative outcomes on which we focus, as users typically care about reversing them.

(2) We treat $\neg\mathsf{1WL}(x)$ as a predicate, which holds if there exists a vertex y such that $\mathcal{M}(y) = \mathsf{false}$, and x and y are 1-WL similar, *i.e.,* they belong to the same 1-WL class. Here $\neg\mathsf{1WL}(x)$ captures model-level topological similarity across vertices with negative GNN predictions. The 1-WL test can be computed in $O((|V| + |E|) \log |V|)$ time [32].

B. Rules with Negated Predicates

We next define RxGNNs with $\neg\mathcal{M}(x)$ and $\neg\mathsf{1WL}(x)$.

Predicates. A *logic predicate* of a pattern $Q[\bar{x}, x_0]$ is:

$$p ::= x.A \oplus y.B \mid x.A \oplus c \mid \neg\mathsf{1WL}(x).$$

Here $\oplus \in \{=, \neq, <, \leq, >, \geq\}$; variables $x, y \in \bar{x}$; c is a constant; A, B are attributes; and $\neg\mathsf{1WL}(x)$ is the 1-WL predicate.

RxGNNs. Given a GNN $\mathcal{M}$, a *Rule φ for eXplaining $\mathcal{M}$* is

$$Q[\bar{x}, x_0](X \rightarrow \neg\mathcal{M}(x_0)),$$

where $Q[\bar{x}, x_0]$ is a graph pattern and X is a conjunction of its predicates. We refer to Q and $X \rightarrow \neg\mathcal{M}(x_0)$ as the *pattern* and *dependency* of φ, respectively. In a dependency, X is its *precondition* and $\neg\mathcal{M}(x_0)$ is its *consequence*. We focus on RxGNNs whose consequence involves the same classifier $\mathcal{M}$, referred to as *RxGNNs pertaining to $\mathcal{M}$*.

One can plug any *pretrained* GNN classifier $\mathcal{M}$ into RxGNNs as a predicate, such as GCN [60], GAT [55] or GIN [61], as long as $\mathcal{M}$ is *fixed* (its architecture and weights no longer change), and *deterministic* (it produces the same output for the same input) [62], [26], [19], [16].

Intuitively, the pattern $Q[\bar{x}, x_0]$ captures the topological structures around vertex variable x_0 that $\mathcal{M}$ inspects, while the precondition X encodes interactions among vertex features. Together, Q and X explain why $\mathcal{M}(x_0)$ makes its prediction. In particular, $\neg\mathsf{1WL}(x)$ leverages the 1-WL test to explain how GNNs classify entities. We separate Q and X to (a) visualize entity topology via Q, and (b) accelerate evaluation by exploiting the locality of pattern matching.

Remark. RxGNNs can be naturally adapted to explain positive predictions by inverting the semantics of negated predicates. We focus on negative ones in this paper for clarity.

Example 5: Below are RxGNNs with patterns in Figure 2(b).

(1) An RxGNN φ_1 is given in Example 1 to explain why $\mathcal{M}_1$ predicts that the job application of x_0 is declined. It extracts important features from applicants' background.

(2) $\varphi_2 = Q_2[\bar{x}, x_0](X_2 \rightarrow \neg\mathcal{M}_2(x_0))$, where X_2 is $\neg\mathsf{1WL}(x_1) \wedge x_3.\mathsf{content} = x_4.\mathsf{content} \wedge x_5.\mathsf{account\_age} < 24\text{h} \wedge x_6.\mathsf{account\_age} < 24\text{h} \wedge x_7.\mathsf{account\_age} < 24\text{h}$. It says that $\mathcal{M}_2$ flags a user account x_0 for suspension since (a) x_0 behaves like a suspended account x_1 (which is in the negative class by 1-WL test), *e.g.,* both post repetitive contents such as bot-generated spammy "Great post!", and they have been accessed from the same IP address; and (b) many followers of x_0 are created within 24 hours (bot-like accounts). The 1-WL predicate assesses whether vertices x_0 and x_1 are structurally similar, where $\mathcal{M}_2$ already predicts negative on x_1.

(3) $\varphi_3 = Q_3[\bar{x}, x_0](X_3 \rightarrow \neg\mathcal{M}_3(x_0))$, where X_3 is $x_0.\mathsf{debt\_to\_income} \geq 40\% \wedge x_3.\mathsf{default\_history} = \mathsf{yes} \wedge \neg\mathsf{1WL}(x_4) \wedge x_2.\mathsf{credit} \leq x_4.\mathsf{credit} \wedge x_1.\mathsf{balance} < 10\text{K}$. It says that $\mathcal{M}_3$ suggests to deny the loan of x_0 because (a) x_0 has high debt-to-income ratio ($\geq 40\%$) and x_0 is employed by a company defaulted on a loan, (b) an immediate relative x_4 of x_0 is put in the negative class by 1-WL test, (c) the co-applicant (spouse) x_2 of x_0 has worse credit than x_4 (in negative class) and their joint account has low balance (< 10K). □

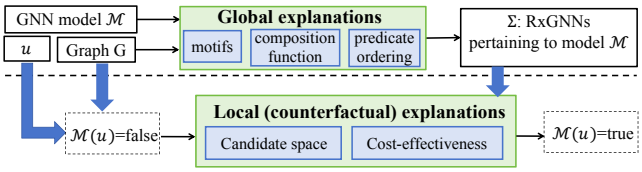


Fig. 3: The framework of NEX

Discussion. Rules on graphs have been studied for diverse tasks: (a) association analysis, *e.g.,* GPARs [63], GARs [52] and TACOs [64]; (b) data cleaning, *e.g.,* GFDs [51] and GCRs [65]; (c) enhancing ML recommendations, *e.g.,* TIEs [66]; and (d) factual explanation, *e.g.,* REPs [38]. Unlike these prior rules, RxGNNs exhibit three unique properties essential for expressive counterfactual explanations (see [36] for an expressivity comparision with graph and tabular rules):

(1) A full range of predicates. RxGNNs support not only constant, variable, and ML predicates, but also (a) limited negation, including $\neg\mathcal{M}(x_0)$, $\neg\mathsf{1WL}(x)$, and $\neg(x.A \oplus y.B)$ via the dual of $\oplus$ (*e.g.,* $\geq$ vs. $<$, $=$ vs. $\neq$), which is crucial for modeling conditions that flip predictions; and (b) the 1-WL test as a predicate, whose computation explicitly simulates message passing through color refinement. Since 1-WL is at least as expressive as GNN classifiers [31], [32], [67], [59], RxGNNs can faithfully explain $\mathcal{M}$'s predictions.

(2) General patterns. RxGNNs are defined on general graph patterns, which capture richer and more complex topological structures than the star-shaped patterns used in [65], [66], [38].

(3) Negated consequence. RxGNNs take $\neg\mathcal{M}(x_0)$ as consequences, enabling counterfactual reasoning.

Semantics. Consider a graph G, an RxGNN $\varphi = Q[\bar{x}, x_0](X \rightarrow \neg\mathcal{M}(x_0))$, and a match h of the pattern Q in graph G.

We say that a match h satisfies a predicate p of $Q[\bar{x}, x_0]$, written $h \models p$, if: (a) $p = x.A \oplus y.B$: vertices $h(x)$ and $h(y)$ carry attributes A and B, and $h(x).A \oplus h(y).B$ (similarly for $x.A \oplus c$); (b) $p = \neg\mathsf{1WL}(x)$: the 1-WL test places $h(x)$ in the same class as some vertex v with $\mathcal{M}(v) = \mathsf{false}$; and (c) $p = \neg\mathcal{M}(x_0)$: $\mathcal{M}$ predicts negative at $h(x_0)$.

We write $h \models X$ if h satisfies all predicates in X, and $h \models \varphi$ if $h \models X$ implies $h \models \neg\mathcal{M}(x_0)$. For a vertex $u \in G$, h is a *witness* of φ *pivoted* at u if $h \models \varphi$ and $h(x_0) = u$. RxGNN φ is *applicable at* u if there exists such a witness.

A graph G *satisfies* φ, written $G \models \varphi$, if every match h of $Q[\bar{x}, x_0]$ in G satisfies φ whenever $h \models X$. For a set Σ of RxGNNs, $G \models \Sigma$ means $G \models \varphi$ for all $\varphi \in \Sigma$.

III. Global and Local Explanations

This section overviews NEX, which provides both global and local explanations using RxGNNs (see Figure 3).

Global explanations. Given a GNN $\mathcal{M}$ and a graph G, NEX discovers a set Σ of RxGNNs for $\mathcal{M}$'s negative predictions. Each RxGNN is composed of important substructures, or *motifs*, from G. Prior studies [17] show that such motifs

can drive predictions; for instance, the Nitrobenzene structure often indicates mutagenicity. Leveraging these signals, NEX guides discovery so that rules in Σ capture instance-independent behaviors of $\mathcal{M}$, reflect different cases via the 1-WL test, and highlight the most discriminative structures and features. Thus, Σ serves as the global explanation of $\mathcal{M}$.

Local explanations. For each vertex u with $\mathcal{M}(u) = \mathsf{false}$, we derive a counterfactual explanation using Σ. Since all RxGNNs in Σ share the consequence $\neg\mathcal{M}(x_0)$, they are non-conflicting. Moreover, Σ is discovered offline and can be reused to generate local explanations for different inputs u.

Counterfactual explanations. Given a witness h of φ mapping variable x to v in G, a *perturbation* is either (a) a *feature modification*, changing an attribute of v so that a predicate $p \in X$ involving x is no longer satisfied, or (b) an *edge removal*, deleting an incident edge of v so that h no longer matches the pattern Q. We treat these two perturbations differently, since they affect predictions in distinct ways. We avoid inserting new edges, so as not to introduce nonexistent relationships.

A *counterfactual explanation* for $\neg\mathcal{M}(u)$ is the minimum set $\mathbb{O}$ of perturbations such that in the updated graph, denoted as $G \ominus \mathbb{O}$, no $\varphi \in \Sigma$ is applicable at u. Intuitively, $\mathbb{O}$ specifies the *necessary changes* to flip $\mathcal{M}(u)$ from negative to positive.

Example 6: In Example 5, RxGNN φ_3 forms part of a global explanation for unsuccessful loan applications by capturing employment ties (*e.g.,* $(x_0, \mathsf{employed\_by}, x_3)$), credit history (*e.g.,* $x_2.\mathsf{credit}$), and social connections (*e.g.,* $(x_0, \mathsf{child}, x_4)$).

For local explanation, for a witness h_3 of φ_3 at u, flipping $\mathcal{M}_3(u)$ from negative to positive can be achieved by, *e.g.,* paying off credit card debt and increasing savings, thus invalidating the high debt-to-income ratio and low balance. □

Extension. NEX readily extends to multi-class classifiers. Specifically, consequence $\neg\mathcal{M}(x_0)$ in RxGNNs can be replaced with an ML predicate $\mathcal{M}(x_0) = i$, where $i \in [1, K]$ denotes the class predicted for x_0 by $\mathcal{M}$ (note that $K = 2$ recovers binary classification). In this setting, counterfactual explanations aim to change the predicted class of x_0 from i to another class. We will evaluate this extension in Section VI.

IV. Model-Guided Rule Discovery

To mine RxGNNs as global explanations, we develop a model-guided rule discovery framework based on *motifs*. We first formulate the discovery problem (Section IV-A), then describe how to identify and compose motifs (Section IV-B), and finally present the discovery algorithm (Section IV-C).

A. The Discovery Problem

We start with criteria to evaluate the quality of RxGNNs.

Support. This measures how often an RxGNN φ can be applied in graph G. Formally, the *support* of φ in G is defined as:

$$\mathsf{supp}(\varphi, G) = \|\mathcal{Q}(\bar{x}, x_0, G, X \wedge \neg\mathcal{M}(x_0))\|,$$

where $\mathcal{Q}(\bar{x}, x_0, G, X \wedge \neg\mathcal{M}(x_0))$ denotes the set of vertices $h(x_0)$ for all matches h of Q in G such that $h \models X \wedge \neg\mathcal{M}(x_0)$. Intuitively, the higher $\mathsf{supp}(\varphi, G)$ is, the more frequent φ can be applied to G. For an integer σ, an RxGNN φ is *σ-frequent* if $\mathsf{supp}(\varphi, G) \geq \sigma$. When $\varphi = Q[\bar{x}, x_0](\emptyset \rightarrow \neg\mathcal{M}(x_0))$ (*i.e.,* X is empty), we denote $\mathsf{supp}(\varphi, G)$ by $\mathsf{supp}(Q, G)$ for simplicity.

Anti-monotonicity. Given two RxGNNs $\varphi = Q[\bar{x}, x_0](X \rightarrow \neg\mathcal{M}(x_0))$ and $\varphi' = Q'[\bar{x}, x_0](X' \rightarrow \neg\mathcal{M}(x_0))$, we say that pattern Q *subsumes* Q', denoted as $Q \ll Q'$, if for any graph G, if Q' has a match in G, then Q also has a match in G, *e.g.,* Q removes edges or vertices from Q'.

Similarly, we say that RxGNN φ *subsumes* φ', denoted as $\varphi \preceq \varphi'$, if $Q \ll Q'$ and $X \subseteq X'$, *i.e.,* φ has a less restrictive pattern and precondition compared with φ'. This results in the well-known *anti-monotonicity* property, *i.e.,* if $\varphi \preceq \varphi'$, then for any graph G, $\mathsf{supp}(\varphi, G) \geq \mathsf{supp}(\varphi', G)$.

Confidence. It measures how strong the connection between precondition X and negative prediction is, formally defined as:

$$\mathsf{conf}(\varphi, G) = \frac{\mathsf{supp}(\varphi, G)}{\|\mathcal{Q}(\bar{x}, x_0, G, X)\|}.$$

For a threshold δ, an RxGNN φ is *δ-confident* if $\mathsf{conf}(\varphi, G) \geq \delta$. If X of φ is empty, we denote $\mathsf{conf}(\varphi, G)$ by $\mathsf{conf}(Q, G)$. Note that confidence does not have the anti-monotonicity.

RxGNNs of interest. Given a GNN-based model $\mathcal{M}$ and a graph G, denote by Σ_A the set of all σ-frequent and δ-confident RxGNNs on G pertaining to $\mathcal{M}$.

A *cover* Σ of Σ_A is a minimal subset satisfying: (a) every rule φ in Σ_A is logically implied by Σ (written $\Sigma \models \varphi$, *i.e.,* for all graphs G', if $G' \models \Sigma$ then $G' \models \varphi$); and moreover, (b) no rule φ in Σ is redundant (*i.e.,* $\Sigma \setminus \{\varphi\} \not\models \varphi$). Thus, Σ includes only nontrivial and non-redundant RxGNNs.

We prune redundant rules and return a non-redundant *cover* Σ of Σ_A using standard implication-based techniques (see [36]). Moreover, we restrict RxGNNs to those composed from a set $\mathcal{S}_\mathcal{M}$ of *motifs* that play essential roles in predictions.

Motif. We define a motif as a graph pattern $\pi[\bar{x}, x_0]$ with a designated vertex x_0 that occurs more often in the negative class than in the positive class of G. More formally, given a threshold η, $\pi[\bar{x}, x_0]$ is a motif if $\frac{n_t}{n_f} < \eta$, where n_t (resp. n_f) is the number of vertices $h(x_0)$ for matches h of $\pi[\bar{x}, x_0]$ in graph G with $\mathcal{M}(h(x_0)) = \mathsf{true}$ (resp. false). For an integer k, we say that a motif is *k-bounded* if $|\bar{x}| \leq k$.

We *compose* multiple motifs into a graph pattern $Q[\bar{x}, x_0]$ using a specified *composition function* $\otimes(\cdot)$. Given a set S of motifs, $\otimes(S)$ denotes the set of patterns composed from S.

Model-guided discovery. The problem is as follows.

- *Input*: A graph G, a GNN model $\mathcal{M}$, a set $\mathcal{S}_\mathcal{M}$ of k-bounded motifs, a composition function $\otimes(\cdot)$, a maximum number N of pattern vertices, a support threshold σ and a confidence threshold δ.
- *Output*: A cover Σ of RxGNNs Σ_A such that for each $\varphi \in \Sigma_A$, (a) its pattern $Q[\bar{x}, x_0]$ is composed from motifs in $\mathcal{S}_\mathcal{M}$ via $\otimes$, *i.e.,* $Q \in \cup_{S \subseteq \mathcal{S}_\mathcal{M}} \otimes(S)$, (b) $|\bar{x}| \leq N$, (c) $\mathsf{supp}(\varphi, G) \geq \sigma$, and (d) $\mathsf{conf}(\varphi, G) \geq \delta$.

Here N primarily controls the discovery cost. While any

Input: G, $\mathcal{M}$, σ as above, a maximum size k and a threshold η.
Output: A set $\mathcal{S}_{\mathcal{M}}$ of motifs.
1. $\mathcal{S}_{\mathcal{M}} := \emptyset$; $\mathcal{G} := \emptyset$;
2. **for** each vertex $v \in G$ such that $\mathcal{M}(v) = \mathsf{false}$ **do**
3. $\quad G_{\mathsf{sub}}(v) := \mathsf{extractor}(G, v)$; $\mathcal{G} := \mathcal{G} \cup \{G_{\mathsf{sub}}(v)\}$;
4. $\mathcal{F} := \mathsf{generator}(\mathcal{G}, \sigma, k)$;
5. **for** each candidate pattern $\pi[\bar{x}, x_0] \in \mathcal{F}$ **do**
6. $\quad n_t := \|\mathcal{Q}(\bar{x}, x_0, G, \mathcal{M}(x_0))\|$; $n_f := \|\mathcal{Q}(\bar{x}, x_0, G, \neg\mathcal{M}(x_0))\|$;
7. $\quad$ **if** $\frac{n_t}{n_f} < \eta$ and $\nexists \pi' \in \mathcal{S}_{\mathcal{M}}$ such that $\pi' \ll \pi$ **then**
8. $\quad\quad \mathcal{S}_{\mathcal{M}} := \mathcal{S}_{\mathcal{M}} \cup \{\pi[\bar{x}, x_0]\}$;
9. **return** $\mathcal{S}_{\mathcal{M}}$;

Fig. 4: Algorithm getMotifs

choice of $\mathcal{S}_{\mathcal{M}}$ and $\otimes(\cdot)$ is possible, we will present in Section IV-B our tailored strategies for identifying and composing motifs specifically for explaining negative predictions of $\mathcal{M}$.

B. Motif Identification and Composition

We next devise strategies to (a) identify the motif set $\mathcal{S}_{\mathcal{M}}$ from $\mathcal{M}$ and G, and (b) define the composition function $\otimes(\cdot)$. Both tasks are challenging for the following reasons.

(1) Exponential motifs. Identifying discriminative structures decisive for $\mathcal{M}$'s predictions can be exponential in the worst case. Thus we need to strike a balance between the cost of generation and the discriminability of the motifs identified.

(2) Excessive compositions. Motif composition is combinatorial. For instance, if we use a naive $\otimes_{\mathsf{all}}(\cdot)$ and there are M pairs of mergable vertices in S, $\otimes_{\mathsf{all}}(S)$ can have $O(2^M)$ patterns, most of which are redundant or useless.

To tackle these, we develop a bounded motif identification strategy with pre-pruning and an effective composition function via reinforcement learning. Below we elaborate them.

Motif identification. To identify useful motifs for RxGNNs, we adopt an "explain-and-summarize" paradigm: we first extract high-quality subgraphs that best explain the GNN model, and then summarize them into motifs, as follows.

(1) Explain: In the "explain" step, we extract a *concise* explanation subgraph $G_{\mathsf{sub}}(v)$ for each negative vertex v in G (*i.e.,* $\mathcal{M}(v) = \mathsf{false}$), such that $\mathcal{M}$ could *reproduce* its prediction in $G_{\mathsf{sub}}(v)$. The subgraphs for all negative vertices in G are stored in a set $\mathcal{G}$. To this end, we employ a primitive operator, called extractor, to extract such subgraphs.

(2) Summarize: We abstract common structures as motifs from subgraphs in $\mathcal{G}$, such that each motif is σ-frequent and k-bounded. To this end, we use a generator, another primitive operator, which produces candidate patterns from $\mathcal{G}$ (satisfying σ-frequency and k-boundedness) for later verification.

Algorithm. Given as input a graph G, a GNN model $\mathcal{M}$, a support threshold σ, the maximum size k and a threshold η, we outline the algorithm in Figure 4. Initializing $\mathcal{S}_{\mathcal{M}} = \emptyset$ (line 1), we extract a subgraph for each negative vertex $v \in G$ via the primitive extractor (lines 2-3). From these subgraphs, we mine σ-frequent and k-bounded patterns π via the primitive generator (line 4). Each candidate π is then evaluated by comparing its occurrence across positive and negative classes (lines 5-8). More specifically, we calculate the ratio $\frac{n_t}{n_f}$ of π, where n_t and n_f are defined in Section IV-A. We add π to $\mathcal{S}_{\mathcal{M}}$ if $\frac{n_t}{n_f} < \eta$ and π is not subsumed by any existing π' in $\mathcal{S}_{\mathcal{M}}$ (lines 7-8); intuitively, this is to avoid redundant motifs in $\mathcal{S}_{\mathcal{M}}$.

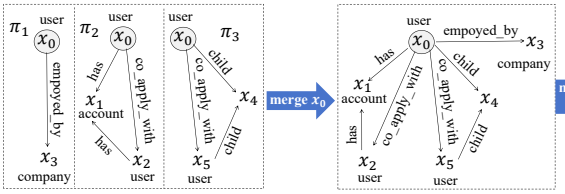


Fig. 5: An example of how Q_3 is composed

Implementation of primitive operators. To extract concise subgraphs that retain critical message flows that drive negative predictions, we apply a graph mask over edges following [68], to filter out unimportant edges. For motif generation, we adopt well-known GASTON [69] to mine σ-frequent, k-bounded patterns from the masked subgraphs, for its efficiency.

By our motif identification strategy, we focus on mining only σ-frequent and k-bounded motifs (see [36] for details).

Discussion. Various techniques have been developed for mining frequent structures in graphs [70], [71], [72], [73], [74], [75], [76], [77] (see survey [78]). However, graph mining alone cannot explain GNNs. We adopt GASTON [69] for its efficiency, though advanced mining algorithms can also be integrated (see [36] for empirical comparisons).

Composition function. We first define the composition of two motifs. Given $\pi_1[\bar{x}_1, x_0] = (V_1, E_1, L_1)$ and $\pi_2[\bar{x}_2, x_0] = (V_2, E_2, L_2)$, a pattern $Q[\bar{x}, x_0] = (V_Q, E_Q, L_Q)$ is composed from π_1 and π_2 via $\otimes(\cdot)$ if V_Q (resp. E_Q) is obtained by merging vertices (resp. edges) in $V_1 \cup V_2$ (resp. $E_1 \cup E_2$), while L_Q inherits label assignments from π_1 and π_2. Note that only vertices with the same label may be merged.

Given a set S of motifs from $\mathcal{S}_{\mathcal{M}}$ and a support threshold σ, our composition operator $\otimes(\cdot)$ returns a pattern $Q^* = \arg\max\{\mathsf{conf}(Q) \mid Q \in \otimes_{\mathsf{all}}(S)$ and Q is a σ-frequent pattern$\}$ if such a Q exists. To avoid excessive compositions, we learn a robust policy that heuristically guides the process, retaining only the most promising candidate in $\otimes(S)$.

Overview. We guide motif composition with deep Q-learning (DQN) [34], learning a policy that composes σ-frequent patterns with maximum confidence. Given a set S of motifs from $\mathcal{S}_{\mathcal{M}}$, patterns are then generated by iteratively merging vertices under this policy, reducing the large search space.

DQN *training*. We next describe how the policy is learned via DQN, a reinforcement learning method with a specified reward function. Here the reward is defined as the confidence gain between patterns before and after composition: for a current pattern Q_i and a new pattern Q_{i+1} obtained by merging a pair of vertices (v, v') with the same label in Q_i, the reward r is:

$$r = \begin{cases} -1 & \text{if } \mathsf{supp}(Q_{i+1}, G) < \sigma, \\ \mathsf{conf}(Q_{i+1}, G) - \mathsf{conf}(Q_i, G) & \text{if } \mathsf{supp}(Q_{i+1}, G) \geq \sigma. \end{cases}$$

We also adopt a special action [end] to terminate when $\mathsf{supp}(Q, G) < \sigma$. Further details of DQN are provided in [36].

Algorithm. Given the trained policy, $\otimes(S)$ begins with an initial pattern Q_0 (formed by merging all x_0 vertices of motifs in S into one), and then iteratively merges vertices with the

highest reward until the termination condition is met.

Example 7: Figure 5 illustrates how pattern Q_3 in Figure 2(b) is generated. We first identify three 3-bounded motifs, π_1, π_2, and π_3, capturing x_0's employment ties, financial status, and social connections, respectively. These motifs are then composed by merging all designated x_0 vertices into one, followed by merging user vertices x_2 and x_5. □

C. Parallel Discovery Algorithm

We present RxGNNMiner, our parallel discovery algorithm. Its key novelties are: (a) leveraging motifs and composition functions to generate expressive graph patterns, and (b) introducing a predicate ordering scheme for precondition generation that enables efficient multi-thread parallelism.

Algorithm. As shown in Figure 6, RxGNNMiner first initializes the set $\mathcal{C}$ of graph patterns and the result set Σ to be empty (line 1). Then it composes the motifs in $\mathcal{S}_\mathcal{M}$ into patterns in $\mathcal{C}$ using the composition function $\otimes(\cdot)$ (line 2), via procedure genPatterns (lines 15-18). Given as input a set S_{sel} of selected motifs, a set S_{re} of remaining motifs, function $\otimes(\cdot)$ and threshold σ, genPatterns conducts depth-first search. In each round, we check whether $Q = \otimes(S_{\mathsf{sel}})$ is σ-frequent (line 15). If so, we add Q into $\mathcal{C}$ (line 16) and recursively call genPatterns by adding a motif π from S_{re} to S_{sel} to find more frequent patterns (lines 17-18). If Q is not σ-frequent, the recursion stops and the current $\mathcal{C}$ is directly returned since exploring more motifs can only lower the support by its anti-monotonicity. In the first call of genPatterns, we set $S_{\mathsf{sel}} = \emptyset$ and $S_{\mathsf{re}} = \mathcal{S}_\mathcal{M}$ (line 2).

For each resulting Q in $\mathcal{C}$, we compute X in a levelwise manner (lines 3-13). More specifically, we first generate the set $\mathcal{P}$ of potential predicates guided by $\mathcal{M}$ via procedure genPredicates (line 4, see below). Then we start with $X = \emptyset$ (line 5) and iteratively expand X with more predicates in $\mathcal{P}$ (lines 6-13), by maintaining a work queue Φ (line 5). Each pair $\langle X, \mathcal{P}\rangle$ in Φ forms a computational task. If $\varphi = Q[\bar{x}, x_0](X \to \neg\mathcal{M}(x_0))$ is σ-frequent and δ-confident, it is valid to be included in Σ and there is no need to expand X since we only want "minimal" RxGNNs (lines 8-9). Otherwise, if φ is σ-frequent but not δ-confident, we expand X by adding a new predicate from $\mathcal{P}$ into X (lines 10-13), leading to a new task to be added to Φ. Unlike conventional levelwise expansion [79], we order predicates in $\mathcal{P}$ based on perplexity-based scores (line 11, see below). When φ is not σ-frequent, we can directly skip its subsequent expansion, since all RxGNNs expanded from φ are not σ-frequent by the anti-monotonicity. Finally, when all tasks in Φ are done, a set of Σ of non-redundant is returned via *cover computation* [79] (line 14, see below).

Example 8: Assume that for pattern Q_3, the set $\mathcal{P}$ has all predicates in X_3 (Example 5), ordered by the occurrence. Initially, $X = \emptyset$ and we check $\varphi' = Q_3(\emptyset \to \neg\mathcal{M}_3(x_0))$. If φ' is σ-frequent but not δ-confident, we expand precondition X with the next predicate x_0.debt_to_income $\geq 40\%$ and check the rule again. This process continues until φ_3 is found. □

Procedure genPredicates. To avoid exhaustively generating all possible predicates $\mathcal{P}$ for a given pattern Q, we only keep predicates p closely related to $\neg\mathcal{M}$, *i.e.,* p is kept in $\mathcal{P}$ if it is satisfied more frequently in the matches h of Q from negative classes than positive classes, *w.r.t.* a threshold (*e.g.,* η).

Algorithm RxGNNMiner
Input: G, $\mathcal{M}$, $\mathcal{S}_\mathcal{M}$, $\otimes(\cdot)$, N, σ, δ as stated above.
Output: A set Σ of RxGNNs pertaining to model $\mathcal{M}$.
1. $\mathcal{C} := \emptyset$; $\Sigma := \emptyset$;
2. $\mathcal{C} :=$ genPatterns$(\emptyset, \mathcal{S}_\mathcal{M}, \mathcal{C}, \otimes, \sigma)$; /* Pattern generation*/
3. **for** each pattern Q in $\mathcal{C}$ **do** /* Precondition generation*/
4. $\quad\mathcal{P} :=$ genPredicates$(Q, \mathcal{M})$;
5. $\quad\Phi :=$ an empty work queue; $X := \emptyset$, Φ.add$(\langle X, \mathcal{P}\rangle)$;
6. $\quad$**while** $\Phi \neq \emptyset$ **do**
7. $\quad\quad\langle X, \mathcal{P}\rangle := \Phi$.pop(); $\varphi := Q[\bar{x}, x_0](X \to \neg\mathcal{M}(x_0))$;
8. $\quad\quad$**if** φ is σ-frequent and δ-confident **then** /*Checked at each thread*/
9. $\quad\quad\quad\Sigma := \Sigma \cup \{\varphi\}$;
10. $\quad\quad$**elseif** φ is σ-frequent but not δ-confident **then**
11. $\quad\quad\quad$**for** each p in $\mathcal{P}$, ordered by their ranking scores **do**
12. $\quad\quad\quad\quad\mathcal{P}_{\mathsf{new}} :=$ the set of all predicates in $\mathcal{P}$ ordered after p;
13. $\quad\quad\quad\quad\Phi$.add$(\langle X \cup \{p\}, \mathcal{P}_{\mathsf{new}}\rangle)$;
14. **return** cover(Σ);

Procedure genPatterns
Input: A set S_{sel} of selected motifs, a set S_{re} of remaining motifs that can be selected, the current set $\mathcal{C}$ of graph patterns, $\otimes(\cdot)$, and σ.
Output: An updated set $\mathcal{C}$ of graph patterns.
15. **if** $Q = \otimes(S_{\mathsf{sel}})$ is σ-frequent **then**
16. $\quad\mathcal{C} := \mathcal{C} \cup \{Q\}$;
17. $\quad$**for** each π in S_{re} **do**
18. $\quad\quad\mathcal{C} := \mathcal{C} \cup$ genPatterns$(S_{\mathsf{sel}} \cup \{\pi\}, S_{\mathsf{re}} \setminus \{\pi\}, \mathcal{C}, \otimes, \sigma)$;
19. **return** $\mathcal{C}$;

Fig. 6: Algorithm RxGNNMiner

Predicate ordering. We distribute workloads evenly across processors for parallel processing [80]. Following [38], multiple threads independently handle tasks $\langle X, \mathcal{P}\rangle$ for precondition generation. Since expansion halts once a non-σ-frequent RxGNN φ is reached, predicate order critically affects efficiency. This motivates us to design an ordering strategy that expands cheap and discriminative predicates first.

To assess how well a predicate discriminates negative predictions $\neg\mathcal{M}(x_0)$ at x_0, we leverage large language models (LLMs) for their rich prior knowledge and contextual understanding. For each predicate p, we define its perplexity as:

$$\mathsf{PPL}(p) \propto \exp\left(-\log P_{\mathrm{LLM}}(\mathsf{nlp}(p) \mid \mathsf{nlp}(Q))\right),$$

where $\mathsf{nlp}(\cdot)$ is natural language description of Q or p, and P_{LLM} is a probability given by LLMs, indicating the likelihood that appending $\mathsf{nlp}(p)$ into $\mathsf{nlp}(Q)$ yields a statement consistent with $\neg\mathcal{M}(x_0)$. Intuitively, $\mathsf{PPL}(p)$ measures the correlation between a predicate p and $\neg\mathcal{M}(x_0)$. A lower $\mathsf{PPL}(p)$ means the LLM is more confident in this correlation, and thus p should be prioritized during processing.

Example 9: Consider two predicates p_1: x_0.debt_to_income $\geq 40\%$ and p_2: x_0.gender $=$ Male. The simplified natural language description of Q_3 appended with p_1 (resp. p_2) is T_1 = "Given the topology in Q_3, x_0's loan is denied since his debt to income ratio exceeds 40%" (resp. T_2 = "…, x_0's loan is denied since he is a male"). Since T_1 is more reasonable, the LLM will assign lower perplexity to p_1. □

We also factor in validation cost. For a predicate p, its ranking score is defined as $\mathsf{PPL}(p) \times \mathsf{time}(p)$, where $\mathsf{time}(p)$ is the estimated evaluation time on sample data. Predicates are processed in ascending order of this score, favoring those strongly correlated with $\neg\mathcal{M}$ while penalizing costly ones.

Cover computation. We compute a cover Σ of σ-frequent and δ-confident RxGNNs by adapting the implication analysis and parallel techniques [79] to RxGNNs (see details in [36]).

Remark. To choose parameters (thresholds σ, δ and motif size k), users may rely on data statistics for initial settings, enabling fast discovery of a preliminary rule set. These parameters can then be iteratively fine-tuned using incremental rule discovery [81], which naturally extends to RxGNNs (see [36]).

Complexity. Algorithm RxGNNMiner takes at most $O(\frac{1}{n} \times |\mathcal{S}_\mathcal{M}|^K \sum_{X \in \mathsf{pow}(\mathcal{P})} |G|^{|X|})$ time with n processors, where K is the maximum number of motifs in $\mathcal{S}_\mathcal{M}$ that can compose a σ-frequent pattern Q ($K \ll |\mathcal{S}_\mathcal{M}|$), $\mathcal{P}$ is the set of all potential predicates defined on a pattern Q and $\mathsf{pow}(\mathcal{P})$ is the power set of $\mathcal{P}$. This is because (a) the computation is dominated by the validation of support and confidence, where the computational tasks are evenly distributed across n processors, and (b) it examines $\mathsf{pow}(\mathcal{P})$ to generate X's for each Q at worst. This is the inherent cost for rule discovery [82], [83], [79], [80]. This said, we show that RxGNNMiner is parallelly scalable [33], reducing runtime nearly n-fold with n processors (see [36] for details; the space cost is also given in [36]).

V. Local Counterfactual Explanations

This section presents our algorithm for identifying counterfactual explanations of negative GNN predictions.

We begin by formulating the problem as follows.

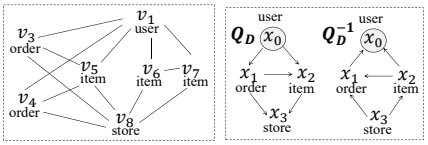
- *Input*: A graph G, a GNN-based classifier $\mathcal{M}$, the set Σ of RxGNNs, and a negative vertex u in G with $\mathcal{M}(u) = \mathsf{false}$.
- *Output*: The minimum set $\mathbb{O}$ of graph perturbations such that no $\varphi \in \Sigma$ is applicable at u in $G \ominus \mathbb{O}$. That is, after updating G with $\mathbb{O}$, $\mathcal{M}(u)$ is swapped from false to true.

Intuition. A brute-force two-stage approach would (i) compute all witnesses h of each RxGNN $\varphi \in \Sigma$ pivoted at u, and (ii) identify a perturbation set $\mathbb{O}$ such that every h ceases to satisfy X or Q of φ in $G \ominus \mathbb{O}$. However, this is costly, since even checking for the existence of matches of a general graph pattern Q is NP-hard [84]. Instead, we design an *inline* algorithm that identifies perturbations during the matching process itself, thereby reducing time complexity.

A key step in computing matches of $Q[\bar{x}, x_0]$ is vertex filtering, which selects candidate vertices in G for each pattern vertex $x \in \bar{x}$ and discards those that cannot appear in any match. Formally, for a pattern vertex x, let $C_Q(x)$ denote its candidate set in G. Then we extend vertex filtering to RxGNNs by requiring candidates also satisfy X (Section V-A), and develop an algorithm to compute perturbations (Section V-B).

A. Candidate sets for RxGNNs

Before formally defining candidate sets for RxGNNs, we specify the required properties of the candidate set $C_Q(x)$.



(a) Data graph G (b) Pattern and DAG

$x \in \bar{x}$	$C_\varphi(x)$
x_0	v_1
x_1	v_3, v_4
x_2	v_5, v_6, v_7
x_3	v_8

(c) Initial C_φ

$x \in \bar{x}$	$C_\varphi(x)$
x_0	v_1
x_1	v_3, v_4
x_2	v_5
x_3	v_8

(d) Refined C_φ

Fig. 7: Candidate space-based counterfactual explanation

- *General-pattern support*. It supports general patterns Q, not just star patterns or patterns with bounded tree-width [85].
- *PTIME cost*. We want $C_Q(x)$ to be computable in PTIME, even though exact matching of general Q is coNP-hard.
- *Soundness.* The candidate set $C_Q(x)$ is *sound* if for each match h of Q where $h(x) = v$, $v \in C_Q(x)$. As a compromise, $C_Q(x)$ may have *false positives*, *i.e.,* for $v \in C_Q(x)$, there may exist no h of Q such that $h(x) = v$.

A naive $C_Q(x)$ is V, which consists of all vertices in G). However, good $C_Q(x)$ should be tight, with minimal false positives, to concisely capture the search space.

Candidate sets. Along the same line as $C_Q(x)$ for pattern Q, given an RxGNN $\varphi = Q[\bar{x}, x_0](X \to \neg\mathcal{M}(x_0))$ and a vertex u in G, we define $C^u_\varphi(x)$ to be the *candidate set* of vertices in G that can be mapped to x in a witness h of φ pivoted at u (*i.e.,* h is a match of pattern Q, $h \models \varphi$ and $h(x_0) = u$). When u is clear in the context, we simply write $C^u_\varphi(x)$ as $C_\varphi(x)$. Denote the union set of $C_\varphi(x)$ for all $x \in \bar{x}$ as *the candidate space* C_φ. We define *sound* C_φ as above.

Sufficient condition. Inspired by the *emptyset failure* [86], we devise a sufficient condition on sound C_φ to check whether a witness of φ exists at $u \in G$ (*i.e.,* φ is applicable at u).

Lemma 1: *Given a graph G, a vertex u in G, an RxGNN φ, and a sound candidate space C_φ of φ, if there exists a pattern vertex x of φ such that $C_\varphi(x) = \emptyset$, then there exists no witness h of φ pivoted at u in G.* □

Proof. Assume that there is a witness h of φ pivoted at u in G; then, by the soundness of C_φ, $h(x) \in C_\varphi(x)$ for each $x \in \bar{x}$, contradicting to $C_\varphi(x) = \emptyset$ for some x. □

We call a sound C_φ *invalid* if it meets the condition described in Lemma 1. Intuitively, for a vertex $u \in G$, if C_φ is invalid, then φ is not applicable at u. Thus, for each $\varphi \in \Sigma$, we compute a sound C_φ; if it is still valid, we iteratively perturb G (and hence C_φ) until it becomes invalid.

Example 10: Consider a graph G and an RxGNN φ with pattern Q_D in Figures 7(a)-(b) (only vertex labels are shown). Assume that φ has two witnesses pivoted at v_1, *i.e.,* h_1: $(x_0, x_1, x_2, x_3) \mapsto (v_1, v_4, v_5, v_8)$ and h_2: $(x_0, x_1, x_2, x_3) \mapsto (v_1, v_3, v_5, v_8)$. A sound and valid C_φ is given in Figure 7(d), since there exists no x in $\bar{x}$ such that $C_\varphi(x) = \emptyset$. □

New Problem. Based on these, it suffices to study a candidate space-based counterfactual explanation problem as follows.

- *Input*: G, $\mathcal{M}$, Σ, and u with $\mathcal{M}(u) = \mathsf{false}$ as above.
- *Output*: A minimum set $\mathbb{O}$ of perturbations on G such that for each $\varphi \in \Sigma$, C_φ is *invalid* after updating G with $\mathbb{O}$.

Below we present a PTIME method for computing C_φ.

Candidate space construction. We construct a tight C_φ by exploiting *DAG-graph dynamic programming* [86], to check both topological structures (*i.e.,* to match Q) and vertex attributes (*i.e.,* to satisfy X) simultaneously.

Overview. Given a graph G, a vertex u, an RxGNN $\varphi \in \Sigma$, we construct a candidate space C_φ pivoted at u in three steps.

(1) Rooted DAG. We first build a *rooted directed acyclic graph (DAG)* Q_D from the pattern $Q[\bar{x}, x_0]$ of φ, by traversing Q in BFS order from x_0 and directing all pattern edges from upper levels to lower levels. The DAG (a) contains no cycle, and (b) only root x_0 in G_D has no incoming edges (see [36]).

(2) Initialization. We initialize the candidate space C_φ pivoted at u by checking labels, degrees and predicates, *i.e.,* $v \in C_\varphi(x)$ if $L_G(v) = L_Q(x)$, $\mathsf{deg}_G(v) \geq \mathsf{deg}_Q(x)$ and v satisfies all constant/1-WL predicates defined on x in X of φ, where deg is the degree in G/Q; in particular, $C_\varphi(x_0) = \{u\}$.

(3) Refinement. We iteratively refine C_φ via *DAG-graph dynamic programming*, alternating between *forward pass* on Q_D (top-down) and *backward pass* on its reverse Q_D^{-1} (bottom-up), until convergence. Each pass ensures that every vertex in C_φ satisfies both topological and predicate constraints (see [36]).

Example 11: In Example 10, pattern Q_D is itself a DAG; its reverse Q_D^{-1} is shown in Figure 7(b). Figure 7(c) shows the initial C_φ. After refinement, we obtain final C_φ (Figure 7(d)). □

The candidate space C_φ has the following property.

Proposition 2: *Given a vertex u and an RxGNN φ, DAG-graph dynamic programming computes a sound candidate space C_φ in $O(|Q||G|)$ time; that is, for each witness h of φ with $h(x) = v$, v must be in $C_\varphi(x)$.* □

Proof. (1) DAG-graph dynamic program is dominated by pattern matching, which takes $O(|Q||G|)$ time by following [86]. (2) The soundness of C_φ is warranted by the following: (a) the initial candidate space is sound; (b) by induction, an auxiliary set $C'_\varphi(x)$ is computed correctly; and (c) if a vertex v does not satisfy the soundness condition, then the candidate space C_φ is still sound after we remove v from $C_\varphi(x)$. □

B. Algorithm for Counterfactual Explanations

We develop an algorithm, CFE, to identify perturbations.

Intuition. Building on the candidate spaces of RxGNNs, algorithm CFE selects graph perturbations to render each candidate space C_φ *invalid*. As removing one vertex from C_φ triggers refinement and can cascade into further removals, the challenge is to minimize perturbations. To this end, CFE employs a *cost-effectiveness* strategy to decide which vertices to remove.

We proceed as follows: (a) present the perturbation strategy, (b) introduce the cost-effectiveness criterion, and (c) putting these together, present the complete CFE algorithm.

Perturbation strategy. Given $v \in C_\varphi(x)$, there are two types of perturbations that make v no longer map to x in *any* witness.

Feature modification. Note that a GNN typically aggregates more information from hub vertices. If v is a *hub vertex* in graph G (*i.e.,* its degree is above a threshold τ), we modify some important attributes A of v to make at least one predicate involving $x.A$ in X no longer satisfied.

To measure attribute importance, we adopt GINI values [66]; it shows how well X divides the pivots into different predicted classes of $\mathcal{M}$ [87], [88]. Denote the GINI value with (resp. without) $x.A$ in X by $\mathsf{Gini}(X)$ (resp. $\mathsf{Gini}(X_{\mathsf{noA}})$). We define the importance of $x.A$ in X as $\Delta\mathsf{Gini}_{x.A} = \mathsf{Gini}(X_{\mathsf{noA}}) - \mathsf{Gini}(X)$, *i.e.,* the reduction of the GINI value by $x.A$. The larger the value, the more important $x.A$.

Remark. Following [47], we want to ensure the *plausibility* (*i.e.,* make sense in the real world) and *feasibility* (*e.g.,* the *gender* of a user cannot be changed) of feature modification, we reuse the perplexity-based computation in Section IV-C for this purpose (details presented in [36]).

Edge removal. We remove incident edges of v such that all witnesses h with $h(x) = v$ no longer match Q of φ. To illustrate, assume *w.l.o.g.* that (x_p, l, x) is a selected edge in Q. We remove all edges (v_p, l, v) in G where $v_p \in C_\varphi(x_p)$. This strategy is suitable for lower-degree v.

Cost-effectiveness. To decide which variable x and vertex $v \in C_\varphi(x)$ to perturb, we consider both the *cost* and *effectiveness*.

Cost. According to the two perturbation strategies above, the cost of applying a perturbation to vertex v in $C_\varphi(x)$ is:

$$\mathsf{cost}(v) = \begin{cases} \max\limits_{x.A \text{ in } X} \Delta\mathsf{Gini}_{x.A}, & \text{if } X \neq \emptyset \text{ and } \mathsf{deg}_G(v) \geq \tau, \\ \beta|\{(v_p, l, v) \mid v_p \in C_\varphi(x_p)\}|, & \text{otherwise,} \end{cases}$$

where τ is a degree threshold and β is a balancing parameter.

Effectiveness. Removing a vertex v from $C_\varphi(x)$ may trigger cascading removals in other sets $C_\varphi(x')$ ($x \neq x'$) during refinement. We leverage this cascading effect by measuring the effectiveness of removing v on the entire candidate space C_φ, rather than only on the set $C_\varphi(x)$. More specifically, we compare the sizes of $C_\varphi(x)$ for all $x \in \bar{x}$ before and after refinement triggered by removing v from $C_\varphi(x)$.

Formally, we define the *effectiveness* of removing v as:

$$\mathsf{eff}(v) = \alpha^K \big(1 - \min_{x \in \bar{x}} \frac{|C_\varphi^\Delta(x)|}{|C_\varphi(x)|}\big),$$

where v is a K-hop neighbor of the pivot u, α is the hop-decay factor, and C_φ^Δ is the refined C_φ after v is removed. Intuitively, vertices closer to u are more important and thus should be removed with higher priority; the smaller a set $C_\varphi^\Delta(x)$ is, the better the effectiveness is. Taken together, the *cost-effectiveness* of each $v \in C_\varphi$ is $\frac{\mathsf{eff}(v)}{\mathsf{cost}(v)}$.

Algorithm. We present algorithm CFE in Figure 8. It returns a set $\mathbb{O}$ such that for each φ in Σ, C_φ of φ is *invalid* on $G \ominus \mathbb{O}$.

We initialize $\mathbb{O} = \emptyset$ (line 1) and process each $\varphi \in \Sigma$ (lines 2–7). For each φ, we construct its candidate space C_φ at u via procedure constructCand (line 3). While C_φ remains valid (line 4), *i.e.,* no variable $x \in \bar{x}$ has $C_\varphi(x) = \emptyset$, we select a vertex v^* to remove using procedure pickVertex based on cost-effectiveness (line 5). We then remove v^*, refine C_φ (line 6),

Input: A graph G, a model $\mathcal{M}$, a set Σ of RxGNNs and a vertex u.
Output: A set $\mathbb{O}$ of perturbations
1. $\mathbb{O} := \emptyset$;
2. **for** each RxGNN $\varphi = Q[\bar{x}, x_0](X \to \neg\mathcal{M}(x_0))$ in Σ **do**
3. $\quad C_\varphi :=$ constructCand (u, φ, G);
4. $\quad$ **while** C_φ is valid **do**
5. $\quad\quad v^* :=$ pickVertex(C_φ); // pick the vertex to be removed
6. $\quad\quad C_\varphi :=$ refine(C_φ, v^*); // compute the refined C_φ after removing v^*
7. $\quad\quad \mathbb{O} = \mathbb{O} \cup \{$the perturbation that removes v^* from $C_\varphi\}$;
8. **return** $\mathbb{O}$;

Fig. 8: Counterfactual algorithm CFE

and add the corresponding perturbations to $\mathbb{O}$ (line 7). Once all C_φ's are invalid, we return the set $\mathbb{O}$ of perturbations (line 8).

Example 12: Continuing Example 10, we compute candidate space C_φ (Figure 7(d)). Consider v_4 and v_5. If we remove v_4, the resulting C_φ (*i.e.,* $C_\varphi(x_0) = \{v_1\}$, $C_\varphi(x_1) = \{v_3\}$, $C_\varphi(x_2) = \{v_5\}$, $C_\varphi(x_3) = \{v_0\}$) is still valid, leading to additional perturbations to invalidate C_φ. Instead, it is more effective to remove v_5 since it directly invalidates C_φ (*i.e.,* $C_\varphi(x_2) = \emptyset$) and we can finish the processing of φ. □

Procedure pickVertex. To efficiently choose the most cost-effective vertex to remove from C_φ, we cluster vertices in C_φ into groups. Intuitively, vertices in the same group share similar neighborhoods and thus likely induce similar effects when removed. For each group g, we evaluate only a representative vertex v_g to estimate its cost-effectiveness.

More specifically, let g^* be the group whose representative v_{g^*} achieves the highest estimated cost-effectiveness. For each $v \in g^*$, we then compute its exact $\mathsf{eff}(v)$ and $\mathsf{cost}(v)$, and select the vertex v^* with the highest true cost-effectiveness.

Complexity. The time cost of CFE is $O(|\Sigma||C_\varphi||Q||G|(\#\mathsf{group} + |g_{\mathsf{max}}|))$, where #group is the number of groups, and g_{max} is the largest group, since for each φ in Σ, (a) C_φ is computed in $O(|Q||G|)$ time, and (b) $O(|C_\varphi|)$ vertices will be removed from C_φ, where each removal takes $O(\#\mathsf{group} + |g_{\mathsf{max}}|)$ times of refinement to decide.

VI. Experimental Study

Using real-life graphs, we empirically tested the accuracy and efficiency of local and global explanations of NEX.

Experimental setting. We start with the experimental setting.

Datasets. We used four real-life graphs G shown in Table III (cf. [36]): (1) Loan [89], a loan graph used to predict whether a user should be approved for loan; (2) Claim [90], a medical claim network dataset designed for insurance fraud detection; (3) Trans [91], a transactional network for financial fraud detection; and (4) Amazon [92], a multi-class e-commerce graph, where products are labeled with various classes.

GNN models. Following [93], [68], [94], we selected four GNN models for vertex classification on heterogeneous graphs: (1) GCN [60], a standard message-passing graph convolutional network; (2) GAT [55], which uses attention mechanism; (3) GIN [61], which effectively captures graph structural information; and (4) HGT [56], which designs a vertex- and edge-type dependent attention mechanism. Unless stated explicitly, we used GCN as our default GNN model.

Baselines. We compared two baselines for hybrid local and global explanations: (1) GVEX [26] and (2) CFGraph [27].

For local explanations, we tested three more baselines: (3) CF-Exp (*i.e.,* CF-GNNExplainer) [11], (4) CLEAR [14], and (5) InduCE [13]. For global explanations, we additionally tested: (6) XGNN [21], (7) GNNInt (*i.e.,* GNNInterpreter [22]), and (8) LMiner, which mines RxGNNs via traditional levelwise search; we only compared LMiner for efficiency.

We adapted all baselines to vertex classification on heterogeneous graphs and optimized all counterfactual methods to minimize perturbations (see more details in [36]).

Rules. On average, we mined 194, 357, 168 and 562 RxGNNs for 4 GNNs on Loan, Claim, Trans and Amazon, respectively.

Default parameters. We set the support threshold σ (resp. confidence δ) as 3K, 8K, 1K and 1K (resp. 0.7, 0.6, 0.7 and 0.5) on Loan, Claim, Trans and Amazon, respectively. We set (a) $k = 5$ (resp. $N = 15$) for the maximum size of motifs (resp. patterns), (b) $\eta = 0.6$ for motif identification, (c) $\alpha = 0.8$ for hop-decay, (d) $\beta = 0.01$ for the balancing parameter, and (e) $n = 8$ for the number of processors.

Environment. Experiments were conducted on a machine powered by 256GB RAM, 32 processors with Intel(R) Xeon(R) Gold 5320 CPU @2.20GHz and a NVIDIA Tesla V100 GPU with 32 GB memory. Each experiment was run 3 times, and the average is reported here. For limited space, representative results are reported on specific datasets/models.

Metrics. We used three widely-adopted metrics [68], [94], [93], [38] for local explanations. Each metric was evaluated on m randomly selected testing vertices. We set $m = 1000$.

(1) *Fidelity*. Since a counterfactual method perturbs the input graph for a prediction switch at u, the counterfactual fidelity measures the ratio of successful prediction switches over all testing vertices, formally defined as follows:

$$\mathsf{fidel} = \frac{1}{m}\sum\nolimits_u \mathbb{1}(\hat{y}_u \neq y_u),$$

where $\hat{y}_u$ (resp. y_u) is the model prediction on the perturbed $G \ominus \mathbb{O}$ (resp. the original G), and $\mathbb{1}(\cdot)$ returns 1 if $\hat{y}_u \neq y_u$.

(2) *Sparsity*. It measures how minimal the edge perturbations are to swap negative predictions, defined as:

$$\mathsf{spars} = \frac{1}{m}\sum\nolimits_u \frac{\#\mathsf{ptb\_edges}_u}{\#\mathsf{edges}_u},$$

where $\#\mathsf{ptb\_edges}_u$ (resp. $\#\mathsf{edges}_u$) is the number of perturbed edges (resp. edges in L-hop neighborhood graph) of u.

(3) *Feature ratio*. We report the feature ratio, *i.e.,* the average ratio of attributes perturbed in the L-hop neighborhood graph.

For global explanations, we also adopt two widely used metrics [21], [23]: (1) *Overall recognizability*: measuring how well a GNN model $\mathcal{M}$ can recognize its own behavior based on the global explanations $\mathcal{E}$ (*e.g.,* $\mathcal{E} = \Sigma$ in NEX), *i.e.,*

$$\mathsf{recog} = \frac{1}{|\mathcal{E}|}\sum\nolimits_{\mathsf{exp}\in\mathcal{E}} \mathbb{1}(\hat{y}(\mathsf{exp}) = \mathsf{false}),$$

GNN	Method	Loan			Claim			Trans			Amazon		
		fidel	spars	time(s)	fidel	spars	time(s)	fidel	spars	time(s)	fidel	spars	time(s)
GCN	GVEX	0.836	5.6%	367.34	0.615	4.3%	492.65	0.754	3.9%	432.33	0.693	5.3%	2587.5
	CFGraph	0.454	17.1%	3.45	0.441	15.5%	4.22	0.481	10.8%	3.30	0.655	10.2%	2.13
	CF-Exp	0.486	23.1%	1.66	0.383	14.7%	2.75	0.56	8.1%	1.99	0.683	11.1%	1.82
	Clear	0.653	10.2%	10.77	0.517	13.4%	15.54	0.635	7.1%	10.11	0.679	9.7%	24.37
	InduCE	0.687	13.7%	5.31	0.573	8.9%	6.81	0.703	5.3%	4.52	0.684	8.4%	4.69
	NEX	**0.935**	**3.3%**	**0.97**	**0.719**	**2.6%**	**0.74**	**0.809**	**2.2%**	**1.36**	**0.734**	**3.3%**	**1.19**
GAT	GVEX	0.843	5.3%	355.53	0.618	4.5%	485.67	0.748	3.9%	444.27	0.695	4.9%	2629.1
	CFGraph	0.456	18.2%	3.67	0.437	15.9%	4.91	0.466	12.3%	3.16	0.632	10.2%	2.73
	CF-Exp	0.510	18.4%	2.34	0.406	14.5%	3.52	0.555	9.7%	2.58	0.688	10.4%	1.89
	Clear	0.739	10.8%	14.44	0.526	14.7%	20.41	0.674	6.3%	12.48	0.689	9.1%	28.84
	InduCE	0.706	9.7%	5.57	0.619	8.2%	8.74	0.671	5.5%	4.42	0.672	8.1%	4.40
	NEX	**0.936**	**2.9%**	**0.87**	**0.694**	**3.5%**	**0.84**	**0.806**	**3.1%**	**1.51**	**0.729**	**2.9%**	**1.34**
GIN	GVEX	0.855	5.6%	403.43	0.626	4.5%	440.83	0.726	4.1%	499.90	0.685	5.5%	2632.4
	CFGraph	0.462	17.7%	4.25	0.455	14.4%	5.15	0.459	10.8%	4.50	0.651	9.9%	2.17
	CF-Exp	0.508	20.4%	2.57	0.375	16.2%	2.79	0.538	9.7%	2.40	0.680	10.6%	1.93
	Clear	0.663	15.3%	12.93	0.509	10.9%	17.23	0.641	5.4%	11.98	0.676	10.2%	22.68
	InduCE	0.638	13.4%	6.14	0.595	9.6%	8.86	0.629	5.8%	4.95	0.677	8.8%	4.29
	NEX	**0.927**	**3.5%**	**1.00**	**0.690**	**3.1%**	**0.80**	**0.764**	**3.5%**	**1.29**	**0.715**	**3.8%**	**1.14**
HGT	GVEX	0.839	5.1%	588.31	0.611	4.8%	695.64	0.735	3.9%	601.78	0.688	5.3%	3938.9
	CFGraph	0.453	15.6%	5.34	0.449	16.2%	6.20	0.469	13.3%	5.55	0.643	11.5%	3.52
	CF-Exp	0.505	20.5%	2.88	0.405	16.6%	5.22	0.542	9.2%	3.56	0.692	12.3%	2.15
	Clear	0.684	14%	14.43	0.512	10.5%	19.14	0.659	6.8%	14.71	0.695	9.2%	34.98
	InduCE	0.671	12.2%	6.99	0.602	9.2%	10.94	0.685	5.9%	6.27	0.684	9.1%	5.61
	NEX	**0.929**	**3.5%**	**0.92**	**0.693**	**3.3%**	**0.96**	**0.797**	**3.3%**	**1.36**	**0.733**	**3.5%**	**1.38**

TABLE II: Local (counterfactual) explanations

Dataset	Loan	Claim	Trans	Amazon
$\|V\|$	504K	702K	15.4M	1.7M
$\|E\|$	1008K	1674K	12.7M	2.5M
#attr	13	16	10	15
#Vlabels	5	4	2	3
#classes	2	2	2	5

TABLE III: Dataset statistic where Vlabels represents the vertex labels (more in [36])

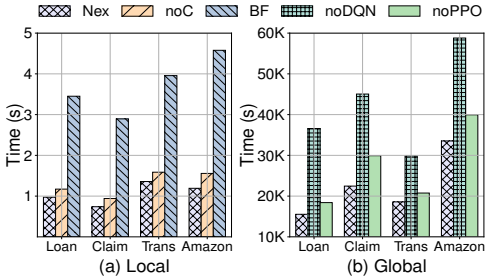
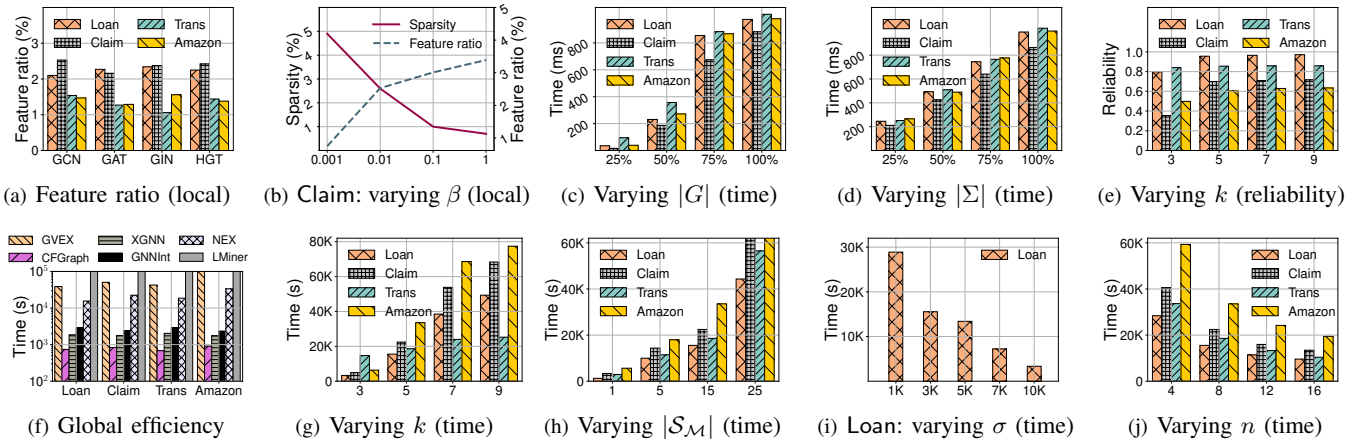


Fig. 9: Ablation study (efficiency)



(a) Feature ratio (local) (b) Claim: varying β (local) (c) Varying $|G|$ (time) (d) Varying $|\Sigma|$ (time) (e) Varying k (reliability)

(f) Global efficiency (g) Varying k (time) (h) Varying $|\mathcal{S}_{\mathcal{M}}|$ (time) (i) Loan: varying σ (time) (j) Varying n (time)

Fig. 10: Performance evaluation

where $\mathsf{exp} \in \mathcal{E}$ is an explanation (*e.g.,* a canonical graph derived from some $\varphi \in \Sigma$), and $\hat{y}(\mathsf{exp})$ is the prediction of $\mathcal{M}$ when exp is fed into it. (2) *Reliability*, evaluating how consistently the explanations apply to testing vertices, *i.e.,* the extent to which $\mathcal{E}$ covers the actual predictions of $\mathcal{M}$:

$$\mathsf{relia} = \frac{|\bigcup_{\mathsf{exp}\in\mathcal{E}} \mathsf{testVertices}(\mathsf{exp})|}{\#\mathsf{totalVertices}},$$

where #totalVertices is the total number of testing vertices, and testVertices (exp) is the set of testing vertices explained by exp (*e.g.,* a vertex v is explained by φ if φ is applicable at v).

Experimental results. We next report our findings.

Exp-1: Local effectiveness. We tested effect of CFE in NEX.

Effectiveness. In Table II, on average, the fidelity of NEX is 8.30%, 60.85%, 53.71%, 24.98% and 20.17% higher than GVEX, CFGraph, CF-Exp, CLEAR and InduceCE on the four datasets, respectively, up to 16.91%, 105.95%, 92.38%, 43.19% and 45.29%, since NEX finds necessary updates based on cost-effectiveness while the baselines depend heavily on hyper-parameters. Although GVEX also achieves reasonable fidelity, it is far less efficient (see Exp-2). The sparsity of NEX outperforms all baselines by 3.25× on average, up to 7.00×.

For feature ratios (Figure 10(a)), NEX perturbs only 1.84% of features on average. In contrast, the five baselines ignore features altogether or apply a uniform feature mask across vertices, treating features across vertices identically.

Varying β. We varied β that controls the perturbation (*i.e.,* modify features or remove edges), from 0.001 to 1. As shown in Figure 10(b), when β gets larger, NEX tends to invalidate C_φ by modifying features only, leading to a lower sparsity and a higher feature ratio, as expected. In contrast, the fidelity of NEX slightly fluctuates and achieves the highest when $\beta =$ 0.01 (not shown), where it strikes a balance.

Exp-2: Local efficiency. We evaluated the efficiency (*i.e.,* the average time for each testing vertex) of CFE in NEX.

Efficiency. In Table II, GVEX is slow since it relies on Jacobian computations to identify influential subgraphs, unlike other methods that only select edges. By contrast, NEX is 7.30× faster than other baselines; *e.g.,* on Trans it averages just 1.38s. This efficiency comes from using C_φ to avoid exhaustive enumeration and selecting perturbations via cost-effectiveness.

Varying $|G|$. We varied the scaling factor of G in Figure 10(c). It takes longer for NEX to handle larger graphs, *e.g.,* on Trans,

GNN	Method	Loan		Claim		Trans		Amazon	
		recog	relia	recog	relia	recog	relia	recog	relia
GCN	GVEX	0.763	0.652	0.843	0.575	0.687	0.672	0.691	0.502
	CFGraph	0.409	0.221	0.276	0.158	0.313	0.496	0.426	0.145
	XGNN	0.373	0.167	0.656	0.174	0.528	0.353	0.418	0.183
	GNNInt	0.755	0.545	0.783	0.218	0.691	0.475	0.697	0.221
	NEX	**1.000**	**0.956**	**0.934**	**0.699**	**0.984**	**0.855**	**0.731**	**0.604**
GAT	GVEX	0.768	0.607	0.857	0.551	0.698	0.688	0.663	0.486
	CFGraph	0.384	0.229	0.255	0.174	0.299	0.51	0.445	0.175
	XGNN	0.414	0.143	0.613	0.176	0.535	0.316	0.47	0.191
	GNNInt	0.735	0.511	0.809	0.113	0.737	0.458	0.719	0.245
	NEX	**1.000**	**0.961**	**0.967**	**0.702**	**0.971**	**0.851**	**0.745**	**0.601**
GIN	GVEX	0.807	0.616	0.833	0.549	0.696	0.654	0.682	0.508
	CFGraph	0.432	0.197	0.273	0.144	0.255	0.539	0.476	0.184
	XGNN	0.410	0.182	0.595	0.154	0.458	0.320	0.459	0.138
	GNNInt	0.787	0.437	0.771	0.166	0.677	0.305	0.688	0.267
	NEX	**1.000**	**0.955**	**0.953**	**0.658**	**1.000**	**0.887**	**0.744**	**0.614**
HGT	GVEX	0.776	0.634	0.861	0.573	0.678	0.694	0.661	0.455
	CFGraph	0.434	0.178	0.212	0.197	0.26	0.474	0.424	0.177
	XGNN	0.398	0.192	0.587	0.160	0.503	0.298	0.429	0.180
	GNNInt	0.765	0.507	0.797	0.188	0.662	0.437	0.694	0.233
	NEX	**1.000**	**0.944**	**0.932**	**0.691**	**0.965**	**0.872**	**0.728**	**0.617**

TABLE IV: Global effectiveness

NEX is 2.84× slower when G changes from 50% to 100%.

Varying $|\Sigma|$. We varied the size of Σ in Figure 10(d). As expected, NEX takes longer with more RxGNNs, yet it remains efficient, handling hundreds within about 1s.

Exp-3: Global effectiveness. We evaluated the impact of the discovered rule set Σ, which serves as global explanations.

Effectiveness. In Table IV, the average recog (resp. relia) of NEX is 15.65%, 167.21%, 80.42% and 17.34% (resp. 31.93%, 253.18%, 302.15% and 171.47%) higher than GVEX, CFGraph, XGNN and GNNInt, respectively, since rule discovery finds discriminative motifs that are mostly responsible for the predictions. In contrast, the baselines may produce patterns that are hardly satisfied in reality.

Varying k. We varied the maximum motif size k from 3 to 9. As shown in Figure 10(e), for $k = 3$, the reliability of NEX on Claim is only 0.355, since the motifs are not sufficiently discriminative. Reliability improves as k increases, with optimal performance observed at $k = 5$.

Exp-4: Global efficiency. We tested runtime of RxGNNMiner.

Efficiency. As shown in Figure 10(f), NEX is at least 2.25× faster than GVEX (which times out on Amazon), though not the fastest overall since it has to identify patterns, features and their conditions. In contrast, the baselines generate explanations without attributes or predicates. Note that rule discovery in NEX incurs only a one-time offline cost.

Compared with the levelwise LMiner, which fails to finish within 24 hours, NEX mines rules efficiently through motif identification and composition. Although Trans is the largest dataset, it is not the slowest for NEX, as its limited vertex label types (Table III) yield simpler motif combinations.

Varying k. We tested the efficiency of NEX by varying k (Figure 10(g)). As k increases, more k-bounded motifs are generated, leading to longer runtime; *e.g.,* increasing k from 3 to 9 makes rule discovery 10.64× slower on average. On Trans, however, the growth is smoother, as fewer label types allow smaller k to already cover most frequent patterns.

Varying #motifs in $\mathcal{S}_{\mathcal{M}}$ gets similar trend (Figure 10(h)).

Varying σ *or* δ. We varied support σ from 1K to 10K on Loan in Figure 10(i). As expected, when σ increases, NEX runs faster by filtering more rules by anti-monotonicity. However, since confidence is not anti-monotonic, it gets slightly slower (not shown) and more rules are checked when δ increases.

Varying n. Varying the number n of processors, we tested the parallel scalability of RxGNNMiner in Figure 10(j). It is 3.06× faster when n varies from 4 to 16, *i.e.,* it is parallelly scalable.

Exp-5: Ablation study. We also tested variants of NEX.

Local explanations. We tested two variants of CFE: (a) noC, which omits clustering and computes cost-effectiveness for all vertices, and (b) BF, the brute-force method that enumerates all witnesses (Section V). As shown in Figure 9(a), CFE outperforms noC and BF by 1.24× and 3.55×, respectively.

Global explanations. For efficiency, we tested RxGNNMiner with two variants: (a) without DQN, which exhaustively enumerates the most confident σ-frequent pattern, and (b) without perplexity-based predicate ordering (PPO). As shown in Figure 9(b), adding DQN and PPO accelerates rule discovery by 1.93× and 1.21× on average, respectively. DQN efficiently guides motif composition, while PPO improves multi-thread parallelism by prioritizing more discriminative predicates.

For effectiveness, we tested RxGNNMiner with star-pattern rules (*i.e.,* an adaptation of [38], [65], [66]). This restriction reduces expressivity: on Amazon with GCN, NEX 's recog falls from 0.731 to 0.427 and relia from 0.604 to 0.224 (not shown), because star patterns cannot capture richer dependencies beyond hub-and-spoke structures, among others.

Summary. We find the following. (1) *Effective local counterfactuals*: Across four real-world graphs, NEX improves fidelity by 33.60% and sparsity by 3.25× over prior methods. (2) *Efficiency*: NEX generates a counterfactual in only 1.38s on average, even for graphs with 15.4M vertices and 12.7M edges. (3) *Accurate global explanations*: The discovered RxGNNs achieve 70.16% higher recognizability and 189.68% higher reliability on average, with gains up to 339.62% and 572.03% over baselines. (4) *Scalability*: Its discovery algorithm is parallelly scalable, enabling efficient rule mining on large graphs.

VII. Conclusion

NEX makes several contributions for explaining GNN-based classifiers: (a) a unified logical framework that provides both global and local explanations for negative predictions; (b) RxGNNs, a new class of graph rules that incorporate negated predicates and the 1-WL test to capture diverse GNN behaviors with higher expressiveness; (c) a model-guided, parallelly scalable algorithm for efficient discovery of RxGNNs as global explanations; and (d) a cost-effective vertex-filtering algorithm for generating local counterfactual explanations. Empirical results confirm that NEX is promising in practice.

Future work includes extending NEX beyond GNNs to more general ML models, supporting both factual and counterfactual explanations across diverse data modalities.

VIII. AI-Generated Content Acknowledgement

The content (including but not limited to text, figures, images, and code) was not generated by artificial intelligence.

References

[1] C. Gao, Y. Zheng, N. Li, Y. Li, Y. Qin, J. Piao, Y. Quan, J. Chang, D. Jin, X. He, and Y. Li, "A survey of graph neural networks for recommender systems: Challenges, methods, and directions," *ACM Transactions on Recommender Systems*, vol. 1, no. 1, pp. 1–51, 2023.

[2] S. Wu, W. Zhang, F. Sun, and B. Cui, "Graph neural networks in recommender systems: A survey," *CoRR*, vol. abs/2011.02260, 2020.

[3] S. X. Rao, S. Zhang, Z. Han, Z. Zhang, W. Min, Z. Chen, Y. Shan, Y. Zhao, and C. Zhang, "xFraud: Explainable fraud transaction detection," *PVLDB*, vol. 15, no. 3, pp. 427–436, 2021.

[4] Y. Luo, G. Wang, Y. Liu, J. Yue, W. Cheng, and B. Fei, "FAF: A risk detection framework on industry-scale graphs," in *CIKM*, 2023, pp. 4717–4723.

[5] M. Sun, S. Zhao, C. Gilvary, O. Elemento, J. Zhou, and F. Wang, "Graph convolutional networks for computational drug development and discovery," *Briefings in bioinformatics*, vol. 21, no. 3, pp. 919–935, 2020.

[6] J. Xiong, Z. Xiong, K. Chen, H. Jiang, and M. Zheng, "Graph neural networks for automated de novo drug design," *Drug discovery today*, vol. 26, no. 6, pp. 1382–1393, 2021.

[7] Y. L. Liu, Y. Wang, O. Vu, R. Moretti, B. Bodenheimer, J. Meiler, and T. Derr, "Interpretable chirality-aware graph neural network for quantitative structure activity relationship modeling in drug discovery," in *AAAI*, vol. 37, no. 12, 2023, pp. 14 356–14 364.

[8] D. Qiu, M. Wang, A. Khan, and Y. Wu, "Generating robust counterfactual witnesses for graph neural networks," *ICDE*, 2024.

[9] J. Cui, M. Yu, B. Jiang, A. Zhou, J. Wang, and W. Zhang, "Interpretable knowledge tracing via response influence-based counterfactual reasoning," in *ICDE*, 2024.

[10] S. An and Y. Cao, "Counterfactual explanation at will, with zero privacy leakage," *PACMMOD*, 2024.

[11] A. Lucic, M. A. Ter Hoeve, G. Tolomei, M. De Rijke, and F. Silvestri, "CF-GNNExplainer: Counterfactual explanations for graph neural networks," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2022, pp. 4499–4511.

[12] M. Bajaj, L. Chu, Z. Y. Xue, J. Pei, L. Wang, P. C.-H. Lam, and Y. Zhang, "Robust counterfactual explanations on graph neural networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 5644–5655, 2021.

[13] S. Verma, B. Armgaan, S. Medya, and S. Ranu, "InduCE: Inductive counterfactual explanations for graph neural networks," *Transactions on Machine Learning Research*, 2024.

[14] J. Ma, R. Guo, S. Mishra, A. Zhang, and J. Li, "CLEAR: Generative counterfactual explanations on graphs," *Advances in neural information processing systems*, vol. 35, pp. 25 895–25 907, 2022.

[15] H.-M. Attolou, K. Tzompanaki, K. Stefanidis, and D. Kotzinos, "Why-not explainable graph recommender," in *ICDE*, 2024.

[16] D. Qiu, J. Chen, A. Khan, Y. Zhao, and F. Bonchi, "Finding counterfactual evidences for node classification," *arXiv preprint arXiv:2505.11396*, 2025.

[17] J. Tan, S. Geng, Z. Fu, Y. Ge, S. Xu, Y. Li, and Y. Zhang, "Learning and evaluating graph neural network explanations based on counterfactual and factual reasoning," in *The Web Conference*, 2022, pp. 1018–1027.

[18] Z. Chen, F. Silvestri, J. Wang, Y. Zhang, Z. Huang, H. Ahn, and G. Tolomei, "GREASE: generate factual and counterfactual explanations for GNN-based recommendations," *CoRR*, vol. abs/2208.04222, 2022.

[19] D. Qiu, M. Wang, A. Khan, and Y. Wu, "Generating robust counterfactual witnesses for graph neural networks," in *ICDE*. IEEE, 2024, pp. 3351–3363.

[20] M. A. Prado-Romero, B. Prenkaj, G. Stilo, and F. Giannotti, "A survey on graph counterfactual explanations: Definitions, methods, evaluation, and research challenges," *ACM Comput. Surv.*, vol. 56, no. 7, pp. 171:1–171:37, 2024.

[21] H. Yuan, J. Tang, X. Hu, and S. Ji, "XGNN: Towards model-level explanations of graph neural networks," in *SIGKDD*, 2020.

[22] X. Wang and H. W. Shen, "Gnninterpreter: A probabilistic generative model-level explanation for graph neural networks," in *International Conference on Learning Representations*, 2022.

[23] G. Lv and L. Chen, "On data-aware global explainability of graph neural networks," *PVLDB*, vol. 16, no. 11, pp. 3447–3460, 2023.

[24] Z. Huang, M. Kosan, S. Medya, S. Ranu, and A. Singh, "Global counterfactual explainer for graph neural networks," in *ACM International Conference on Web Search and Data Mining*, 2023, pp. 141–149.

[25] G. Fournier and S. Medya, "COMRECGC: Global graph counterfactual explainer through common recourse," *arXiv preprint arXiv:2505.07081*, 2025.

[26] T. Chen, D. Qiu, Y. Wu, A. Khan, X. Ke, and Y. Gao, "View-based explanations for graph neural networks," *Proc. ACM Manag. Data*, vol. 2, no. 1, pp. 1–27, 2024.

[27] C. Abrate and F. Bonchi, "Counterfactual graphs for explainable classification of brain networks," in *SIGKDD*, 2021, pp. 2495–2504.

[28] "The dataset for job application prediction," 2024, *https://www.kaggle.com/datasets/sumittr26/employee-upskilling-and-hiring-success-dataset.*

[29] B. Y. Weisfieler and A. A. Leman, "The reduction of a graph to canonical form and the algebra which appears therein," *NIT*, vol. 2, 1968.

[30] H. L. Morgan, "The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service," *Journal of Chemical Documentation*, vol. 5, no. 2, pp. 107–113, 2065.

[31] J. Cai, M. Fürer, and N. Immerman, "An optimal lower bound on the number of variables for graph identification," *Comb.*, vol. 12, no. 4, pp. 389–410, 1992.

[32] M. Grohe, "word2vec, node2vec, graph2vec, x2vec: Towards a theory of vector embeddings of structured data," in *PODS*, 2020, pp. 1–16.

[33] C. P. Kruskal, L. Rudolph, and M. Snir, "A complexity theory of efficient parallel algorithms," *Theor. Comput. Sci.*, vol. 71, no. 1, pp. 95–132, 1990.

[34] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nat.*, 2015.

[35] "Code and dataset," 2025, https://github.com/hdqlkncso24jvf/NEX.

[36] "Full version," 2025, https://github.com/hdqlkncso24jvf/NEX/blob/main/paper_full_version.pdf.

[37] S. Azzolin, A. Longa, P. Barbiero, P. Liò, and A. Passerini, "Global explainability of GNNs via logic combination of learned concepts," in *ICLR 2023*, 2023, pp. 1–19.

[38] W. Fan, L. Fan, D. Lin, and M. Xie, "Explaining GNN-based recommendations in logic," *PVLDB*, 2025.

[39] C. Rudin and Y. Shaposhnik, "Globally-consistent rule-based summary-explanations for machine learning models: Application to credit-risk evaluation," *J. Mach. Learn. Res.*, vol. 24, pp. 16:1–16:44, 2023.

[40] C. Chen, K. Lin, C. Rudin, Y. Shaposhnik, S. Wang, and T. Wang, "An interpretable model with globally consistent explanations for credit risk," *CoRR*, vol. abs/1811.12615, 2018.

[41] Z. Geng, M. Schleich, and D. Suciu, "Computing rule-based explanations by leveraging counterfactuals," *PVLDB*, vol. 16, no. 3, 2022.

[42] L. Bertossi, "Declarative approaches to counterfactual explanations for classification," *Theory and Practice of Logic Programming*, vol. 23, no. 3, pp. 559–593, 2023.

[43] G. Audemard, J.-M. Lagniez, and P. Marquis, "On the computation of contrastive explanations for boosted regression trees," in *The 27th European Conference on Artificial Intelligence*, 2024.

[44] L. Bertossi, J. Li, M. Schleich, D. Suciu, and Z. Vagena, "Causality-based explanation of classification outcomes," in *International Workshop on Data Management for End-to-End Machine Learning*, 2020, pp. 1–10.

[45] J. Marques-Silva, "Logic-based explainability: Past, present and future," in *International Symposium on Leveraging Applications of Formal Methods*. Springer, 2024, pp. 181–204.

[46] A. Darwiche, "Logic for explainable AI," in *Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 2023, pp. 1–11.

[47] M. Schleich, Z. Geng, Y. Zhang, and D. Suciu, "GeCo: Quality counterfactual explanations in real time," *PVLDB*, 2021.

[48] P. M. VanNostrand, H. Zhang, D. M. Hofmann, and E. A. Rundensteiner, "FACET: Robust counterfactual explanation analytics," *Proc. ACM Manag. Data*, vol. 1, no. 4, pp. 242:1–242:27, 2023.

[49] A. Karimi, G. Barthe, B. Balle, and I. Valera, "Model-agnostic counterfactual explanations for consequential decisions," *CoRR*, vol. abs/1905.11190, 2019.

[50] R. K. Mothilal, A. Sharma, and C. Tan, "Explaining machine learning classifiers through diverse counterfactual explanations," in *Conference on Fairness, Accountability, and Transparency (FAT)*. ACM, 2020, pp. 607–617.

[51] W. Fan, Y. Wu, and J. Xu, "Functional dependencies for graphs," in *SIGMOD*. ACM, 2016, pp. 1843–1857.

[52] W. Fan, R. Jin, M. Liu, P. Lu, C. Tian, and J. Zhou, "Capturing associations in graphs," *PVLDB*, vol. 13, no. 11, pp. 1863–1876, 2020.

[53] W. Fan and P. Lu, "Dependencies for graphs," *ACM Trans. Database Syst. (TODS)*, vol. 44, no. 2, pp. 5:1–5:40, 2019.

[54] W. Akhtar, A. Cortés-Calabuig, and J. Paredaens, "Constraints in RDF," in *SDKB*, 2010.

[55] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[56] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," in *The Web conference 2020*, 2020, pp. 2704–2710.

[57] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," *Advances in neural information processing systems*, vol. 32, 2019.

[58] H. Tang, C. Wang, J. Zheng, and C. Jiang, "Enabling graph neural networks for semi-supervised risk prediction in online credit loan services," *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 1, pp. 1–24, 2024.

[59] M. Zopf, "1-WL expressiveness is (almost) all you need," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 1–8.

[60] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *ICLR*, 2017.

[61] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *ICLR*, 2019.

[62] D. Qiu, H. Che, A. Khan, and Y. Wu, "Generating skyline explanations for graph neural networks," *arXiv preprint arXiv:2505.07635*, 2025.

[63] W. Fan, X. Wang, Y. Wu, and J. Xu, "Association rules with graph patterns," *PVLDB*, vol. 8, no. 12, pp. 1502–1513, 2015.

[64] W. Fan, R. Jin, P. Lu, C. Tian, and R. Xu, "Towards event prediction in temporal graphs," *PVLDB*, vol. 15, no. 9, pp. 1861–1874, 2022.

[65] W. Fan, W. Fu, R. Jin, M. Liu, P. Lu, and C. Tian, "Making it tractable to catch duplicates and conflicts in graphs," *Proc. ACM Manag. Data*, vol. 1, no. 1, pp. 86:1–86:28, 2023.

[66] L. Fan, W. Fan, P. Lu, C. Tian, and Q. Yin, "Enriching recommendation models with logic conditions," *Proc. ACM Manag. Data*, vol. 1, no. 3, pp. 210:1–210:28, 2024.

[67] M. Grohe, "The logic of graph neural networks," in *LICS*, 2021, pp. 1–17.

[68] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "GNNExplainer: Generating explanations for graph neural networks," in *NeurIPS*, 2019, pp. 9240–9251.

[69] S. Nijssen and J. N. Kok, "The Gaston tool for frequent subgraph mining," *Electronic Notes in Theoretical Computer Science*, vol. 127, no. 1, pp. 77–87, 2005.

[70] A. P. Iyer, Z. Liu, X. Jin, S. Venkataraman, V. Braverman, and I. Stoica, "ASAP: Fast, approximate graph pattern mining at scale," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018, pp. 745–761.

[71] S. Ranu and A. K. Singh, "GraphSig: A scalable approach to mining significant subgraphs in large graph databases," in *ICDE*. IEEE, 2009, pp. 844–855.

[72] M. Thoma, H. Cheng, A. Gretton, J. Han, H.-P. Kriegel, A. Smola, L. Song, P. S. Yu, X. Yan, and K. M. Borgwardt, "Discriminative frequent subgraph mining with optimality guarantees," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 3, no. 5, pp. 302–318, 2010.

[73] J. Huan, W. Wang, J. Prins, and J. Yang, "Spin: Mining maximal frequent subgraphs from graph databases," in *SIGKDD*, 2004.

[74] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis, "GRAMI: Frequent subgraph and pattern mining in a single large graph," *PVLDB*, vol. 7, no. 7, pp. 517–528, 2014.

[75] W. Lin, X. Xiao, and G. Ghinita, "Large-scale frequent subgraph mining in MapReduce," in *ICDE*, 2014.

[76] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," in *International Conference on Data Mining*. IEEE, 2002.

[77] X. Yan, H. Cheng, J. Han, and P. S. Yu, "Mining significant graph patterns by leap search," in *SIGMOD*, 2008, pp. 433–444.

[78] C. Jiang, F. Coenen, and M. Zito, "A survey of frequent subgraph mining algorithms," *Knowledge Eng. Review*, vol. 28, no. 01, pp. 75–105, 2013.

[79] W. Fan, C. Hu, X. Liu, and P. Lu, "Discovering graph functional dependencies," *TODS*, vol. 45, no. 3, pp. 1–42, 2020.

[80] W. Fan, W. Fu, R. Jin, P. Lu, and C. Tian, "Discovering association rules from big graphs," *PVLDB*, vol. 15, no. 7, pp. 1479–1492, 2022.

[81] H. Chen, W. Fan, and J. Zheng, "Incremental rule discovery in response to parameter updates," *Proc. ACM Manag. Data*, vol. 3, no. 3, pp. 175:1–175:28, 2025.

[82] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen, "TANE: An efficient algorithm for discovering functional and approximate dependencies," *The computer journal*, vol. 42, no. 2, 1999.

[83] X. Chu, I. F. Ilyas, and P. Papotti, "Discovering denial constraints," *PVLDB*, vol. 6, no. 13, pp. 1498–1509, 2013.

[84] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[85] N. Alon, R. Yuster, and U. Zwick, "Color-coding," *Journal of the ACM (JACM)*, vol. 42, no. 4, pp. 844–856, 1995.

[86] M. Han, H. Kim, G. Gu, K. Park, and W.-S. Han, "Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together," in *SIGMOD*, 2019, pp. 1429–1446.

[87] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Wadsworth, 1984.

[88] M. V. Gwetu, J. Tapamo, and S. Viriri, "Random forests with a steepend gini-index split function and feature coherence injection," in *MLN*, 2019, pp. 255–272.

[89] "The dataset for loan approval prediction," 2024, *https://www.kaggle.com/datasets/hydracsnova/loan-approval-dataset*.

[90] R. Zhang, D. Cheng, J. Yang, Y. Ouyang, X. Wu, Y. Zheng, and C. Jiang, "Pre-trained online contrastive learning for insurance fraud detection," in *The AAAI Conference on Artificial Intelligence*, vol. 38, no. 20, 2024, pp. 22 511–22 519.

[91] "The dataset for fraudulent activities detection," 2024, *https://www.kaggle.com/datasets/rohit265/fraud-detection-dynamics-financial-transaction*.

[92] "Amazon product graph," 2025, *https://cseweb.ucsd.edu/∼jmcauley/datasets/*.

[93] H. Yuan, H. Yu, J. Wang, K. Li, and S. Ji, "On explainability of graph neural networks via subgraph explorations," in *ICML*. PMLR, 2021, pp. 12 241–12 252.

[94] H. Yuan, H. Yu, S. Gui, and S. Ji, "Explainability in graph neural networks: A taxonomic survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 45, no. 5, pp. 5782–5799, 2022.

[95] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *International conference on machine learning*. PMLR, 2018, pp. 5453–5462.

[96] S. Zhang, H. Li, M. Wang, M. Liu, P.-Y. Chen, S. Lu, S. Liu, K. Murugesan, and S. Chaudhury, "On the convergence and sample complexity analysis of deep q-networks with ϵ-greedy exploration," *Advances in Neural Information Processing Systems*, vol. 36, pp. 13 064–13 102, 2023.

[97] W. Fan, P. Lu, and C. Tian, "Unifying logic rules and machine learning for entity enhancing," *Sci. China Inf. Sci.*, vol. 63, no. 7, 2020.

[98] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for capturing data inconsistencies," *TODS*, vol. 33, no. 2, p. 6, 2008.

[99] W. Fan, H. Gao, X. Jia, J. Li, and S. Ma, "Dynamic constraints for record matching," *VLDB J.*, vol. 20, no. 4, pp. 495–520, 2011.

[100] C. Wyss, C. Giannella, and E. Robertson, "FastFD: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances extended abstract," in *DaWaK*, 2001.

[101] A. Cortés-Calabuig and J. Paredaens, "Semantics of constraints in RDFS," in *AMW*, 2012.

[102] M. S. Schlichtkrull, N. De Cao, and I. Titov, "Interpreting graph neural networks for nlp with differentiable edge masking," in *ICLR*, 2020.

[103] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang, "Parameterized explainer for graph neural network," *NeurIPS*, vol. 33, pp. 19 620–19 631, 2020.

[104] J. Ma, I. Takigawa, and A. Yamamoto, "C2Explainer: Customizable mask-based counterfactual explanation for graph neural networks," in *Conference on Fairness, Accountability, and Transparency (FAT)*, 2025, pp. 137–149.

[105] C. Borgelt and M. R. Berthold, "Mining molecular fragments: Finding relevant substructures of molecules," in *2002 IEEE International Conference on Data Mining*. IEEE, 2002, pp. 51–58.

[106] M. Kuramochi and G. Karypis, "Frequent subgraph discovery," in *IEEE international conference on data mining*. IEEE, 2001, pp. 313–320.

APPENDIX

This appendix is organized as follows.

- Appendix A: Details about Rule Discovery
- Appendix B: Details of Local Counterfactual Explanation
- Appendix C: Other Related Work
- Appendix D: Details about Experiments

Appendix A: Details about Rule Discovery

This section provides additional details of our rule discovery approach, RxGNNMiner. We first describe the motif identification strategy, which mines σ-frequent and k-bounded motifs (Appendix A-1). Next, we elaborate on motif composition using DQN (Appendix A-2). We then discuss the parallel scalability of RxGNNMiner (Appendix A-3). Finally, we explain how to compute a cover of RxGNNs (Appendix A-4).

Appendix A-1: Discussion about Motif Identification

By our motif identification strategy, we mine only σ-frequent and k-bounded motifs, as follows.

(1) The use of σ is to prune useless motifs without missing any rule above the support threshold. Specifically, given a motif π and a pattern Q composed from π (possibly with other motifs), π subsumes Q and thus, $\mathsf{supp}(\pi, G) \geq \mathsf{supp}(Q, G)$ by anti-monotonicity. Therefore, if a motif is not σ-frequent, all patterns composed from it will not be σ-frequent, which can be safely excluded from consideration.

(2) We use k to balance cost and discriminability: if k is too small, the motifs may not be discriminative enough; and if k is too large, it becomes costly to obtain all k-bounded motifs. In practice, it often suffices to set k small (*e.g.,* $k \leq 5$), *e.g.,* the best performance of a GNN-based model is observed when it aggregates information from its 2-hop neighbors [95], and "deeper versions of the model that, in principle, have access to more information, perform worse [60]".

Appendix A-2: Details of Motif Composition

Motif composition leads to a large state space. To address this, we adopt deep Q-learning (DQN) [34], which handles large state spaces efficiently and models complex structures with neural networks for better generalization. In contrast, traditional Q-learning, while simpler, struggles with scalability due to large Q-tables and inefficiency. Below we first formulate motif composition as a Markov Decision Process (MDP), and then describe the training and inference of DQN.

MDP. Given a set S of motifs, we create an initial pattern $Q_0[\bar{x}, x_0]$ by merging vertices x_0 from motifs in S into one x_0 in Q_0. We model the composition as a sequential decision process: given the current $Q[\bar{x}, x_0]$, we make a decision by selecting a pair of vertices in Q with the same label to merge. This can be modeled as an MDP in the following.

- *State*: The current pattern Q represents a state s.
- *Action*: Each mergable pair (v, v') of vertices in Q (*i.e.,* $L_Q(v) = L_Q(v')$) is an action a. We also use a special action [end] to terminate the composition, and stop if the $\mathsf{supp}(Q, G) < \sigma$ (by anti-monotonicity). Denote the action space of a state s by $\mathcal{A}(s)$.
- *Transition*: We transit from a state to another by taking an action, *e.g.,* given the current state $s_i = Q_i$ and an action $a = (v, v')$, we transit to a new state $s_{i+1} = Q_{i+1}$, denoted as $s_i \rightarrow s_{i+1}$, where v and v' are represented as a single merged vertex in Q_{i+1}.
- *Reward*: A reward r is associated with the transition $s_i \rightarrow s_{i+1}$, indicating the effect of taking action (v, v'):

$$r = \begin{cases} -1 & \text{if } \mathsf{supp}(Q_{i+1}, G) < \sigma, \\ \mathsf{conf}(Q_{i+1}, G) - \mathsf{conf}(Q_i, G) & \text{if } \mathsf{supp}(Q_{i+1}, G) \geq \sigma. \end{cases}$$

Training of DQN. To utilize DQN for pattern composition, we adopt several component networks: (a) a *state encoder*, $\mathsf{encoder}_\phi(Q)$ (resp. *action encoder*, $\mathsf{encoder}_\phi(v, v')$), with parameter ϕ that maps a state $s = Q$ (resp. an action $a = (v, v')$)) to an embedding $\mathbf{e}_s$ (resp. $\mathbf{e}_a$), and (b) a *Q-value network* $\mathbb{Q}_\theta(s, a)$ with parameter θ that predicts the expected accumulated reward (*i.e.,* Q-value) by taking action a at s.

Specifically, we leverage the vertex embedding $\mathbf{e}_v^L$ outputted by the L-layer GNN model $\mathcal{M}$ to encode a pattern Q, *i.e.,*

$$\mathbf{e}_s = \mathsf{encoder}_\phi(s) = \mathsf{readout}(\{\mathbf{e}_v^L \mid v \in Q \text{ where } s = Q\}),$$

where the readout function aggregates vertex embeddings into a pattern-level representation; likewise, action embeddings are obtained by concatenating vertex embeddings, *i.e.,*

$$\mathbf{e}_a = \mathsf{encoder}_\phi(a) = \mathsf{concat}(\mathbf{e}_v^L, \mathbf{e}_{v'}^L), \text{ where } a = (v, v').$$

For the Q-value network, we adopt

$$\mathbb{Q}_\theta(s, a) = \mathsf{FFN}_\theta(\mathbf{e}_s, \mathbf{e}_a),$$

where $\mathsf{FFN}()$ is a feed-forward neural network. We train it with the ϵ-greedy strategy, minimizing the loss $\mathcal{L}(\theta, \phi)$:

$$\mathbb{E}_{(s_i, a, r, s_{i+1}) \sim \mathcal{T}} \left[\left(r + \gamma \max_{a' \in \mathcal{A}(s_{i+1})} \mathbb{Q}_{\theta'}(s_{i+1}, a') - \mathbb{Q}_\theta(s_i, a) \right)^2 \right],$$

where $\mathcal{T}$ is a replay buffer containing transitions (s_i, a, r, s_{i+1}) (*i.e.,* we transit from s_i to s_{i+1} by taking action a with reward r), γ is a discount factor, and θ' is the parameter of a target network $\mathbb{Q}_{\theta'}$, periodically copied from $\mathbb{Q}_\theta$ for providing a stable target during training.

DQN *Inference*. Given the trained $\mathbb{Q}_\theta(s, a)$, we initialize with pattern Q_0 and iteratively merge the vertex pair with the highest Q-value until the termination condition is met.

Sample complexity. The sample complexity of DQN with ϵ-greedy is polynomially with the network size (*e.g.,* linear to the dimension of the feature space for state-action pairs and the number of layers in the neural network), and inversely with the square of the desired accuracy (see [96]).

Appendix A-3: Extended Discussion of Discovery

This section extends discussion on RxGNNMiner.

Parallel scalability. We adopt *parallel scalability* [33] to measure the effectiveness of RxGNNMiner. Consider a problem $\mathbb{P}$ on graph G. We denote by $T_s(|I_\mathbb{P}|, |G|)$ the worst-case complexity of a sequential algorithm $\mathcal{A}$ for handling an instance $I_\mathbb{P}$ of $\mathbb{P}$ over G. For a parallel algorithm $\mathcal{A}_p$ for $\mathbb{P}$, we denote by $T_p(|I_\mathbb{P}|, |G|, n)$ its time taken for processing

problem instance $I_{\mathbb{P}}$ on G using n processors. We say that algorithm $\mathcal{A}_p$ is *parallelly scalable relative to* $\mathcal{A}$ if

$$T_p(|I_{\mathbb{P}}|, |G|, n) = O\big(T_s(|I_{\mathbb{P}}|, |G|)/n\big)$$

for any instance $I_{\mathbb{P}}$. That is, the parallel algorithm $\mathcal{A}_p$ is able to "linearly" reduce the sequential cost of a yardstick algorithm $\mathcal{A}$. Denote by SeqMiner the sequential version of RxGNNMiner that validates each RxGNN one by one, without multi-thread parallelism. We have the following.

Theorem 3: RxGNNMiner *is parallelly scalable relative to sequential rule mining algorithm* SeqMiner. □

Proof. We show that the complexity of RxGNNMiner is $O(\frac{1}{n} \times |\mathcal{S}_{\mathcal{M}}|^K \sum_{X \in \mathsf{pow}(\mathcal{P})} |G|^{|X|})$ with n processors, where K is the maximum number of motifs in $\mathcal{S}_{\mathcal{M}}$ that can compose a σ-frequent pattern Q (typically, $K \ll |\mathcal{S}_{\mathcal{M}}|$), $\mathcal{P}$ is the set of all potential predicates defined on a pattern Q and $\mathsf{pow}(\mathcal{P})$ is the power set of $\mathcal{P}$. This is because (a) the computation is dominated by the validation of support and confidence, where the computational tasks are evenly distributed across n processors, and (b) it examines $\mathsf{pow}(\mathcal{P})$ to generate preconditions X for each pattern Q at worst. □

Space cost. The space complexity is dominated by the storage of minimal RxGNNs, which is $O(|\mathcal{S}_{\mathcal{M}}|^K \times (N + \#\mathsf{max}|\mathcal{P}|) + |G|)$, where N is the maximum number of vertices for each pattern Q and $\#\mathsf{max} = \binom{|\mathcal{P}|}{\lfloor \frac{|\mathcal{P}|}{2} \rfloor}$ is the maximum number of minimal RxGNNs with the same Q.

Remark on k-bounded motifs. We focus on k-bounded motifs, where the value of k balances cost and discriminability. In practice, users can often (a) start with a conservative k (*e.g.,* $k = 3$), incrementally increase it, and stop when marginal gains diminish; or (b) set time or memory budget, and dynamically adjust k to stay within resource limits; or (c) leverage data statistics to select a suitable k; or (d) train ML models to guide the choice of k. In addition, an incremental tuning strategy [81] can also be employed for k.

Appendix A-4: Cover Computation

The problem of cover computation is as follows.

- *Input*: A set Σ of RxGNNs discovered.
- *Output*: A cover Σ^c of Σ such that (1) $\Sigma^c \models \Sigma$, *i.e.,* $\Sigma^c \models \varphi$ for all $\varphi \in \Sigma$, and (2) $\Sigma^c \setminus \{\varphi\} \not\models \Sigma^c$ for any $\varphi \in \Sigma^c$.

Implication analysis (*i.e.,* decide whether a rule φ is implied by a set Σ of rules, denoted as $\Sigma \models \varphi$) is embedded in cover computation. Below we first review the characterization of implication in [79] and adapt it to RxGNNs, which take the same classifier $\mathcal{M}$ in the consequences, followed by a parallel algorithm for cover computation along the same line as in [79].

Characterization. For $\varphi = Q[\bar{x}, x_0](X \to \neg\mathcal{M}(x_0))$, the set of RxGNNs embedded in Q, denoted by Σ_Q, consists of RxGNNs of the form $\varphi' = Q'[\bar{x}', x_0](f(X') \to \neg\mathcal{M}(x_0))$ such that (a) there exists an RxGNN $\varphi_m = Q'[\bar{x}', x_0](X' \to \neg\mathcal{M}(x_0))$ in Σ and (b) there exists an homomorphic mapping f from Q' to Q, where $f(X')$ is obtained by replacing x by $f(x)$ for every vertex x appearing in X'. Intuitively, Σ_Q consists of φ' in Σ s.t. Q' can be mapped to Q and hence, φ' is enforced on every match of Q in a graph satisfying Σ.

For each Σ_Q and a given RxGNN $\varphi = Q[\bar{x}, x_0](X \to \neg\mathcal{M}(x_0))$, we define a set closure (Σ_Q, X) as follow:

(a) all predicates in X are in closure (Σ_Q, X); and
(b) for every $\varphi' = Q'[\bar{x}', x_0](X' \to \neg\mathcal{M}(x_0))$ in Σ_Q, $\neg\mathcal{M}(x_0)$ is in closure (Σ_Q, X) if either $X' = \emptyset$ or $X' \neq \emptyset$ and all predicates in X' can be deduced from closure (Σ_Q, X) via equality transitivity.

One can verify that, given Σ_Q, closure (Σ_Q, X) can be computed in PTIME, though determining Σ_Q itself is intractable. We say that closure (Σ_Q, X) is *conflicting* if there exists an attribute $x.A$ and distinct constants c and d such that both $x.A = c$ and $x.A = d$ can be derived.

The implication of RxGNNs is characterized as follows. For any set Σ of RxGNNs and $\varphi = Q[\bar{x}, x_0](X \to \neg\mathcal{M}(x_0))$, $\Sigma \models \varphi$ if and only if closure(Σ_Q, X) is conflicting or $\neg\mathcal{M}(x_0)$ is deducible from closure(Σ_Q, X). We say that $\neg\mathcal{M}(x_0)$ is *deducible* from Σ and X if there exists a subset Σ_Q of RxGNNs of Σ that are embedded in Q and derived from Σ, such that $\neg\mathcal{M}(x_0)$ can be derived from closure(Σ_Q, X) via the transitivity of equality atoms. Intuitively, it says that either (a) Σ and (Q, X) are inconsistent and hence, $\neg\mathcal{M}(x_0)$ is their "logical consequence"; or (b) closure(Σ_Q, X) is consistent and $\neg\mathcal{M}(x_0)$ is enforced by Σ and (Q, X).

Sequential cover computation. By utilizing the above characterization, a sequential cover computation algorithm works by inspecting RxGNNs in Σ one by one. For each $\varphi \in \Sigma$, it checks whether $\Sigma \setminus \{\varphi\} \models \varphi$ based on the characterization; if so, it removes φ from Σ, until no more φ can be removed and the resulting Σ is returned as Σ^c.

Following [79], one can verify that this algorithm correctly computes the cover Σ^c of Σ, since (a) each $\varphi \in \Sigma$ is checked based on the characterization of implication, and (b) upon termination, no φ can be further removed from Σ^c.

Parallelization. We parallelize the sequential algorithm. We partition the set Σ of RxGNNs into m groups $\Sigma_{Q_1}, \Sigma_{Q_2}, \ldots$ Σ_{Q_m}, where each Σ_{Q_i} $(i \in [1, m])$ is the set Σ of RxGNNs that pertain to the same pattern Q_i. It suffices to check the implication of RxGNNs within each group, since the implication checking is *pairwise independent* among the groups [79].

For each Σ_{Q_i}, we create a work unit and collect it in a set W. The workload in W are evenly distributed to all workers. Upon receiving the assigned work unit W_j, each worker P_j invokes the sequential algorithm above to compute local cover Σ_j^c in parallel. When all work units are processed, the union of all Σ_j^c's are returned.

Along the same lines as [79], one can readily verify that the parallel cover computation is also parallelly scalable.

Appendix B: Details about Local Counterfactual Explanation

This section provides additional details about our local counterfactual method CFE. We first present the details of DAG building and refinement in candidate space construction (Appendix B-1). We then show how to ensure the plausibility

and feasibility of feature modifications (Appendix B-2).

Appendix B-1: DAG Building and Refinement

We present the details of DAG building and refinement.

DAG building. To build a DAG Q_D from Q, we treat the designated vertex x_0 as the root. We traverse Q in a BFS order from x_0 and direct all pattern edges from upper levels to lower levels. To process more selective vertices earlier, the directions between vertices at the same level are decided by an *infrequent-first* order: we group vertices by their labels (one group per label) and then sort the groups in the ascending order of their numbers of occurrence in G; within each group, we sort vertices in the descending order of degrees.

Refinement. This step refines candidate space C_φ by propagating topological and logical constraints in both forward and backward directions. It implements dynamic programming between DAG Q_D and graph G [86].

More specifically, to refine $C_\varphi(x)$, it computes an auxiliary set $C'_\varphi(x)$ by dynamic programming such that vertex $v \in C'_\varphi(x)$ iff (a) $v \in C_\varphi(x)$, (b) there exists v_c adjacent to v such that $v_c \in C'_\varphi(x_c)$ for every child x_c of x in Q_D, and (c) there exists v_p such that $v.A \oplus v_p.B$ and $v_p \in C'_\varphi(x_p)$ for every variable predicate $x.A \oplus x_p.B$ in X.

Based on this, we compute $C_\varphi(x)$ by alternating forward refinement (propagating along Q_D) and backward refinement (along the reverse Q_D^{-1} of Q_D). This bidirectional refinement continues until no changes can be made to the candidate sets.

In practice, we can empirically set a maximum number of refinements, *e.g.,* when the filtering rate after the first three refinements is below 1% [86].

Appendix B-2: Plausibility and Feasibility

To ensure the plausibility and feasibility of feature modifications [47], we reuse the perplexity-based computation in Section IV-C, to leverage the capability and rich prior knowledge of LLMs. More specifically, for each attribute A, we pre-compute a function ρ_A: $\mathsf{dom}(A) \times \mathsf{dom}(A) \to \mathbb{R}^+$ for measuring the plausibility and feasibility of a modification, where $\mathsf{dom}(A)$ is the domain of attribute A:

$$\rho_A(a, a') = \exp\left(-\log P_{\text{LLM}}(\mathsf{nlp}(a \to a'))\right).$$

Here a (resp. a') is the original (resp. new) A-value, and $\mathsf{nlp}(a \to a')$ is the natural language description of the value modification, denoted as $a \to a'$, *e.g.,* "the A-attribute value is modified from a to a'", and P_{LLM} is the probability estimated by LLMs. Intuitively, $\rho_A(a, a')$ assigns a lower perplexity score for more plausible and feasible modification.

In this paper, we only consider modification $a \to a'$ whose $\rho_A(a, a')$ is low enough *w.r.t.* a threshold.

Appendix C: Other Related Work

This section provides additional literature review. We first present a detailed comparison of rule-based methods (Appendix C-1), and then review existing techniques that are incorporated as components of NEX (Appendix C-2).

Appendix C-1: Comparison of Rule-based Methods

Table V compares the expressivity of rule-based methods, *w.r.t.* their supported patterns, predicates, and consequences.

Rules are divided into two categories: rules for graph data and rules for tabular data. For each type of rules, we discuss (1) whether they support graph patterns (*e.g.,* general patterns or restricted patterns)? (2) What types of predicates the rules support, *e.g.,* constant predicates, variable predicates, inequality predicates ($\neq, \geq, \leq, >, <$), ML predicates, 1WL predicates and negated predicates? (3) What consequences of the rules and the primary purposes of the rules are.

Rules for graph data. As shown in Table V, among all the prior graph rules, RxGNNs are the most expressive ones, by supporting both general graph patterns and all types of predicates for counterfactual explanation. The most related rules are REPs proposed in Makex [38], but they differ from RxGNNs in the following aspects.

- *Patterns.* RxGNNs employ general graph patterns while REPs are defined on restricted star patterns. General patterns are more expressive than star patterns, albeit at the cost of higher computational complexity, *e.g.,* it only takes PTIME to compute the matches of a given star pattern, but this problem becomes NP-hard for a general pattern.
- *Variable predicates.* RxGNNs support all types of predicates. In contrast, to support efficient validation of REPs, they only partially support variable predicates on the centers and leaves (*i.e.,* vertices without any child vertices) of the star patterns; moreover, each center/leaf can carry at most one variable predicate.
- *Purposes.* Although both RxGNNs and REPs are designed for GNN explanations, REPs are used to generate *factual explanations*, while RxGNNs aim to provide *counterfactual explanations*. The different purposes of REPs and RxGNNs call for different designs and methodologies. The former identifies key substructures or features as sufficient conditions supporting the prediction, without modifying the graph; this does not necessarily require negated predicates. In contrast, the latter requires suggesting appropriate modifications to the input graphs as necessary conditions; we need rules that support negated predicates.

To give concrete illustration about the expressivity for RxGNNs, consider $\varphi_1 = Q_1[\bar{x}, x_0](X_1 \to \neg \mathcal{M}_1(x_0))$ in Figure 1(e), where (1) Q_1 is a general pattern and (2) X_1 involves (a) constant predicate such as $x_0.\#\mathsf{AI\_skill} = \mathsf{None}$, (b) variable predicate such as $x_2.\mathsf{name} \neq x_3.\mathsf{name}$, (c) comparison predicate $x_0.\#\mathsf{experience} > 5$, and (d) negated 1-WL predicate $\neg 1\mathsf{WL}(x_1)$. Note that φ_1 *cannot* be expressed by any existing rules, as elaborated below.

Prior rules REPs [38], TIEs [66] and GCRs [65] trade expressivity for efficiency by restricting graph patterns and variable predicates. REPs do not support general patterns and negated predicates as noted earlier. GARs [52] and TACOs [64] partially alleviate this by supporting general patterns and richer predicates, but still omit an essential class: 1-WL predicates. As argued in [38], GNNs are at most as powerful as the 1-WL test, and most GNNs are based on it; hence, 1-WL is necessary

Data	Rules	Patterns	Predicates						Consequences	Purpose of Rules
			Constant?	Variable?	beyond equality (*e.g.,* $\neq, \geq, \leq$)?	ML?	1WL?	ML/1WL Negation?		
Graph	RxGNNs	general	✓	✓	✓	✓	✓	✓	$\neg\mathcal{M}(x_0)$	counterfactual explanation
	REPs [38]	star only	✓	partially	✓	✓	✓	×	$\mathcal{M}(x_0)$	factual explanation
	TIEs [66]	star only	✓	partially	✓	✓	×	×	(x, likes, y) (*i.e.,* recommend item y to user x)	improve ML recommendation
	GARs [52]	general	✓	✓	✓	✓	×	×	all supported predicate types	association analyses
	GCRs [65]	star only	✓	partially	✓	✓	×	×	$x.A = y.B$ or $x.A = c$	graph data cleaning
	GFDs [51]	general	✓	✓	×	×	×	×	$x.A = y.B$ or $x.A = c$	graph data cleaning
	TACOs [64]	general	✓	✓	✓	✓	×	×	$((x, l, y), \tau)$ (*i.e.,* the event specified by (x, l, y) will take place within time window τ)	association analyses for temporal event prediction
	GPARs [63]	general	×	×	×	×	×	×	(x, l, y) (*i.e.,* the existence of edge (x, l, y))	association analyses
	GEDs [53]	general	✓	✓	×	×	×	×	$x.A = y.B$ or $x.A = c$	graph data cleaning
Tabular	[39], [40]	×	✓	×	✓	×	×	×	$\mathcal{M}(t) = i$	understand ML predictions
	RCs [41]	×	✓	×	✓	×	×	×	$\neg\mathcal{M}(t)$	counterfactual explanation
	REEs [97]	×	✓	✓	✓	✓	×	×	all supported predicate types	tabular data cleaning
	DCs [83]	×	✓	✓	✓	×	×	×	all supported predicate types	tabular data cleaning
	CFDs [98]	×	✓	✓	×	×	×	×	$t.A = s.B$ or $t.A = c$	conflict resolution
	MDs [99]	×	×	✓	×	✓	×	×	$t.\mathsf{id} = s.\mathsf{id}$	entity resolution
	FDs [100]	×	×	✓	×	×	×	×	$t.A = s.A$	database design

TABLE V: Expressivity comparison for different rules

to explain GNN behaviors. Without it, graph patterns with only simple arithmetic comparisons (*e.g.,* $x.A \oplus y.B$, $x.A \oplus c$) lack sufficient expressive power for complex GNNs. GFDs [101] and GEDs [53] suffer the same limitation with even fewer predicates, while GPARs [63] are the most restrictive, defined solely by graph patterns without predicates.

Rules for tabular data. Rules are common in tabular data, supporting tasks such as ML explanation [41], [39], [40], data quality [97], [83], [98], [99], and database design [100]. Yet the structural complexity of graphs makes these tabular rules ill-suited for explaining GNNs, which rely on message passing over vertices and edges to capture rich patterns. In particular, rule-based ML explanation methods for tabular data [41], [39], [40] support only constant predicates, thus inheriting the same expressivity limitations as GFDs [101] and GEDs [53].

Appendix C-2: Reviews of Existing Techniques

This section reviews existing techniques that we leverage as components in NEX for building the first effective, scalable and unified framework for local and global GNN explanations.

Note that these techniques are not claimed as our major contributions and thus they are just briefly mentioned in the main text, with key adaptation highlighted. Interested readers can find full details in the cited papers.

Incremental discovery. Rule discovery aims to mine all rules whose supports and confidences exceed given thresholds σ and δ, respectively. Building on this, incremental rule discovery seeks to compute the updated rule set when thresholds change, *i.e.,* $\sigma + \Delta\sigma$ or $\delta + \Delta\delta$. Algorithms are developed in [81] that minimize redundant recomputation, avoiding a full restart of the costly discovery process. For support updates, it incrementally reuses the logic and anti-monotonicity property of a batch miner, while for confidence updates, it employs a sampling strategy to accelerate traversal of the search lattice.

Based on incremental discovery, the users can tune parameters σ and δ iteratively (Section IV-C). They may start with quick initial settings to obtain an initial rule set, then refine σ and δ using insights from the results. This allows mining high-quality rules first and then adjust σ and δ to find more [81]. Along the same lines, we tune motif size k as well.

Graph mask. To identify important substructures, we adopt a graph masking strategy that removes non-critical edges, following prior work [68], [102], [93], [103], [104].

In our experiments, we use GNNExplainer [68] as the masker due to its simple yet effective design: it learns continuous edge-importance weights by maximizing the mutual information between the masked computation graph and the model's prediction. Compared with other masking techniques, it offers a good balance between effectiveness and efficiency, satisfying our requirements for motif identification.

Frequent pattern mining. Our motif identification summarizes key motifs from the set of subgraphs explaining GNN predictions. This step can be naturally extended using frequent pattern mining techniques [76], [69], [105], [106]. We employ GASTON [69] to mine σ-frequent and k-bounded patterns from masked subgraphs for its efficiency. GASTON discovers the frequent substructures by a layered growth scheme, *i.e.,* a number of phases of increasing complexity. More specifically, it searches first for frequent paths, then frequent free trees and finally cyclic graphs. Compared with other techniques, it achieves competitive efficiency (see more in Appendix D-3).

Appendix D: Details about Experiments

This section provides additional experimental details. We first describe the datasets (Appendix D-1), then explain how baseline methods are adapted to our setting (Appendix D-2), and finally present supplementary results evaluating different frequent pattern mining methods (Appendix D-3).

Appendix D-1: Additional Details about Datasets

In this section, we provide a detailed description of each dataset. We begin with a summary of four key statistical properties of four datasets in Table VI, followed by an elaboration on the vertices, edges, and attributes for each dataset.

Metrics. Four key statistical metrics were evaluated in Table VI, *i.e.,* the average degree, the degree distribution entropy,

TABLE VI: Basic statistics of the datasets.

Statistic	Loan	Claim	Trans	Amazon
Average degree	4.00	4.76	1.65	3.91
Degree distribution entropy	1.01	1.29	1.24	1.28
Assortativity coefficient	-0.20	-0.18	-0.08	-0.12
Spectral radius	174.26	720.13	7.42	232.89

the assortativity coefficient, and the spectral radius, all of which characterize the graph topology as elaborated below.

- *Average degree.* It measures how connected the vertices in a graph are on average. It is calculated by dividing the total number of edges by the total number of vertices.
- *Degree distribution entropy.* It measures the complexity of a graph's degree distribution. It is calculated as $-\sum_k P(k)\log_2 P(k)$, where $P(k)$ is the fraction of vertices with degree k. The higher the value is, the more diverse the degrees of the vertices are.
- *Assortativity coefficient.* It quantifies the tendency of vertices to connect with those of similar degrees, calculated by Pearson-correlation, in the range $[-1, 1]$, where a positive value indicates an *assortative* graph where vertices of similar degrees are connected, while a negative value represents a *disassortative* graph where the vertices of higher degree are connected with the ones of lower degree.
- *Spectral radius.* It measures the network connectivity. Specifically, it is defined as $\max_i |\lambda_i|$, where λ_i's are the eigenvalues of the adjacency matrix $\mathbf{A}$ of a graph. The larger the value, the denser the graph.

Loan. The Loan dataset contains comprehensive information about loan applications and their associated risk factors, designed for loan default prediction and risk assessment tasks. The dataset captures multi-dimensional relationships between borrowers, their geographical locations, professional backgrounds, and loan characteristics through a heterogeneous graph structure. It enables advanced analysis of loan approval patterns, risk factors, and predictive modeling for financial institutions to make informed lending decisions.

Vertices. Loan has the following types of vertices.

- Loan vertices, which represent individual loan applications or loan products within the system.
- Applicant vertices, which represent loan applicants or borrowers seeking financial assistance.
- City vertices, which represent geographical city locations where applicants reside or loans are processed.
- State vertices, which represent state-level geographical divisions for broader regional analysis.
- Profession vertices, which represent occupational categories or job types of loan applicants.

Edges. Loan has the following types of edges.

- Applicant-Loan. It indicates which applicant has applied for or is associated with a specific loan.
- Applicant-City, the residential location of the applicant.
- City-State. It describes which state each city belongs to.
- Applicant-Profession, the occupational categories.

Attributes. The vertices of Loan carry the following attributes.

- VertexId: A unique identifier assigned to each vertex for individual tracking and reference across vertex types.
- Label: A categorical indicator used to distinguish vertex types (*i.e.,* Loan, Applicant, City, State or Profession).
- Income: The financial income level of the loan applicant, representing their earning capacity and financial stability.
- Age: The age of the loan applicant.
- Experience: The total work experience of the applicant.
- Married_Single: The marital status of the applicant.
- House_Ownership: The housing ownership status of the applicant (owned, rented, etc.), indicating the asset ownership.
- Car_Ownership: The ownership status of the applicant, representing additional asset ownership and financial capacity.
- Profession: The occupational category of the applicant.
- Current_Job_Year: The number of years the applicant has been in their current job position, for employment stability.
- Current_House_Year: The duration the applicant has been residing at their current address.
- Risk_Flag: A binary ground truth label indicating whether the loan application poses a high risk (1) or low risk (0).
- GNNPrediction: The GNN prediction for a specific loan.

Claim. The Claim dataset contains real-world medical insurance information designed for fraud detection in healthcare claims. It encompasses approximately 200,000 beneficiaries, over 5,000 healthcare providers, and around 550,000 medical insurance claims, with 38.1% of claims identified as fraudulent. The dataset features meticulous fraud labels and timestamps annotated by experts, providing typicality and authority for medical fraud detection research.

Vertices. Claim has the following types of vertices.

- Claim vertices, which represent individual medical insurance claims submitted for reimbursement.
- Beneficiary vertices, which represent patients or individuals who receive medical services and file insurance claims.
- Provider vertices, which represent healthcare providers, hospitals, or medical facilities that deliver services.
- DiagnosisGroup vertices, which represent categories of medical diagnoses or disease classifications.

Edges. Claim has the following types of edges.

- Beneficiary-Claim. It indicates which beneficiary has filed a specific medical insurance claim.
- Provider-Claim. It shows which healthcare provider is associated with or has submitted a particular claim.
- Claim-DiagnosisGroup. It connects claims to their corresponding medical diagnosis categories or disease groups.

Attributes. The vertices of Claim carry the following attributes.

- VertexId: A unique identifier assigned to each vertex for individual tracking and reference across all vertex types.
- Label: A categorical indicator used to distinguish vertex

types (Claim, Provider, DiagnosisGroup or Beneficiary).
- InscClaimAmtReimbursed: The reimbursed value.
- ClmAdmitDiagnosisCode: The medical diagnosis code assigned upon hospital admission.
- DeductibleAmtPaid: The deductible amount paid by the beneficiary, for the out-of-pocket expense responsibility.
- Inpatient: A binary indicator for whether the medical service was provided as inpatient care or outpatient treatment.
- IsGroupFraud: A binary flag for whether the claim is part of a coordinated group fraud scheme with multiple parties.
- Gender: The gender of the beneficiary.
- Race: Racial demographic information of the beneficiary.
- RenalDiseaseIndicator: A binary indicator showing whether the beneficiary has renal (kidney) disease.
- State: The state information of the beneficiary/claim.
- County: The county information of the beneficiary/claim.
- numDiseases: The total number of diseases or medical conditions affecting the beneficiary.
- IsDead: A binary flag for whether the beneficiary is alive.
- IsFraud: A binary ground truth classification label indicating whether the claim is fraudulent (1) or legitimate (0).
- GNNPrediction: The GNN prediction for a specific claim.

Transaction. The Trans dataset contains comprehensive information about financial transactions for fraud detection. It comprises over 6.36 million transaction records, providing a rich and diverse collection of transactional data for analysis and modeling. Each transaction includes detailed information about the parties involved, monetary amounts, account balances, and fraud labels, making it particularly valuable for developing and evaluating fraud detection algorithms using graph-based machine learning approaches.

Vertices. Trans has the following types of vertices.
- Transaction vertices, which represent individual financial transactions within the system.
- Account vertices, which represent user accounts or entities participating in transactions.

Edges. Edges represent the relationships between Account vertices and Transaction vertices, indicating the flow of funds and transaction participation patterns within the financial network.

Attributes. The vertices of Trans carry the following attributes.
- VertexId: A unique identifier assigned to each vertex (transaction or account) for individual tracking and reference.
- Label: A categorical indicator used to distinguish vertex types (*i.e.,* Transaction or Account).
- Type: The category of financial transaction, including types such as PAYMENT, TRANSFER, CASH_OUT, DEBIT, etc., which help us classify different transaction behaviors.
- Amount: The monetary value involved in the transaction, for the financial magnitude and scale of each transaction.
- oldbalanceOrig: The account balance of the originating entity before the transaction occurs.
- newbalanceOrig: The updated account balance of the originating entity after the transaction is completed.
- oldbalanceDest: The account balance of the destination entity before receiving the transaction.
- newbalanceDest: The updated account balance of the destination entity after receiving the transaction.
- isFraud: A binary ground truth classification label for whether the transaction is fraudulent (1) or legitimate (0).
- GNNPrediction: The GNN prediction for a transaction.

Amazon. The dataset captures an Amazon clothing e-commerce network, modeling interactions among users, products and reviews. It covers commercial relationships, user preferences, and product information, supporting analyses of purchasing patterns, review authenticity, and recommendations.

Vertices. Amazon has the following types of vertices.
- Product vertices, which represent individual clothing products available for purchase on the Amazon platform.
- Review vertices, which represent individual customer reviews and ratings submitted for products.
- User vertices, which represent customers who purchase products or submit reviews on the platform.

Edges. Amazon has the following types of edges.
- User-Review, the ownership between users and reviews.
- User-(buy)-Product, indicating a user has bought a product.
- User-(comment)-Product. It indicates that the user has commented on a product but not purchased yet.
- Product-Review. It connects products to the reviews they have received from customers.

Attributes. The vertices carry the following attributes.
- VertexId: A unique identifier assigned to each vertex for individual tracking and reference across all vertex types.
- Label: A categorical indicator used to distinguish between the three different vertex types (Product, Review or User).
- reviewText: The detailed content of the customer review.
- reviewSummary: A concise summary or title of the review.
- reviewScore: A numerical rating score given by the reviewer, typically on a scale representing satisfaction level.
- productTitle: The title of the product.
- brand: The manufacturer or brand name of the product.
- price: The selling price of the product.
- reviewTime: The timestamp for review submission.
- reviewerName: The identifier of the customer who submitted the review.
- style: The specific attributes of the clothing item such as color, size, material, fit, or design options.
- vote: The number of votes/likes received by a review, indicating its perceived usefulness by other users.
- description: The product description provided by the seller.
- category: A multi-class ground truth classification label indicating product categories, *i.e.,* men's clothing (0), women's clothing (1), women's underwear (2), outdoor sports (3), jewelry (4).
- GNNPrediction: The GNN prediction for a specific product.

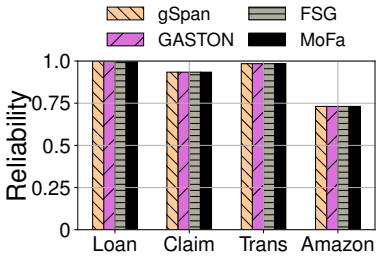


Fig. 11: Effectiveness of frequent pattern mining methods

Dataset	Loan	Claim	Trans	Amazon
gSpan	12520.4	36696.63	2284.83	8603.45
FSG	18023.66	53245.18	**1352.37**	12541.86
MoFa	15705.89	41197.48	1653.94	10959.61
GASTON	**9957.43**	**27482.12**	1408.67	**7038.08**

TABLE VII: Efficinecy of pattern mining methods (in seconds)

Appendix D-2: Details about Baselines

This section presents more details about the baselines. Note that not all baselines are designed for vertex classification on heterogeneous graphs where vertices can carry both labels and attributes. For example, in our problem, even two vertices are both labeled as user vertices, they can still carry different attributes, *e.g.,* ages, names and genders; this may not be the case in other settings, *e.g.,* graph classification for molecules, where each vertices are atoms. Thus we will adapt the baselines for (a) vertex classification and (b) attributed graphs. Note that to tackle labels and attributes on vertices, existing GNN techniques typically associate vertices with feature vectors (in the form of numerical encodings, *e.g.,* word embedding or one-hot encoding) [26]. We follow this in our adaptations.

Local explanation. These methods aim to generate graph perturbations that flip GNN predictions. CF-Exp restricts itself to edge removals, whereas InduCE and CFGraph allow both edge removals and insertions. CLEAR adopts a VAE framework, encoding an input graph into a latent space and decoding a counterfactual graph that preserves structural fidelity while flipping the prediction. GVEX, originally designed for graph classification, when adapted to vertex classification, it need to (a) construct a receptive field graph for each training vertex, and (b) then for each input graph, compute Jacobian matrices to identify vertices with maximal feature influence and performs graph bisection, gradually partitioning the graph into two subgraphs with opposing prediction outcomes.

Since most baselines ignore vertex feature modifications, we implemented a vertex-level masking scheme [68] under the same budget as edge modifications for fair comparison. In this scheme, baselines can apply feature masks to selected vertex feature vectors to perturb model predictions.

However, feature masking is often less intuitive than the explicit attribute dependencies in RxGNNs, where individual attributes are directly highlighted in the explanation. As noted in [38], applying the same feature mask to multiple vertices implicitly assumes that "features on different vertices are of equal importance", which is typically not the case.

To optimize counterfactual methods, we apply graph perturbations incrementally, one at a time when feasible. Once the prediction flips, the process stops; otherwise, it continues until no further perturbations are possible.

In particular, we would like to express our sincere gratitude to the authors of GVEX for their extensive discussions when we adapted their methods. Since GVEX is particularly designed for graph classification, it incurs performance degradation when adapting to vertex classification where the receptive field graph constructed for each training vertex (*i.e.,* the pivot vertex) is much larger than their original settings in [26]. This is mainly because that GVEX requires identifying influential subgraphs, which incurs a much higher computational complexity than identifying edges to insert/remove like most other methods. Moreover, since GVEX requires computing the Jacobian matrix to measure feature influence, it has to compute the pairwise "influence" of all vertices in each receptive field graph (*i.e.,* $O(n^2)$ where n is the number of vertices in the graph constructed for a training vertex). Although we can approximate this by only considering the "influence" of all vertices to the pivot (*i.e.,* reduce from $O(n^2)$ to $O(n)$), GVEX is still slower than NEX by around two orders of magnitudes. In our experiments, we mainly adopted the original setting in [26] and followed the suggestions of the authors.

Global explanation. Both XGNN and GNNInt generate discriminative graph patterns to characterize model behavior: XGNN employs reinforcement learning to maximize specific predictions, while GNNInt uses a probabilistic generative approach with numerical optimization to learn explanation graph distributions. Both methods were originally developed for molecular graph analysis and focus exclusively on topological pattern generation without incorporating vertex features. GVEX summarizes global graph patterns from local explanation subgraphs. CFGraph generates global explanations by aggregation of multiple counterfactual instances, by analyzing the structural differences and edge modification patterns across instances to identify discriminative structures.

Since XGNN and GNNInt generate only topological patterns without vertex features, we adapt them by learning embeddings for each vertex type in these two methods. For GVEX and CFGraph, we adapt them by extracting vertex embeddings from the input graph and cluster them to obtain representative embeddings for each vertex type in these two methods.

Appendix D-3: Effect of Various Pattern Mining Methods

Advanced pattern mining algorithms can also be integrated into our motif identification process, *e.g.,* gSpan [76] that introduces a lexicographic order among graphs and performs depth-first search to efficiently enumerate frequent connected subgraphs, FSG [106] that employs an Apriori-style candidate generation strategy based on subgraph isomorphism tests, and MoFa [105] that applies a pattern-growth framework using canonical forms. We tested the impact of mining methods on rule discovery, including GASTON [69] (our default).

Figure 11 shows the recognizability of RxGNNs obtained with different mining methods. Because all abstract patterns

originate from the same set of subgraphs, with the same support requirement, their recognizability scores are largely consistent. However, these methods differ in efficiency. Table VII shows their running time. On all datasets except Trans, GASTON runs the fastest among all methods due to its layered growth scheme that incrementally extends frequent paths into trees and cyclic graphs. On dataset Trans that has a small number of patterns, FSG is slightly faster than GASTON since it generates all the candidate patterns before validation while GASTON generates the patterns via a hierarchical schema.

Considering the stable and superior efficiency of GASTON, we used it as our default primitive extractor.