SAPIENZA
UNIVERSITÀ DI ROMA

Graph Counterfactual Explanations via Gradient Perturbations.

Facoltà di Ingegneria dell'informazione, informatica e statistica

Corso di Laurea Magistrale in Computer Science

Candidate

Riccardo De Sanctis
ID number 1937859

Thesis Advisor

Prof. Stefano Faralli

Co-Advisor

Dr. Bardh Prenkaj

Academic Year 2024/2025

Graph Counterfactual Explanations via Gradient Perturbations.
Master's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: desanctis.1937859@studenti.uniroma1.it

Dedicated to my family

Abstract

Graph Neural Networks (GNNs) are increasingly adopted in domains like molecular biology and social network analysis, yet their black-box nature hinders interpretability and trust. This is especially problematic in high-stakes applications, such as predicting molecule toxicity, drug discovery, or guiding financial fraud detections, where transparent explanations are essential. Counterfactual explanations – minimal changes that flip a model's prediction – offer a transparent lens into graph neural networks' behaviour. This manuscript introduces and proposes an explainer method named XPlore, a novel technique that significantly broadens the counterfactual search space. It consists of gradient-guided perturbations to adjacency and node feature matrices. Unlike most prior methods, which focus solely on edge deletions, the proposed approach belongs to the growing class of techniques that optimize edge insertions and node-feature perturbations, here jointly performed under a unified gradient-based framework, enabling a richer and more nuanced exploration of counterfactuals. To quantify both structural and semantic fidelity, a cosine similarity metric on learned graph embeddings is introduced, addressing a key limitation of traditional distance-based metrics, demonstrating that XPlore produces more coherent and minimal counterfactuals. Empirical results on 13 real-world and 5 synthetic benchmarks show up to +56.3% improvement in validity and +52.8% in fidelity over state-of-the-art baselines, while retaining competitive runtime. Qualitative assessment of XPlore's counterfactual explanation is further carried out to ensure the main objective of the explainer is attained. The study lays the foundations for a study of the oracles' expressive power by analysing XPlore behaviour across numerous state-of-the-art models; shifting completely the burden of the generalization capabilities from the explainer to the oracle itself, moreover allowing to focus and to invest the entirety of the computing resources on the training and production of powerful models that can be easily explained and corrected with a fast and light-weight explainer.

Contents

Chapter 1

Introduction

Explainability and interpretability of the rapidly growing number of automated decision-making systems have become central challenges for ensuring safety, security, and fairness in modern society. Recent advancements in artificial intelligence and, in particular, deep learning have enabled machines to tackle complex tasks once considered far beyond computational reach. Yet, as these systems increasingly replace or complement human decision-makers their opacity raises significant regulatory, ethical, and security concerns. Even tough the hardware, model architectures, and training pipelines behind modern artificial intelligence systems are often documented and publicly accessible, their internal workings and decision process remain largely obscure and inscrutable. Understanding how and why a model arrives at a specific prediction is difficult, sometimes impossible, even for domain experts.

This lack of transparency becomes especially problematic when artificial intelligence models are entrusted with high-stakes tasks, where fairness, reliability, and accountability are essential. When the decision process is opaque, it becomes harder to guarantee fairness, legitimacy, and reliability, and justify model outputs. Therefore, developing simple and effective tools that allow us to probe and sieve the model behaviour is a critical step toward a trustworthy artificial intelligence. Explainability methods offer meaningful insights and information about the decision-making process of a model, help identify structural flaws or unintended shortcuts in its design, and support the detection of corrupted, biased, or otherwise compromised training data. Such tools not only enhance human ability to validate artificial intelligence systems but also reinforce public confidence of their deployment in critical domains.

1.1 Motivation

Explainability is crucial in fields such as healthcare and finance, where transparent and accountable decisions are necessary (Guidotti et al., 2018). While deep neural networks are powerful, Petch et al. (2021) point out that their black-box nature limits interpretability, hindering trust in sensitive applications. White-box models are more interpretable (Loyola-González, 2019) and, as Verenich et al. (2019) discusses, are often better suited for regulated settings; however, they frequently underperform compared to black-box models on complex, high-dimensional tasks (Aragona et al., 2021; Ding et al., 2019).

GNNs have gained significant attention (Wei et al., 2022), but like other deep learning

models, they lack interpretability. To mitigate this, post-hoc methods, particularly counterfactual explanations, aim to reveal how input changes affect predictions. Recently, Graph Counterfactual Explainability has recently emerged as a key area (Prado-Romero et al., 2023c).

Consider a research collaboration network, where nodes represent scientists, and edges indicate co-authorships on published papers. Suppose a young researcher, James, is recommended for a prestigious grant based on their position in the collaboration graph. A counterfactual explanation might state:

> *"If James had not co-authored a paper with Professor Y, he would not have been considered for the grant."*

This explanation highlights how specific connections within the graph structure influence high-stakes decisions. The counterfactual framework reveals how adding or removing certain collaborations alters the model's prediction, *offering insights into network-driven career advancements and systemic biases in academic recognition* – see fig. 1.1.

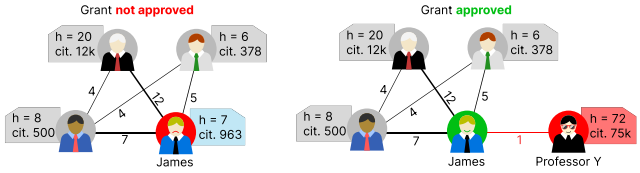


Figure 1.1. (left) James works with his research lab in publishing scientifically good papers with decent impact for the community; however, his grant application is not approved. (right) James writes a single, not-so-interesting paper with Professor Y, who happens to be highly influential within a broader community, which helps make James's grant application more likely to be approved. *James is happy, but at what cost?*

GCE techniques can be broadly categorized into search-based, heuristic-based, and learning-based approaches (Prado-Romero et al., 2023c). Search-based methods identify counterfactuals by searching within the existing data distribution. Heuristic-based approaches modify the input graph G to create a perturbed version G', ensuring that a prediction model Φ produces different classifications ($\Phi(G) \neq \Phi(G')$). These methods, however, require carefully designed heuristics, which often rely on domain expertise. For instance, generating chemically valid counterfactuals for molecular graphs necessitates an understanding of atomic valences and bonding rules. Learning-based strategies, in contrast, automate the discovery of meaningful perturbations by training on data, allowing for the generation of counterfactual examples at inference time.

1.2 Problem Statement

The main limitation of deep learning models is their inherent opacity: their powerful task solving and predictive capabilities often come at the expense of intelligibility. After the training phase, it is not possible to easily inspect or interpret the internal state of these systems, which have a complex and high-dimensional representation. Although their strong performance is typically obtained through empirical experimentation and heuristic optimization, designed to replicate the human way of learning and solving problems, their learned patterns often diverge substantially from human intuition. As a consequence, these models may rely on spurious correlations or unexpected cues, behaving differently from what practitioners would anticipate. Existing explanation methods, developed to shed light on such internal decision processes, frequently produce unstable, incomplete, or contradictory explanations.

Many explainers are themselves based on complex deep models, adding another layer of opacity rather than reducing it. As a result, explanations in critical decision systems may still appear unreliable, biased, or uncertain, compromising trust in the underlying model.

Graph neural network explainers, despite their success on relational and structured data, inherit these shortcomings and introduce additional challenges due to the combinatorial nature of graph inputs. Current graph neural network explainers exhibit several limitations. Explanation subgraphs are often noisy or excessively large, failing to capture the minimal and faithful structures responsible for the model's prediction. They rarely provide a contrastive or counterfactual insight, useful for clarifying the minimal change required for altering the model decision. Many methods lacks structure controllability: it is not possible to constrain specific components, regulate sparsity, or ensure that explanations remain within a desired graph edit distance. Furthermore, most approaches lack theoretical grounding, relying on heuristics without guarantees regarding faithfulness, stability, or robustness. Existing explainers are also susceptible to adversarial manipulation and overlook fairness considerations, potentially producing biased or misleading rationales.

Another critical limitation concerns semantic consistency. Explanations are typically evaluated at the structural level, without assessing whether proposed counterfactuals correspond to semantically coherent or meaningful instances. As a result, counterfactual graphs may be nearly identical in structure yet represent entirely different semantic concepts, or conversely, differ substantially despite encoding very similar meanings; both of which undermine the validity of the explanation.

Real-world graph learning tasks span heterogeneous domains such as chemistry, social networks, finance, computer vision, and bioinformatics. Many graph explainers have not been empirically evaluated on sufficiently diverse benchmarks and datasets, resulting in limited evidence of their practical applicability. These settings demand explainers that are scalable, theoretically grounded, user-controllable, and capable of generating interpretable structure-level explanations. Without reliable interpretability tools, decisions based on graph neural networks risk being unsafe, susceptible to adversarial interference, and potentially biased or discriminatory.

1.3 Research Objectives

This thesis aims at improving interpretability, security, and fairness of predictive graph based machine learning models by addressing key shortcoming in current state-of-the-art methods. To this end, the thesis introduces XPlore, a novel graph neural network explainer designed to provide reliable, easily interpretable, theoretically motivated, and computationally efficient explanations. XPlore is supported by both a formal characterisation of its behaviour and extensive empirical studies on real-world and synthetic datasets.

In parallel, the thesis contributes to the GRETEL counterfactual explanation evaluation framework by:

1. Extending its scope to node-level explanation tasks.
2. Integrating strong state-of-the-art explainer baselines.
3. Designing and implementing generative pipelines for emblematic synthetic graph datasets and integrating real-world ones.

Together, these objectives advance the methodological foundations of graph explainability and provide practical tools for more trustworthy and transparent graph neural network based decision systems.

1.4 Contributions

This thesis work introduces XPlore, a substantial advancement over Lucic et al. (2022), designed to enhance counterfactual generation for graph neural networks. Unlike the original method, which modifies graphs solely through edge deletion, XPlore introduces additional flexibility by incorporating edge additions and node feature perturbations. This expansion allows for a more comprehensive exploration of graph modifications, improving the quality of counterfactual examples. The proposed approach employs a specialized loss function that autonomously guides perturbations, minimizing unnecessary modifications, a huge improvement over heuristic-driven methods. Moreover, rather than incorporating generative modeling into the learning objective, XPlore directly leverages gradient-based optimization to identify the closest counterfactual instance efficiently. This results in greater transparency: under access to oracle gradients, the entire explainer remains fully interpretable, avoiding the additional black-box component, and keeping the explanation pipeline lightweight, faithful to the classifier's decision boundary, and easier to audit.

The contributions of the thesis are as follows.

1. **Comprehensive Input Perturbation:** the proposed framework supports edge additions, edge removals, and node feature perturbations, enabling perturbation of nearly all input degrees of freedom. This richer perturbation space allows us to explore how structural and attribute-level changes jointly influence the oracle's prediction process and to uncover richer, truly counterfactual motifs that edge-only methods cannot express.

2. **Gradient-Guided Optimization:** by exploiting the properties of the loss, it is possible to find counterfactuals that are locally closest in the loss-optimization landscape, as guided by directed gradient-based modifications (3.5.1). This guarantees that the counterfactual explanation is not only minimal under the objective but also consistent with the oracle's learned decision boundaries.

3. **Mitigation of Out-of-Distribution Issues:** a cosine similarity metric is introduced to quantify both structural and semantic fidelity of counterfactuals. Together with the use of edge additions and feature perturbations, this mitigates out-of-distribution artifacts (Chen et al., 2023) and preserves class-relevant discriminative cues. The discussion follows in section 4.5.

4. **Domain coverage and Strong Empirical Performance:** the proposed method is evaluated on a diverse suite of real-world and synthetic graph benchmarks, and it is shown that it consistently outperforms state-of-the-art methods in validity and fidelity while maintaining competitive runtime and semantic coherence.

1.5 Manuscript Structure

This **Introduction** chapter (chapter 1) presented the **Motivation** (section 1.1), **Problem Statement** (section 1.2), **Research Objectives** (section 1.3) and **Con-**

tributions (section 1.4) sections of this thesis work. The **manuscript structure** of the work is outlined below.

The **Theoretical Framework** chapter (chapter 2) covers the background, literature review, and related works; it is organized as follows:

- The **Background** section (section 2.1) introduces the necessary foundational concepts and definitions used throughout the thesis.
- The **Literature Review** section (section 2.2) provides a broad survey of the field.
- The **Related Works** section (section 2.3) presents methods directly comparable to the one introduced in this thesis.

The **Method** chapter (chapter 3) reports the main theoretical definitions, methodologies, and techniques adopted in this thesis work; it is organized as follows:

- The **Problem Formulation** section (section 3.1) defines the explainability problem formulation.
- The **Proposed Method** section (section 3.2) introduces and formally defines XPlore.
- The **Algorithmic Implementation** section (section 3.3) provides the reproducibility pipeline for the proposed method.
- The **Oracles Models** section (section 3.4) reports the implemented architectures definitions and details.
- The **Theoretical Foundations of Gradient-Guided Counterfactual Perturbations** section 3.5 lays out the formal support for the proposed explanation method.

The **Experimental Evaluation** chapter (chapter 4) discusses the configuration and the empirical results obtained; it is organized as follows:

- The **Setup** section (section 4.1) covers the general framework settings used for the explainability experiments.
- The **Datasets** section (section 4.2) presents the benchmarks adopted for the empirical assessment.
- The **Configuration** section (section 4.3) illustrate the specific settings, hardware and metrics used for the experiments.
- The **Results** section (section 4.4) reports and discusses XPlore results.
- The **Residual Out-of-distribution Influence** section (section 4.5) studies XPlore's behaviour and semantic quality.
- The **Interpretability** section (section 4.6) presents valuable intelligibility results of XPlore's inspection of the oracle's models.
- The **Ablation Studies** section (section 4.7) provides additional insights into the configuration, results, and behaviour of the proposed method.

The **Conclusion** chapter (chapter 5) summarizes the work presented into the manuscript and its contributions, and outlines the future research direction; it is organized as follows:

- The **Future Work** section (section 5.1) defines the future research line and how to augment XPlore utilizations and support.
- The **Summary** section (section 5.2) synthesizes the advantages and results of the method proposed in this manuscript.

The **Bibliography** (section 5.2) reports all the sources cited and referred to in this manuscript.

Chapter 2

Theoretical Framework

This chapter covers the background, literature review, and related works. It is organized as follows:

- The **background** section (section 2.1) introduces the necessary foundational concepts and definitions used throughout the thesis. In particular it covers: artificial neural networks, graphs, graph neural networks and their variants, diffusion model, graph diffusion models, explainability, and GRETEL.
- The **review of the literature** section (section 2.2) provides a broad survey of the field, graph explainers and their applications.
- The **related works** section (section 2.3) presents methods directly comparable to the one introduced in this thesis.

2.1 Background

2.1.1 Artificial Neural Network

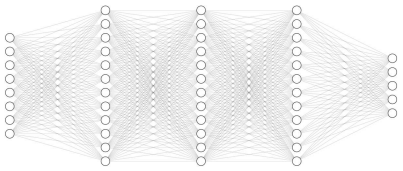


Figure 2.1. Fully Connected Feed Forward Neural Network (FCNN).

Artificial Neural Networks (NN) are computational models inspired by the biological Neural Networks structure and functioning. A formal definition provided by E.Fiesler**?**, that applies to both natural and artificial Neural Network, is presented in summary:

$$\mathcal{N} = (T, C, S(0), \Phi) \qquad (2.1.1.1)$$

- T is the topology which consists of the framework and interconnection structure.
- C is the set of constraints associated with the neural network. Constraints include limitations on parameters, connectivity, or other properties relevant to the network's behavior.
- *S(0)* is the set of initialization states, indicating the initial states or condition of the neural network. It refer to the starting point of the network, considering initial weights, initial activation and initial local threshold.

- Φ is the set of transition functions, describing how the neural network evolves from one state to another over time. Transition functions describe the behaviour of neurons output and the learning rules, how parameters are updated.

A neural network can been seen as a parameterized function that taken a D-dimensional input, produces a D'-dimensional output, $\mathcal{N}_\theta : \mathbb{R}^D \to \mathbb{R}^{D'}$, parameterized by θ.

2.1.2 Convolutional Neural Network

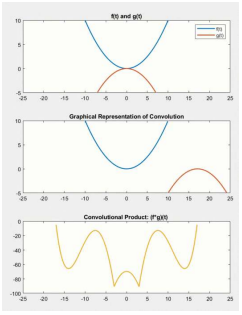


Figure 2.2. Continuous Convolution.

Convolution is a mathematical operation which combines two functions to describe the overlap between them. Convolution creates a new function by sliding one of the input function over the other one, multiplying the function values at each point where they overlap, and adding up the products. The resulting function is representative of the interaction between the two original functions. Formally, convolution is an integral that expresses the amount of overlap of one function $f(t)$ as it is shifted over function $g(t)$:

$$f * g)(t) =^{def} \int_{-\infty}^{\infty} f(\tau) \cdot g(t-\tau)\, d\tau = \int_{-\infty}^{\infty} f(t-\tau) \cdot g(\tau)\, d\tau \qquad (2.1.2.1)$$

The convolution operation is commutative. Convolution can be applied also in the discrete case, such as those encountered in Machine Learning problems. Let *f, g* be defined on the set $\mathbb{Z}$ of integers. The discrete convolution of f and g is given by:

$$(f * g)[n] =^{def} \sum_{m=-\infty}^{\infty} f[m] \cdot g[n-m] = \sum_{m=-\infty}^{\infty} f[m-n] \cdot g[n] \qquad (2.1.2.2)$$

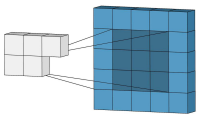


Figure 2.3. 2-d Discrete Convolution.

Convolutional neural networks are based on the convolution operator. They were inspired by the visual cortex of animals where individual cortical neurons respond to stimuli only in a restricted region of the visual field, known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field. This behaviour is replicated in CNN by using a convolution filter (or kernel) that slides over the whole input features. The kernels possess shared weights for the whole layer. This consistently reduces the number of parameters, allowing the network to have more layers and go deeper. CNNs particularly excels in domains such as image analysis, natural language processing and signal processing, among others.

CNN architecture consists of an input layer, hidden layers that perform convolutions and an output layer. The process involves the dot product between the input layer's matrix and the convolution kernel, followed by an activation function. The convolution kernel slides along the input matrix of the layer, generating a feature map (or activation map) through the convolution operation. This feature map then contributes to the input of the next layer. Subsequent layers, such as pooling layers, fully connected layers, and normalization layers, could then be employed. The

output resulting from a kernel slide exhibits lower dimensionality than the input, therefore padding is employed. This technique guarantees dimensionality consistency by introducing values, typically zeroes, to the boundaries of the data.

Sampling

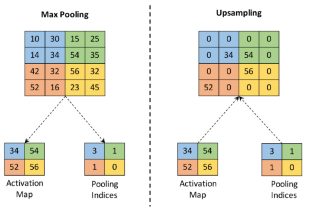


Figure 2.4. Max Pooling and Up-sampling techniques illustration.

Different down-sampling and up-sampling techniques are employed within CNN layers. Max Pooling is a down-sampling technique frequently employed in convolutional neural networks (CNNs) to diminish the spatial dimensions of an input volume. This non-linear down-sampling method aims to make the representation smaller and more manageable, while concurrently reducing the number of parameters and computational load in the network. Max pooling operates independently on each depth slice of the input, resizing it spatially. There are three primary advantages to Max Pooling:

- Feature Invariance: Max pooling helps the model to become invariant to the location and orientation of features.
- Dimensionality Reduction: By down-sampling the input, max pooling significantly reduces the number of parameters and computations in the network, thus speeding up the learning process and reducing the risk of overfitting.
- Noise Suppression: Max pooling helps to suppress noise in the input data. By taking the maximum value within the window, it emphasizes the presence of strong features and diminishes the weaker ones.

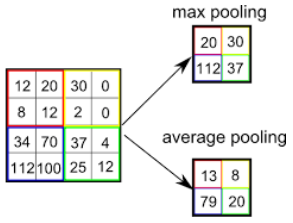


Figure 2.5. Max Pooling and Average Pooling illustration.

Max pooling is applied to the convolutional layers of a CNN. This process entails sliding a window (commonly referred to as a filter or kernel) across the input data, similarly to the convolution step. However, instead of conducting a matrix multiplication, max pooling selects the maximum value within the window. The inverse operation of Max Pooling is the upsampling interpolation. An alternative to Max Pooling is Average Pooling, where the average of the values whitin the window is selected.

2.1.3 Graph Neural Networks

Multiple surveys (Wu et al., 2020; Kriege et al., 2020; Zhou et al., 2020; Liang et al., 2022; Corso et al., 2024) studied and categorized graph neural networks.

Graph neural networks (GNNs) history. The first neural network application to directed acyclic graph by Sperduti and Starita (1997) motivated early studies on graph neural networks, which notion was initially outlined in Gori et al. (2005), and further elaborated in Scarselli et al. (2008) and Gallicchio and Micheli (2010). They learn a target node's representation by propagating neighbour information in an iterative manner until a stable fixed point is reached. Since the process is computationally expensive, there are have been consequent increasing efforts to

overcome these challenges: Li et al. (2015) used gated recurrent units and Dai et al. (2018) used steady-states learning of iterative algorithms.

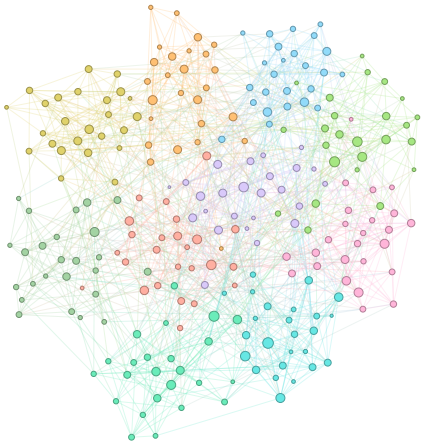


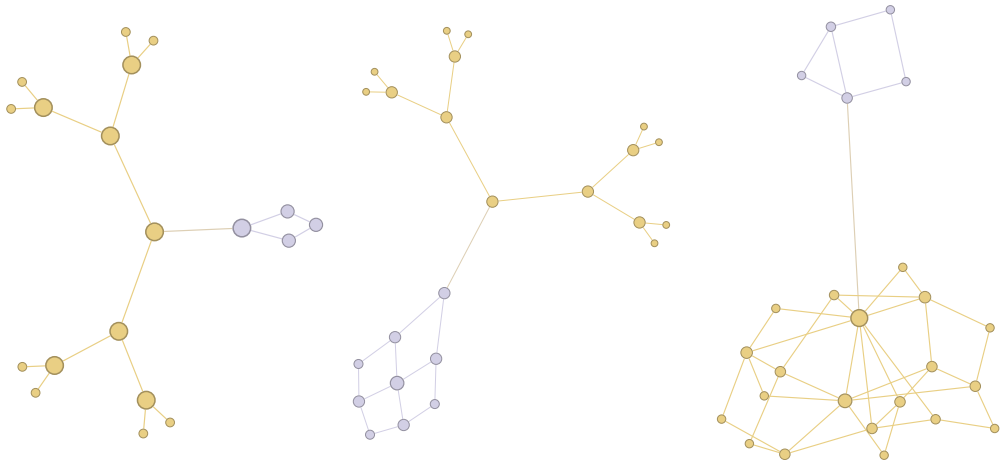
Figure 2.6. A Stochastic Block Model (SBM) graph.

Encouraged by the success of CNNs in the computer vision domain, a large number of methods that re-define the notion of convolution for graph data were developed in parallel. These approaches are under the umbrella of convolutional graph neural networks (ConvGNNs). ConvGNNs are divided into two main streams, the spectral-based approaches and the spatial-based approaches. The first prominent research on spectral-based ConvGNNs was presented by Bruna et al. (2013), which developed a graph convolution based on the spectral graph theory. Since then, there have been increasing improvements, extensions, and approximations on spectral-based ConvGNNs (Henaff et al., 2015; Defferrard et al., 2016; Kipf and Welling, 2017; Levie et al., 2018). The research of spatial-based ConvGNNs started much earlier than spectral-based ConvGNNs. In 2009, Micheli (2009) first addressed graph mutual dependency by architecturally composite nonrecursive layers while inheriting ideas of message passing from recurrent graph neural networks (RecGNNs). However, the importance of this work was overlooked. Until recently, many spatial-based ConvGNNs (Atwood and Towsley, 2016; Niepert et al., 2016; Gilmer et al., 2017) emerged. The timeline of representative RecGNNs and ConvGNNs is shown in the first column of Table II. Apart from RecGNNs and ConvGNNs, many alternative GNNs have been developed in the past few years, including graph autoencoders (GAEs) and spatial-temporal graph neural networks (STGNNs). These learning frameworks can be built on RecGNNs, ConvGNNs, or other neural architectures for graph modeling.

Network embeddings. The research on GNNs is closely related to graph embedding or network embedding, another topic which attracts increasing attention from both the data mining and machine learning communities (Hamilton et al., 2017; Cui et al., 2018; Zhang et al., 2018; Cai et al., 2018; Goyal and Ferrara, 2018; Pan et al., 2016). Network embedding aims at representing network nodes as low-dimensional vector representations, preserving both network topology structure and node content information, so that any subsequent graph analytics task such as classification, clustering, and recommendation can be easily performed using simple off-the-shelf machine learning algorithms (e.g., support vector machines for classification). Meanwhile, GNNs are deep learning models aiming at addressing graph-related tasks in an end-to-end manner. Many GNNs explicitly extract high-level representations. The main distinction between GNNs and network embedding is that GNNs are a group of neural network models which are designed for various tasks while network embedding covers various kinds of methods targeting the same task. Therefore, GNNs can address the network embedding problem through a graph autoencoder framework. On the other hand, network embedding contains other non-deep learning methods such as matrix factorization (Shen et al., 2018; Yang et al., 2018) and random walks (Perozzi et al., 2014).

Graph kernels methods. Graph kernels are historically dominant techniques to solve the problem of graph classification (Vishwanathan et al., 2010; Shervashidze et al., 2011; Navarin et al., 2017). These methods employ a kernel function to measure the similarity between pairs of graphs so that kernel-based algorithms like support vector machines can be used for supervised learning on graphs. Similar to GNNs, graph kernels can embed graphs or nodes into vector spaces by a mapping function. The difference is that this mapping function is deterministic rather than learnable. Due to a pair-wise similarity calculation, graph kernel methods suffer significantly from computational bottlenecks. GNNs, on one hand, directly perform graph classification based on the extracted graph representations and therefore are much more efficient than graph kernel methods. Kriege et al. (2020) conducted a comprehensive review of graph kernel methods.

Graph definition. A graph is represented as $G = (V, E)$ where V is the set of vertices or nodes, and E is the set of edges. Let $v_i \in V$ to denote a node and $e_{ij} = (v_i, v_j) \in E$ to denote an edge pointing from v_j to v_i . The neighbourhood of a node v is defined as $N(v) = \{u \in V | (v, u) \in E\}$. The adjacency matrix A is a $n \times n$ matrix with $A_{ij} = 1$ if $e_{ij} \in E$ and $A_{ij} = 0$ if $e_{ij} \notin E$. A graph may have node attributes X, where $X \in \mathbb{R}^{n \times d}$ is a node feature matrix with $x_v \in \mathbb{R}^d$ representing the feature vector of a node v. Meanwhile, a graph may have edge attributes X^e , where $X^e \in \mathbb{R}^{m \times c}$ is an edge feature matrix with $x^e_{v,u \in \mathbb{R}^c}$ representing the feature vector of an edge (v, u).



(a) Tree Cycles Rand. **(b)** Tree Grid. **(c)** Barabási-Albert Shape.

Figure 2.7. Illustration of synthetic graphs section 4.2.

Directed graph definition. A directed graph is a graph with all edges directed from one node to another. An undirected graph is considered as a special case of directed graphs where there is a pair of edges with inverse directions if two nodes are connected. A graph is undirected if and only if the adjacency matrix is symmetric.

Spatial-temporal graph definition. A spatial-temporal graph is an attributed graph where the node attributes change dynamically over time. The spatial-temporal graph is defined as $G(t) = (V, E, X^{(t)})$ with $X^{(t)} \in R^{n \times d}$.

Message passing. A graph neural network model M embeds each node $v \in V$ into a low-dimensional space $\mathbf{Z} : \mathbb{R}^{n \times d}$ by following this message passing rule for k steps as

$$\mathbf{Z}_v^{(k+1)} = \text{UPDATE}_\Phi(\mathbf{Z}_v^{(k)}, \text{AGG}(\{\text{MSG}_\Theta(\mathbf{Z}_v^{(k)}, \mathbf{Z}_u^k) : (u, v) \in E\})) \tag{2.1.3.1}$$

such that $\mathbf{Z}^0 = \mathbf{X}$ and $\mathbf{Z} := \mathbf{Z}^{(k)}$. Different instances of the update UPDATE_Θ, aggregation AGG_Φ and message generator MSG_Θ functions give rise to different GNN architectures. For example, GCN (Kipf and Welling, 2017) has an identity message, a mean aggregation, and weighted update functions, while GAT (Velickovic et al., 2017) learns an attention-based message generation instead. These embeddings are trained for a specific task $\mathcal{T}$ that can be either supervised (e.g., node classification, graph classification, etc.) or unsupervised (e.g., self-supervised link prediction, clustering, etc.).

2.1.4 Graph Convolution Networks

This subsection provides theoretical motivation for a specific graph-based convolution neural network introduced by Kipf and Welling (2017). The layer-wise propagation rule of a multi-layer Graph Convolutional Network can be defined as:

$$H^{(l+1)} = \sigma\Big(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\Big). \tag{2.1.4.1}$$

Here, $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph $\mathcal{G}$ with added self-connections. I_N is the identity matrix, $\tilde{D}$ is the degree matrix with $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, and $W^{(l)}$ is a layer-specific trainable weight matrix. $\sigma(\mathring{u})$ denotes an activation function, such as the $\text{ReLU}(\cdot) = max(0, \cdot)$. $H^{(l)} \in \mathbb{R}^{N \times D}$ is the matrix of activations in the l^{th} layer; $H^{(0)} = X$. In the following, we show that the form of this propagation rule can be motivated via a first-order approximation of localized spectral filters on graphs (Hammond et al., 2011; Defferrard et al., 2016).

Spectral Graph Convolution

Consider spectral convolutions on graphs defined as the multiplication of a signal $x \in \mathbb{R}^N$ (a scalar for every node) with a filter $g_\theta = \text{diag}(\theta)$ parameterized by $\theta \in \mathbb{R}^N$ in the Fourier domain, i.e.:

$$g_\theta \star x = U g_\theta U^\mathsf{T} x, \tag{2.1.4.2}$$

where U is the matrix of eigenvectors of the normalized graph Laplacian $L = I_N - D^{\frac{1}{2}}AD^{\frac{1}{2}} = U\Lambda U^\mathsf{T}$, with a diagonal matrix of its eigenvalues Λ and $U^\mathsf{T}x$ being the graph Fourier transform of x. It is possible to understand g_θ as a function of the eigenvalues of L, i.e. $g_\theta(\Lambda)$. Evaluating eq. (2.1.4.2) is computationally expensive, as multiplication with the eigenvector matrix U is $\mathcal{O}(N^2)$. Furthermore, computing the eigendecomposition of L in the first place might be prohibitively expensive for large graphs. To circumvent this problem, it was suggested in Hammond et al. (2011) that $g_\theta(\Lambda)$ can be well-approximated by a truncated expansion in terms of Chebyshev polynomials $T_k(x)$ up to K^{th} order:

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x, \tag{2.1.4.3}$$

with $\tilde{L} = \frac{2}{\lambda_{\max}} L - I_N$; as can easily be verified by noticing that $(U\Lambda U^\mathsf{T})k = U\Lambda^k U^\mathsf{T}$. Note that this expression is now K-localized since it is a K^{th}-order polynomial in the Laplacian, i.e. it depends only on nodes that are at maximum K steps away from the central node (K^{th}-order neighborhood). The complexity of evaluating eq. (2.1.4.3) is $\mathcal{O}(|\mathcal{E}|)$, i.e. linear in the number of edges. Defferrard et al. (2016) use this K-localized convolution to define a convolutional neural network on graphs.

Layer-Wise Linear Model

A neural network model based on graph convolutions can therefore be built by stacking multiple convolutional layers of the form of eq. (2.1.4.3), each layer followed by a point-wise non-linearity. Now, if the layer-wise convolution operation is limited to $K = 1$ (see eq. (2.1.4.3)), i.e. a function that is linear with respect to L and therefore a linear function on the graph Laplacian spectrum.

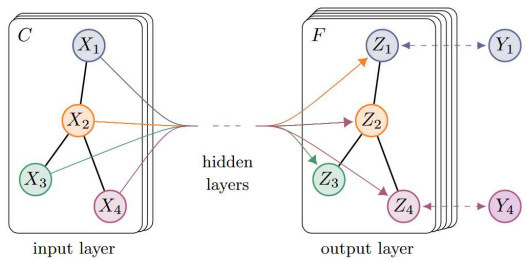


Figure 2.8. Multi-layer GCN for semisupervised learning. The graph structure is shared over layers.

In this way, it is possible to still recover a rich class of convolutional filter functions by stacking multiple such layers, but we are not limited to the explicit parameterization given by, e.g., the Chebyshev polynomials. We intuitively expect that such a model can alleviate the problem of overfitting on local neighborhood structures for graphs with very wide node degree distributions, such as social networks, citation networks, knowledge graphs and many other real-world graph datasets. Additionally, for a fixed computational budget, this layer-wise linear formulation allows us to build deeper models, a practice that is known to improve modeling capacity on a number of domains (He et al., 2016).

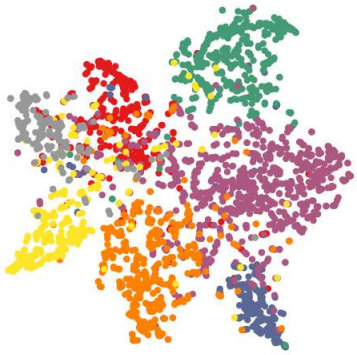


Figure 2.9. t-SNE (der Maaten, 2008) visualization of hidden layer activations of a two-layer GCN.

In this linear formulation of a graph convolution network it is possible to further approximate $\lambda_{\max} \approx 2$, as it is possible to expect that neural network parameters will adapt to this change in scale during training. Under these approximations eq. (2.1.4.3) simplifies to:

$$g_{\theta'} \star x \approx \theta'_1(L - I_N)x = \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x, \tag{2.1.4.4}$$

with two free parameters θ'_0 and θ'_1. The filter parameters can be shared over the whole graph. Successive application of filters of this form then effectively convolve the k$^{\text{th}}$-order neighbourhood of a node, where k is the number of successive filtering operations or convolutional layers in the neural network model. In practice, it can be beneficial to constrain the number of parameters further to address overfitting and to minimize the number of operations (such as matrix multiplications) per layer. This leaves us with the following expression:

$$g_\theta \star x \approx \theta\Big(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}\Big)x, \tag{2.1.4.5}$$

with a single parameter $\theta = \theta'_0 = -\theta'_1$. Note that $IN + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ now has eigenvalues in the range $[0, 2]$. Repeated application of this operator can therefore lead to numerical instabilities and exploding/vanishing gradients when used in a deep neural network model. To alleviate this problem, the following renormalization trick is introduced: $I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}}A\tilde{D}^{-\frac{1}{2}}$, with $\tilde{A} = A + I_N$ and $D_{ii} = \sum_j \tilde{A}_{ij}$. We can generalize this definition to a signal $X \in \mathbb{R}^{N \times C}$ with C input channels (i.e. a C-dimensional feature vector for every node) and F filters or feature maps as follows:

$$Z = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta, \tag{2.1.4.6}$$

where $\Theta \in \mathbb{R}^{C \times F}$ is now a matrix of filter parameters and $Z \in \mathbb{R}^{N \times F}$ is the convolved signal matrix. This filtering operation has complexity $\mathcal{O}(|\mathcal{E}|FC)$, as $\tilde{A}X$ can be efficiently implemented as a product of a sparse matrix with a dense matrix.

2.1.5 Graph Attention Networks

This section presents the building block layer used to construct arbitrary graph attention networks (GATs) (through stacking this layer) in the domain of neural graph processing introduced by Velickovic et al. (2017); Wang et al. (2021c).

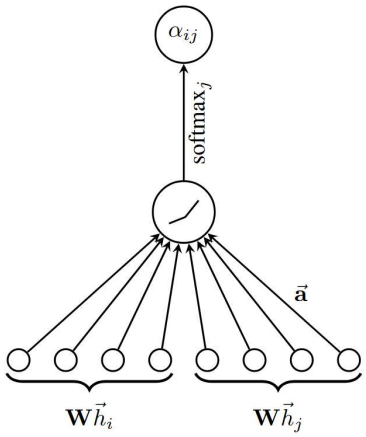


Figure 2.10. The attention mechanism $a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$, parametrized by a weight vector $\vec{a} \in \mathbb{R}^{2F'}$, applying a LeakyReLU activation.

Graph Attention Layer The single graph attentional layer is the building block layer utilized in the graph attention network architectures. The particular attentional setup utilized closely follows the work of Bahdanau et al. (2014) but the framework is agnostic to the particular choice of attention mechanism.

The input to the layer is a set of node features, $\mathbf{h} = \{\vec{h}_1, \vec{h}_2, ..., \vec{h}_N\}, \vec{h}_i \in \mathbb{R}^F$, where N is the number of nodes, and F is the number of features in each node. The layer produces a new set of node features (of potentially different cardinality F'), $\mathbf{h}' = \{\vec{h}'_1, \vec{h}'_2, ..., \vec{h}'_N\}, \vec{h}'_i \in \mathbb{R}^{F'}$, as its output.

In order to obtain sufficient expressive power to transform the input features into higher-level features, at least one learnable linear transformation is required. To that end, as an initial step, a shared linear transformation, parametrized by a *weight matrix*, $\mathbf{W} \in \mathbb{R}^{F' \times F}$, is applied to every node. Then *self-attention* is performed on the nodes, a shared attentional mechanism $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \rightarrow \mathbb{R}$ computes *attention coefficients*

$$e_{ij} = a\Big(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j\Big) \tag{2.1.5.1}$$

that indicate the *importance* of node j's features to node i. In its most general formulation, the model allows every node to attend on every other node, *dropping all structural information*. The graph structure is injected into the mechanism by performing masked attention, e_{ij} are only computed for nodes $j \in \mathcal{N}_i$, where $\mathcal{N}_i$ is some neighbourhood of node i in the graph. These will be exactly the first-order

neighbours of i (including i). To make coefficients easily comparable across different nodes, they are normalized across all choices of j using the softmax function:

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})} \tag{2.1.5.2}$$

the attention mechanism a is a single-layer feed-forward neural network, parametrized by a weight vector $\vec{a} \in \mathbb{R}^{2F'}$, and applying the LeakyReLU nonlinearity. Fully expanded out, the coefficients computed by the attention mechanism (fig. 2.10) may then be expressed as:

$$\alpha_{ij} = \frac{\exp\Big(\text{LeakyReLU}\Big(\vec{a}^T[\mathbf{W}\vec{h}_i || \mathbf{W}\vec{h}_j]\Big)\Big)}{\sum_{k \in \mathcal{N}_i} \exp\Big(\text{LeakyReLU}\Big(\vec{a}^T[\mathbf{W}\vec{h}_i || \mathbf{W}\vec{h}_k]\Big)\Big)} \tag{2.1.5.3}$$

where $\cdot^T$ represents transposition and $||$ is the concatenation operation.

Once obtained, the normalized attention coefficients are used to compute a linear combination of the features corresponding to them, to serve as the final output features for every node (after potentially applying a nonlinearity, σ):

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W}\vec{h}_j\right). \tag{2.1.5.4}$$

To stabilize the learning process of self-attention, extending the mechanism to employ *multi-head attention* to be beneficial, similarly to Vaswani et al. (2017). Specifically, K independent attention mechanisms execute the transformation of eq. (2.1.5.10), and then their features are concatenated, resulting in the following output feature representation:

$$\vec{h}'_i = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right) \tag{2.1.5.5}$$

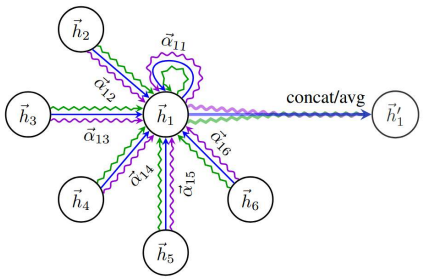


Figure 2.11. Multi-head attention, $K = 3$ independent heads. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}'_1$.

where $||$ represents concatenation, α_{ij}^k are normalized attention coefficients computed by the k-th attention mechanism (a^k), and $\mathbf{W}^k$ is the corresponding input linear transformation's weight matrix. Note that, in this setting, the final returned output, $\mathbf{h}'$, will consist of KF' features (rather than F') for each node.

Specially, if multi-head attention is performed on the final (prediction) layer of the network, concatenation is no longer sensible, instead, *averaging* is employed, and delay applying the final nonlinearity (usually a softmax or logistic sigmoid for classification problems) until then:

$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right) \tag{2.1.5.6}$$

The aggregation process of a multi-head graph attentional layer is illustrated by fig. 2.11.

Edge-Featured Graph Attention Layer

Wang et al. (2021c) extend the graph attention network architecture by providing the edge-featured graph attention networks (EGAT), implementing an edge attention block. Each EGAT layer is designed in a symmetrical scheme; thus, the node and edge features can update themselves in a parallel and equivalent way. Together with the node features, the layer accept a set of edge features, $E = \{\vec{e}_1, \vec{e}_2 ..., \vec{e}_M\}, \vec{e}_p \in \mathbb{R}^{F_E}$, as inputs. M represents the number of edges, while F_E symbolizes the number of their features. After processing, the layer will produce high-level outputs, which include a the set of node features and a new set of edge features, $E' = \{\vec{e}'_1, \vec{e}'_2 ..., \vec{e}'_M\}, \vec{e}'_p \in \mathbb{R}^{F'_E}$. The cardinality of F and F' may be different (whether the F is F_H or F_E), since the linear transformations performed on the node and edge features are not same. A new learnable matrices, $W_E \in \mathbb{R}^{F_E} \times F'_E$, is introduced to achieve such transformations for edges. For each node i and edge p, their transformed edge features can be computed by $\vec{e}^*_p = \mathbf{W}_E \, \vec{e}_p$. Equation (2.1.5.9) is extended as follows:

$$\alpha_{ij} = \frac{\exp\Big(\text{LeakyReLU}\Big(\vec{\mathbf{a}}^T[\vec{h}_i || \vec{h}_j || \vec{e_{ij}}]\Big)\Big)}{\sum_{k \in \mathcal{N}_i} \exp\Big(\text{LeakyReLU}\Big(\vec{\mathbf{a}}^T[\vec{h}_i || \vec{h}_k || \vec{e_{ij}}]\Big)\Big)} \tag{2.1.5.7}$$

Equation (2.1.5.10) is extended as follows:

$$\vec{h}'_i = \sigma\Bigg(\sum_{j \in \mathcal{N}_i} \alpha_{ij} (\vec{h}_j || \vec{e}_{ij})\Bigg). \tag{2.1.5.8}$$

Edge attention block. For each edge p, the normalized attention weight of edge q can be expressed as:

$$\beta_{pq} = \frac{\exp\Big(\text{LeakyReLU}\Big(\vec{\mathbf{b}}^T[\vec{e}_p || \vec{e}_q || \vec{h_{pq}}]\Big)\Big)}{\sum_{k \in \mathcal{N}_i} \exp\Big(\text{LeakyReLU}\Big(\vec{\mathbf{b}}^T[\vec{e}_p || \vec{e}_q || \vec{h_{pk}}]\Big)\Big)} \tag{2.1.5.9}$$

where $\mathcal{N}p$ is the first-order neighbour set of edge p (including p itself), and $\vec{\mathbf{b}}$ is a weight vector with a size of $\mathbb{R}^{2F'_E + F'_H}$. The computing of new set of edge features can be represented as:

$$\vec{e}'_p = \sigma\Bigg(\sum_{q \in \mathcal{N}_p} \beta_{pq} \vec{e}_q\Bigg). \tag{2.1.5.10}$$

The edge-featured graph attention network architecture is defined accordingly:

$$\vec{h}^*_i = \Big\Vert_{k=1}^{K} \sigma\Bigg(\Big\Vert_{l=1}^{L} m_i^{l,k}\Bigg) \tag{2.1.5.11}$$

where L indicates the number of the EGAT layers, and $m_i^{l,k}$ represents the edge-integrated node features of node i produced in iteration l of the group k.

2.1.6 Graph Isomorphism Networks

This subsection present Graph Isomorphism Networks (GINs), as developed by Xu et al. (2018) to generalize the Weisfeiler-Lehman (WL) graph isomorphism test (Leman and Weisfeiler, 1968) and achieve maximum discriminative power among graph neural networks.

First, the maximum representational capacity of a general class of GNN-based models is characterized. Ideally, a maximally powerful GNN could distinguish different graph structures by mapping them to different representations in the embedding space. This ability to map any two different graphs to different embeddings, however, implies solving the challenging graph isomorphism problem. That is, we want isomorphic graphs to be mapped to the same representation and non-isomorphic ones to different representations. In the analysis, the representational capacity of GNNs is characterized via a slightly weaker criterion: a powerful heuristic called Weisfeiler-Lehman (WL) graph isomorphism test (Leman and Weisfeiler, 1968), that is known to work well in general, with a few exceptions, e.g., regular graphs (Cai et al., 1992; Douglas, 2011; Evdokimov and Ponomarenko, 1999).

Lemma 2.1.1. *Let G_1 and G_2 be any two non-isomorphic graphs. If a graph neural network $\mathcal{A} : \mathcal{G} \to \mathbb{R}^d$ maps G_1 and G_2 to different embeddings, the Weisfeiler-Lehman graph isomorphism test also decides G_1 and G_2 are not isomorphic.*

Hence, any aggregation-based GNN is at most as powerful as the WL test in distinguishing different graphs. A natural follow-up question is whether there exist GNNs that are, in principle, as powerful as the WL test? The answer, in Theorem 3, is yes: if the neighbor aggregation and graph-level readout functions are injective, then the resulting GNN is as powerful as the WL test.

Theorem 2.1.1. *Let $\mathcal{A} : \mathcal{G} \to \mathbb{R}^d$ be a GNN. With a sufficient number of GNN layers, $\mathcal{A}$ maps any graphs G_1 and G_2 that the Weisfeiler-Lehman test of isomorphism decides as non-isomorphic, to different embeddings if the following conditions hold:*

a) $\mathcal{A}$ aggregates and updates node features iteratively with

$$h_v^{(k)} = \phi\Big(h_v^{(k-1)}, f\Big(\Big\{h_u^{(k-1)} : u \in \mathcal{N}(v)\Big\}\Big)\Big),$$

where the functions f, which operates on multisets, and ϕ are injective.

b) $\mathcal{A}$'s graph-level readout, which operates on the multiset of node features $\{h_v^{(k)}\}$, is injective.

For countable sets, injectiveness well characterizes whether a function preserves the distinctness of inputs. Uncountable sets, where node features are continuous, need some further considerations. In addition, it would be interesting to characterize how close together the learned features lie in a function's image. Then the focus is placed on the case where input node features are from a countable set (that can be a subset of an uncountable set such as $\mathbb{R}^n$).

Lemma 2.1.2. *Assume the input feature space $\mathcal{X}$ is countable. Let $g^{(k)}$ be the function parameterized by a GNN's k-th layer for $k = 1, ..., L$, where $g^{(1)}$ is defined on multisets $X \subset \mathcal{X}$ of bounded size. The range of $g^{(k)}$, i.e., the space of node hidden features $h_v^{(k)}$, is also countable for all $k = 1, ..., L$.*

Here, it is also worth discussing an important benefit of GNNs beyond distinguishing different graphs, that is, capturing similarity of graph structures. Note that node

feature vectors in the WL test are essentially one-hot encodings and thus cannot capture the similarity between subtrees. In contrast, a GNN satisfying the criteria in theorem 2.1.1 generalizes the WL test by learning to embed the subtrees to low-dimensional space. This enables GNNs to not only discriminate different structures, but also to learn to map similar graph structures to similar embeddings and capture dependencies between graph structures. Capturing structural similarity of the node labels is shown to be helpful for generalization particularly when the co-occurrence of subtrees is sparse across different graphs or there are noisy edges and node features (Yanardag and Vishwanathan, 2015).

To model injective multiset functions for the neighbour aggregation, a theory of "deep multisets" was developed, i.e., parameterizing universal multiset functions with neural networks. Lemma 2.1.3 states that sum aggregators can represent injective, in fact, universal functions over multisets.

Lemma 2.1.3. *Assume X is countable. There exists a function $f : \mathcal{X} \to \mathbb{R}^n$ so that $h(X) = \sum_{x \in X} f(x)$ is unique for each multiset $X \subset \mathcal{X}$ of bounded size. Moreover, any multiset function g can be decomposed as $g(X) = \phi(\sum_{x \in X} f(x))$ for some function ϕ.*

An important distinction between deep multisets and sets is that certain popular injective set functions, such as the mean aggregator, are not injective multiset functions. With the mechanism for modeling universal multiset functions in lemma 2.1.3 as a building block, it is possible to conceive aggregation schemes that can represent universal functions over a node and the multiset of its neighbours, and thus will satisfy the injectiveness condition a theorem 2.1.1. The next corollary provides a simple and concrete formulation among many such aggregation schemes.

Corollary 2.1.1. *Assume $\mathcal{X}$ is countable. There exists a function $f : \mathcal{X} \to \mathbb{R}^n$ so that for infinitely many choices of ϵ, including all irrational numbers, $h(c, X) = (1+\epsilon) \cdot f(c) + \sum_{x \in X} f(x)$ is unique for each pair (c, X), where $c \in \mathcal{X}$ and $X \subset \mathcal{X}$ is a multiset of bounded size. Moreover, any function g over such pairs can be decomposed as $g(c, X) = \phi((1+\epsilon) \cdot f(c) + \sum_{x \in X} f(x))$ for some function ϕ.*

We can use multi-layer perceptrons (MLPs) to model and learn f and ϕ in corollary 2.1.1, thanks to the universal approximation theorem (Hornik et al., 1989; Hornik, 1991). In practice, we model $f^{(k+1)} \circ \phi^{(k)}$ with one MLP, because MLPs can represent the composition of functions. In the first iteration, we do not need MLPs before summation if input features are one-hot encodings as their summation alone is injective. We can make ϵ a learnable parameter or a fixed scalar. Then, GIN updates node representations as

$$h_v^{(k)} = \mathrm{MLP}^{(k)}\left(\left(1 + \epsilon^{(k)}\right) \cdot h_v^{(k-1)} + \sum_{u \in \mathcal{N}(\sqsubseteq)} h_u^{(k-1)}\right). \tag{2.1.6.1}$$

Generally, there may exist many other powerful GNNs. GIN is one such example among many maximally powerful GNNs, while being simple.

2.1.7 Gated Graph Neural Networks

This subsection describes Gated Graph Neural Networks (GG-NNs), Li et al. (2015) adaptation of GNNs that is suitable for non-sequential outputs. The biggest modification of GNNs is that a Gated Recurrent Units (Cho et al., 2014) is used and the recurrence is unrolled for a fixed number of steps T and backpropagation through

time is used in order to compute gradients. This requires more memory than the Almeida-Pineda algorithm (Almeida, 1990; Pineda, 1987), but it removes the need to constrain parameters to ensure convergence. Also, the underlying representations and output model are extended.

Propagation Model

The basic recurrence of the propagation model is:

$$\mathbf{h}_v^{(1)} = [\mathbf{x}_v^\mathsf{T}, \mathbf{0}]^\mathsf{T} \tag{2.1.7.1}$$

$$\mathbf{a}_v^{(t)} = \mathbf{A}_{v:}^\mathsf{T}\left[\mathbf{h}_1^{(\text{t-1})\mathsf{T}} ... \mathbf{h}_{|\mathcal{V}|}^{(\text{t-1})\mathsf{T}}\right]^\mathsf{T} + \mathbf{b} \tag{2.1.7.2}$$

$$\mathbf{z}_v^{(t)} = \sigma\left(\mathbf{W}^z\mathbf{a}_v^{(t)} + \mathbf{U}^z\mathbf{h}_v^{(t-1)}\right) \tag{2.1.7.3}$$

$$\mathbf{r}_v^{(t)} = \sigma\left(\mathbf{W}^r\mathbf{a}_v^{(t)} + \mathbf{U}^r\mathbf{h}_v^{(t-1)}\right) \tag{2.1.7.4}$$

$$\mathbf{h}_v^{(t)} = \tanh\left(\mathbf{W}\mathbf{a}_v^{(t)} + \mathbf{U}\left(\mathbf{r}_v^{(t)} \odot \mathbf{h}_\mathbf{v}^{\mathbf{t-1}}\right)\right) \tag{2.1.7.5}$$

$$\mathbf{h}_v^{(t)} = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{(t-1)} + \mathbf{z}_v^t \odot \mathbf{h}_v^{(t)}. \tag{2.1.7.6}$$

The matrix $\mathbf{A} \in \mathbb{R}^{D|\mathcal{V}| \times 2D|\mathcal{V}|}$ determines how nodes in the graph communicate with each other.

There are several types of one-step outputs that could be produced in different situations. First, GG-NNs support *node selection* tasks by making $o_v = g(h_v^{(T)}, x_v)$ for each node $v \in V$ output node scores and applying a softmax over node scores. Second, for graph-level outputs, a graph level representation vector is defined as

$$\mathbf{h}_\mathcal{G} = \tanh\left(\sum_{v \in V} \sigma\left(i(\mathbf{h}_v^{(T)}, x_v)\right) \odot \tanh\left(j(\mathbf{h}_v^{(T)}, x_v)\right)\right), \tag{2.1.7.7}$$

where $\sigma(i(\mathbf{h}_v^{(T)}, x_v))$ acts as a soft attention mechanism that decides which nodes are relevant to the current graph-level task. i and j are neural networks that take the concatenation of $h_v^{(T)}$ and x_v as input and outputs real-valued vectors. The tanh functions can also be replaced with the identity.

Gated Graph Sequential Neural Networks

In gated graph neural networks (GGS-NNs), several graph gated neural networks operate in sequence to produce an output sequence $o^{(1)} ... o^{(K)}$.

For the k^{th} output step, the matrix of node annotations is noted as $\mathcal{X}^{(k)} = [x_1^{(k)}; ...; x_{|\mathcal{V}|}^{(k)}]^\mathsf{T} \in \mathbb{R}^{|\mathcal{V}| \times L_\mathcal{V}}$. Two GG-NNS, $\mathcal{F}_o^{(k)}$ and $\mathcal{F}_\mathcal{X}^{(k)} : \mathcal{F}_o^{(k)}$ are used for predicting $o^{(k)}$ from $\mathcal{X}^{(k)}$, and $\mathcal{F}_\mathcal{X}^{(k)}$ for predicting $\mathcal{X}^{(t+1)}$ from $\mathcal{X}^{(k)}$. $\mathcal{X}^{(t+1)}$ can be seen as the states carried over from step k to $k+1$. $\mathcal{F}_o^{(k)}$ and $\mathcal{F}_\mathcal{X}^{(k)}$ contain a propagation model and an output model. In the propagation models, the matrix of node vectors at the t^{th} propagation step of the k^{th} output step is denoted as $\mathcal{H}^{(k,t)} = [h_1^{(k,t)}; ...; x_{|\mathcal{V}|}^{(k,t)}]^\mathsf{T} \in \mathbb{R}^{|\mathcal{V}| \times D}$. In step k, $\mathcal{H}^{(k,1)}$ is set by 0-extending $\mathcal{X}^{(k)}$ per node. An overview of the model is shown in fig. 2.12. Alternatively, $\mathcal{F}_o^{(k)}$ and $\mathcal{F}_\mathcal{X}^{(k)}$

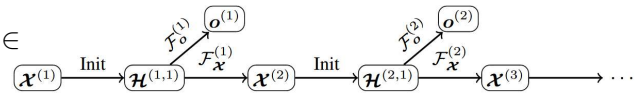


Figure 2.12. Architecture of GGS-NN models.

can share a single propagation model, and just have separate output models. This simpler variant is faster to train and evaluate, and in many cases can achieve similar performance level as the full model. But in cases where the desired propagation behaviour for$\mathcal{F}_{\wr}^{(k)}$ and $\mathcal{F}_{\mathcal{X}}^{(k)}$ are different, this variant may not work as well.

A *node annotation* output model for predicting $\mathcal{X}^{(t+1)}$ from $\mathcal{H}^{(k,T)}$. The prediction is done for each node independently using a neural network $j(\mathbf{h}_v^{(k,T)}, x_v^{(k)})$ that takes the concatenation of $\mathbf{h}_v^{(k,T)}$ and $x_v^{(k)}$ as input and outputs a vector of real-valued scores:

$$x_v^{(k+1)} = \sigma\Big(j(\mathbf{h}_v^{(k,T)}, x_v^{(k)})\Big). \tag{2.1.7.8}$$

There are two settings for training GGS-NNs: specifying all intermediate annotations $\mathcal{X}^{(k)}$, or training the full model end-to-end given only $\mathcal{X}^{(1)}$, graphs and target sequences. The former can improve performance when we have domain knowledge about specific intermediate information that should be represented in the internal state of nodes, while the latter is more general.

2.1.8 Message Passing Neural Networks

This section formulates existing models into a single common framework called Message Passing Neural Networks (MPNNs) introduced by Gilmer et al. (2017). Eight different notable architectures are described using the MPNN framework.

The described MPNNs operate on undirected graphs G with node features x_v and edge features e_{vw}, but it is trivial to extend the formalism to directed multigraphs. The forward pass has two phases, a message passing phase and a readout phase. The message passing phase runs for T time steps and is defined in terms of message functions M_t and vertex update functions U_t. During the message passing phase, hidden states h_v^t at each node in the graph are updated based on messages m_v^{t+1} according to

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw}) \tag{2.1.8.1}$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}) \tag{2.1.8.2}$$

where in the sum, $N(v)$ denotes the neighbors of v in graph G. The readout phase computes a feature vector for the whole graph using some readout function R according to

$$\hat{y} = R(\{h_v^T | v \in G\}). \tag{2.1.8.3}$$

The message functions M_t, vertex update functions U_t, and readout function R are all learned differentiable functions. R operates on the set of node states and must be invariant to permutations of the node states in order for the MPNN to be invariant to graph isomorphism. In what follows, previous models in the literature are defined by specifying the message function M_t, vertex update function U_t, and readout function R used. Note one could also learn edge features in an MPNN by introducing hidden states for all edges in the graph $h_{e_{vw}}^t$ and updating them analogously to eqs. (2.1.8.1) and (2.1.8.2). Kearnes et al. (2016) use this idea.

Convolutional Networks for Learning Molecular Fingerprints (Duvenaud et al., 2015). The message function used is $M(h_v, h_w, e_{vw}) = (h_w, e_{vw})$ where $(.,.)$ denotes concatenation. The vertex update function used is $U_t(h_v^t, m_v^{t+1}) =$

$\sigma(H_t^{deg(v)} m_v^{t+1})$, where σ is the sigmoid function, $\deg(v)$ is the degree of vertex v and H_t^N is a learned matrix for each time step t and vertex degree N. R has skip connections to all previous hidden states h_v^t and is equal to $f\Big(\sum_{v,t} \text{softmax}(W_t h_v^t)\Big)$, where f is a neural network and W_t are learned readout matrices, one for each time step t. This message passing scheme may be problematic since the resulting message vector is $m_v^{t+1} = (\sum_{v,t} h_w^t, \sum e_{vw})$, which separately sums over connected nodes and connected edges. It follows that the message passing implemented in Duvenaud et al. (2015) is unable to identify correlations between edge states and node states.

Gated Graph Neural Networks (GG-NN) (Li et al., 2015)). The message function used is $M_t(h_v^t, h_w^t, e_{vw}) = A_{e_{vw}} h_w^t$, where $A_{e_{vw}}$ is a learned matrix, one for each edge label e (the model assumes discrete edge types). The update function is $U_t = \text{GRU}(h_v^t, m_v^{t+1})$, where GRU is the Gated Recurrent Unit introduced in Cho et al. (2014). This work used weight tying, so the same update function is used at each time step t. Finally,

$$R = \sum_{v \in V} \sigma\Big(i(h_v^T, h_v^0)\Big) \odot \Big(j(h_v^T)\Big) \tag{2.1.8.4}$$

where i and j are neural networks, and $\odot$ denotes element-wise multiplication.

Interaction Networks (Battaglia et al., 2016). This work considered both the case where there is a target at each node in the graph, and where there is a graph level target. It also considered the case where there are node level effects applied at each time step, in such a case the update function takes as input the concatenation (h_v, x_v, m_v) where x_v is an external vector representing some outside influence on the vertex v. The message function $M(h_v, h_w, e_{vw})$ is a neural network which takes the concatenation (h_v, h_w, e_{vw}). The vertex update function $U(h_v, x_v, m_v)$ is a neural network which takes as input the concatenation (h_v, x_v, m_v). Finally, in the case where there is a graph level output, $R = f(\sum_{v \in G} h_v^T)$ where f is a neural network which takes the sum of the final hidden states h_v^T . Note the original work only defined the model for $T = 1$.

Molecular Graph Convolutions (Kearnes et al., 2016). This work deviates slightly from other MPNNs in that it introduces edge representations e_{vw}^t which are updated during the message passing phase. The message function used for node messages is $M(h_v^t, h_w^t, e_{vw}^t) = e_{vw}^t$. The vertex update function is $U_t(h_v^t, m_v^{t+1}) = \alpha(W_1(\alpha(W_0 h_v^t), m_v^{t+1}))$ where $(.,.)$ denotes concatenation, α is the ReLU activation and W_1, W_0 are learned weight matrices. The edge state update is defined by $e_{vw}^{t+1} = U_t'(e_{vw}^t, h_v^t, h_w^t) = \alpha(W_4(\alpha(W_2, e_{vw}^t), \alpha(W_3(h_v^t, h_w^t))))$ where the W_i are also learned weight matrices.

Deep Tensor Neural Networks (Schütt et al., 2017). The message from w to v is computed by

$$M_t = \tanh(W^{fc}((W^{fc} h_w^t + b_1) \odot (W^{df} e_{ew} + b_2))) \tag{2.1.8.5}$$

where W^{fc}, W^{cf}, W^{df} are matrices and b_1, b_2 are bias vectors. The update function used is $U_t(h_v^t, m_v^{t+1}) = h_v^t + m_v^{t+1}$. The readout function passes each node independently through a single hidden layer neural network and sums the outputs, in particular

$$R = \sum_v \text{NN}(h_v^T). \tag{2.1.8.6}$$

Laplacian Based Methods (Bruna et al., 2013; Defferrard et al., 2016; Kipf and Welling, 2017). These methods generalize the notion of the convolution operation typically applied to image datasets to an operation that operates on an arbitrary graph G with a real valued adjacency matrix A. The operations defined in Bruna et al. (2013); Defferrard et al. (2016) result in message functions of the form $M_t(h_v^t, h_w^t) = C_{vw}^t h_w^t$, where the matrices C_{vw}^t are parameterized by the eigenvectors of the graph laplacian L, and the learned parameters of the model. The vertex update function used is $U_t(h_v^t, m_v^{t+1}) = \sigma(m_v^{t+1})$ where σ is some pointwise non-linearity (such as ReLU).

The Kipf and Welling (2017) model results in a message function $M_t(h_v^t, h_w^t) = c_{vw}^t h_w^t$ where $c_{vw}^t = (\deg(v)\deg(w))^{-1/2} A_{vw}$. The vertex update function is $U_t(h_v^t, m_v^{t+1}) = \text{ReLU}(W^t m_v^{t+1})$. Exact expressions for the C_{vw}^t and the derivation of the reformulation of these models as MPNNs, in the supplementary material in Gilmer et al. (2017).

Gated Message Passing Neural Network. Gilmer et al. (2017) introduce a MPNN variant of the GG-NN model.

Matrix Multiplication: The message function used is the same as in GG-NN (Li et al., 2015) which is defined by the equation $M(h_v, h_w, e_{vw}) = A_{e_{vw}} h_w$.

Edge Network: To allow vector valued edge features the following message function is proposed: $M(h_v, h_w, e_{vw}) = A e_{vw} h_w$, where $A(e_{vw})$ is a neural network which maps the edge vector e_{vw} to a $d \times d$ matrix.

Pair Message: One property that the matrix multiplication rule has is that the message from node w to node v is a function only of the hidden state h_w and the edge e_{vw}. In particular, it does not depend on the hidden state h_v^t. In theory, a network may be able to use the message channel more efficiently if the node messages are allowed to depend on both the source and destination node. Thus also the variant on the message function as described in Battaglia et al. (2016) was used. Here the message from w to v along edge e is $mwv = f(h_w^t, h_v^t, e_{vw})$ where f is a neural network. When we apply the above message functions to directed graphs, there are two separate functions used, M^{in} and an M^{out}. Which function is applied to a particular edge e_{vw} depends on the direction of that edge.

Virtual Graph Elements. Two different ways to pass messages throughout the model are proposed, the first involves adding a separate "virtual" edge for non-connected node pairs. The second consists in adding a latent "master" node connected to every input node with a special edge type.

Readout functions. Two readout functions are proposed, first the one used in GG-NN (Li et al., 2015) and defined in eq. (2.1.8.4). The second is a set2set model from Vinyals et al. (2015). The set2set model is specifically designed to operate on sets and should have more expressive power than simply summing the final node states. This model first applies a linear projection to each tuple (h_v^T, x_v) and then takes as input the set of projected tuples $T = (h_v^T, x_v)$. Then, after M steps of computation, the set2set model produces a graph level embedding q_t^* which is invariant to the order of the of the tuples T . This embedding q_t^* is fed through a neural network to produce the output.

Multiple Towers. To address the MPNNs scalability issue, the d dimensional node embeddings h_v^t are splitted into k different d/k dimensional embeddings $h_v^{t,k}$ and run

a propagation step on each of the k copies separately to get temporary embeddings $\{\tilde{h}_v^{t+1,k}, v \in G\}$, using separate message and update functions for each copy. The k temporary embeddings of each node are then mixed together according to the equation

$$(h_v^{t,1}, h_v^{t,2}, ..., h_v^{t,k}) = g(\tilde{h}_v^{t,1}, \tilde{h}_v^{t,2}, ..., \tilde{h}_v^{t,k}) \tag{2.1.8.7}$$

where g denotes a neural network and $(x, y, ...)$ denotes concatenation, with g shared across all nodes in the graph. This mixing preserves the invariance to permutations of the nodes, while allowing the different copies of the graph to communicate with each other during the propagation phase. This can be advantageous in that it allows larger hidden states for the same number of parameters, which yields a computational speedup in practice.

2.1.9 General Powerful Scalable Graph Transformer

This subsection introduces the (general powerful scalable) GPS layer presented by Rampášek et al. (2022), which is a hybrid message passing neural network (MPNN) + Transformer layer. The layer update equations can be instantiated with a variety of MPNN and Transformer layers. Different surveys investigated graph transformers (Shehzad et al., 2024; Yuan et al., 2025).

Preventing early smoothing. Architectures like GraphTrans (Wu et al., 2021b) comprising of a few layers of MPNNs before the Transformer are limited by problems of MPNNs' over-smoothing, over-squashing, and low expressivity against the WL test (Alon and Yahav, 2020; Topping et al., 2021), these layers could irreparably fail to keep some information in the early stage. Although they could make use of PE/SE or more expressive MPNNs (Beaini et al., 2021; Dwivedi et al., 2021), they are still likely to lose information. An analogous 2-stage strategy was successful in computer vision (d'Ascoli et al., 2021; Guo et al., 2022) thanks to the high expressivity of convolutional layers on grids, but it is not expected to achieve the same success on graphs due to the limitations of message-passing.

Update function. At each layer, the features are updated by aggregating the output of an MPNN layer with that of a global attention layer and described by the equations below. Note that the edge features are only passed to the MPNN layer, and that residual connections with batch normalization (Ioffe, 2015) are omitted for clarity. Both the MPNN and GlobalAttn layers are modular, i.e., MPNN can be any function that acts on a local neighbourhood and GlobalAttn can be any fully-connected layer.

$$\mathbf{X}^{\ell+1}, \mathbf{E}^{\ell+1} = \mathrm{GPS}^{\ell}(\mathbf{X}^{\ell}, \mathbf{E}^{\ell}, \mathbf{A}) \tag{2.1.9.1}$$

$$\text{computed as} \quad \mathbf{X}_M^{\ell+1}, \mathbf{E}^{\ell+1} = \mathrm{MPNN}_e^{\ell}(\mathbf{X}^{\ell}, \mathbf{E}^{\ell}, \mathbf{A}), \tag{2.1.9.2}$$

$$\mathbf{X}_T^{\ell+1} = \mathrm{GlobalAttn}^{\ell}(\mathbf{X}^{\ell}), \tag{2.1.9.3}$$

$$\mathbf{X}^{\ell+1} = \mathrm{MLP}^{\ell}(\mathbf{X}_M^{\ell+1} + \mathbf{X}_T^{\ell+1}), \tag{2.1.9.4}$$

where $\mathbf{A} \in \mathbb{R}^{N \times N}$ is the adjacency matrix of a graph with N nodes and E edges; $X^{\ell} \in \mathbb{R}^{N \times d_\ell}, \mathbf{E}^{\ell} \in \mathbb{R}^{E \times d_\ell}$ are the d_ℓ-dimensional node and edge features, respectively; $\texttt{MPNN}_e^{\ell}$ and $\texttt{GlobalAttn}^{\ell}$ are instances of an MPNN with edge features and of a global attention mechanism at the ℓ-th layer with their corresponding learnable parameters, respectively; `MLP` is a 2-layer MLP block.

Modularity is achieved by allowing drop-in replacement for a number of module choices, including the initial PE/SE types, the networks that processes those PE/SE, the MPNN and global attention layers that constitute a GPS layer, and the final task-specific prediction head. Further, as research advances in different directions, GRAPHGPS allows to easily implement new PE/SE and other layers.

Scalability is achieved by allowing for a computational complexity linear in both the number of nodes and edges $O(N + E)$; excluding the potential precomputation step required for various PE, such as Laplacian eigen-decomposition. By restricting the PE/SE to real nodes and edges, and by excluding the edge features from the global attention layer, it is possible to avoid materializing the full quadratic attention matrix. Therefore it is possible to utilize a linear Transformer with $O(N)$ complexity, while the complexity of an MPNN is $O(E)$. For sparse graphs such as molecular graphs, regular graphs, and knowledge graphs, the edges are practically proportional to the nodes $E = \Theta(N)$, meaning the entire complexity can be considered linear in the number of nodes $O(N)$.

Expressivity in terms of sub-structure identification and the Weisfeiler-Leman (WL) test is achieved via providing a rich set of PE/SE, as proposed in various works (Beaini et al., 2021; Kreuzer et al., 2021; Dwivedi et al., 2021; Bodnar et al., 2021; Bouritsas et al., 2022). Further, the Transformer allows to resolve the expressivity bottlenecks caused by over-smoothing (Kreuzer et al., 2021) and over-squashing (Alon and Yahav, 2020) by allowing information to spread across the graph via full-connectivity. Finally, given the right components, the proposed architecture does not lose edge information and is a universal function approximator on graphs.

2.1.10 Principal Neighbourhood Aggregation

This subsection presents Principal Neighbourhood Aggregation (PNA), an architecture combining multiple aggregators with degree-scalers (which generalize the sum aggregator) introduced by Corso et al. (2020).

Aggregators

The following paragraphs will describe the aggregators leveraged in the PNA architecture.

Mean Aggregation $\mu(X^l)$. The most common message aggregator in the literature, wherein each node computes a weighted average or sum of its incoming messages. Equation (2.1.10.1) presents, on the left, the general mean equation, and, on the right, the direct neighbour formulation, where X is any multiset, X^l are the nodes' features at layer$l$, $N(i)$ is the neighbourhood of node i and $d_i = |N(i)|$. For clarity $\mathbb{E}[f(X)]$ is used where X is a multiset of size d to be defined as $\mathbb{E}[f(X)] = \frac{1}{d}_{x \in X} f(x)$.

$$\mu(X) = \mathbb{E}[X] \quad , \quad \mu_i(X^l) = \frac{1}{d_i} \sum_{j \in N(i)} X_j^l \tag{2.1.10.1}$$

Maximum and minimum aggregations $max(X^l)$, $min(X^l)$. Also often used in literature, they are very useful for discrete tasks, for domains where credit assignment is important and when extrapolating to unseen distributions of graphs

(Veličković et al., 2019). Alternatively, other aggregators, i.e. the softmax and softmin aggregators, which are differentiable and work for weighted graphs, do not perform as well on benchmarks.

$$\max_i(X^l) = \max_{j \in N(i)} X_j^l \quad , \quad \min_i(X^l) = \min_{j \in N(i)} X_j^l \tag{2.1.10.2}$$

Standard deviation aggregation $\sigma(X^l)$. The standard deviation (STD or σ) is used to quantify the spread of neighbouring nodes features, such that a node can assess the diversity of the signals it receives. Equation (2.1.10.3) presents, on the left, the standard deviation formulation and, on the right, the STD of a graph-neighbourhood. ReLU is the rectified linear unit used to avoid negative values caused by numerical errors and ϵ is a small positive number to ensure σ is differentiable.

$$\sigma(X) = \sqrt{\mathbb{E}[X^2] - \mathbb{E}[X]^2} \; , \; \sigma_i(X^l) = \sqrt{ReLU\Big(\mu_i(X^{l^2}) - \mu_i(X^l)^2\Big) + \epsilon} \tag{2.1.10.3}$$

Normalized moments aggregation $M_n(X^l)$ The mean and standard deviation are the first and second normalized moments of the multiset ($n = 1, n = 2$). Additional moments, such as the skewness ($n = 3$), the kurtosis ($n = 4$), or higher moments, could be useful to better describe the neighbourhood. These become even more important when the degree of a node is high because four aggregators are insufficient to describe the neighbourhood accurately. The n^{th} root normalization is chosen, as presented in eq. (2.1.10.4), because it gives a statistic that scales linearly with the size of the individual elements (as the other aggregators); this gives the training adequate numerical stability. Once again an ϵ is added to the absolute value of the expectation before applying the n^{th} root for numerical stability of the gradient.

$$M_n(X) = \sqrt[n]{\mathbb{E}[(X - \mu)^n]} \quad , \quad n > 1 \tag{2.1.10.4}$$

Degree-based scalers. Scalers are introduced as functions of the number of messages being aggregated (usually the node degree), which are multiplied with the aggregated value to perform either an amplification or an attenuation of the incoming messages. The logarithmic scaler S_{amp} is presented in eq. (2.1.10.5), where δ is a normalization parameter computed over the training set, and d is the degree of the node receiving the message.

$$S_{amp}(d) = \frac{log(d+1)}{\delta} \quad , \quad d = \frac{1}{|\text{train}|} \sum_{i \in \text{train}} \log(d_i + 1) \tag{2.1.10.5}$$

This scaler is further generalized in eq. (2.1.10.6), where α is a variable parameter that is negative for attenuation, positive for amplification or zero for no scaling. Other definitions of $S(d)$ can be used, such as a linear scaling, as long as the function is injective for $d > 0$.

$$S(d, \alpha) = \left(\frac{log(d+1)}{\delta}\right)^{\alpha}, \quad d > 0, \quad -1 \leq \alpha \leq 1 \tag{2.1.10.6}$$

Combined Aggregation

The aggregators and scalers presented in previous paragraphs are combined to obtain the Principal Neighbourhood Aggregation (PNA). This is a general and flexible architecture, which in our tests we used with four neighbour-aggregations with three degree-scalers each, as summarized in Equation 7.

The aggregators are defined in eqs. (2.1.10.1) and (2.1.10.2) and **??**, while the scalers are defined in eq. (2.1.10.6), with $\otimes$ being the tensor product.

$$\bigoplus = \underbrace{\begin{bmatrix} I \\ S(D, \alpha = 1) \\ S(D, \alpha = -1) \end{bmatrix}}_{scalers} \otimes \underbrace{\begin{bmatrix} \mu \\ \sigma \\ \max \\ \min \end{bmatrix}}_{aggregators} \tag{2.1.10.7}$$

Higher degree graphs such as social networks could benefit from further aggregators (e.g. using the moments proposed in eq. (2.1.10.4)). The PNA operator is inserted within the framework of a message passing neural network (Gilmer et al., 2017), obtaining the following GNN layer:

$$X_i^{(t+1)} = U\left(X_i^{(t)}, \bigoplus_{(j,i)\in E} M\left(X_i^{(t)}, E_{j\to i}, X_j^{(t)}\right)\right) \tag{2.1.10.8}$$

where $E_{j\to i}$ is the feature (if present) of the edge (j, i), M and U are neural networks. U reduces the size of the concatenated message (in space $\mathbb{R}^{13F}$) back to $\mathbb{R}^F$ where F is the dimension of the hidden features in the network. Similarly to the MPNN paper (Gilmer et al., 2017), multiple towers are employed to improve computational complexity and generalization performance.

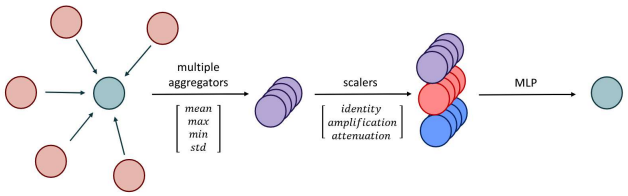


Figure 2.13. Diagram for the Principal Neighbourhood Aggregation or PNA.

Using twelve operations per kernel will require the usage of additional weights per input feature in the U function, which could seem to be just quantitatively, not qualitatively, more powerful than an ordinary MPNN with a single aggregator (Gilmer et al., 2017). However, the overall increase in parameters in the GNN model is modest and, as per the theoretical analysis presented in Corso et al. (2020), a limiting factor of GNNs is likely their usage of a single aggregation.

This is comparable to convolutional neural networks (CNN) where a simple 3×3 convolutional kernel requires 9 weights per feature (1 weight per neighbour). Using a CNN with a single weight per 3×3 kernel will reduce the computational capacity since the feedforward network won't be able to compute derivatives or the Laplacian operator. Hence, it is intuitive that the GNNs should also require multiple weights per node.

2.1.11 Diffusion Models

Multiple surveys (Zhang et al., 2023; Yang et al., 2023a; Shou et al., 2026) classified and presented graph diffusion models.

Diffusion models are a class of generative models that gradually introduce noise into data until it conforms to a prior distribution (Ho et al., 2020). The models then learn to reverse this process to generate viable samples. By leveraging the power of diffusion models, researchers can create generative models that can accurately capture the underlying structure of complex datasets and produce high-quality, realistic samples. There are three sub-types: denoising diffusion probabilistic models (DDPMs) (Vignac et al., 2022; Haefeli et al., 2022; Anand and Achim, 2022; Trippe et al., 2022; Luo et al., 2022), scorebased generative models (SGMs) (Niu et al., 2020; Chen et al., 2022a), and stochastic differential equations (SDEs) (Huang et al., 2022a; Jo et al., 2022; Luo et al., 2023), which differ in how they implement the forward and backward diffusion pass. Figure 2.14 presents a generic framework that is based on three categories.

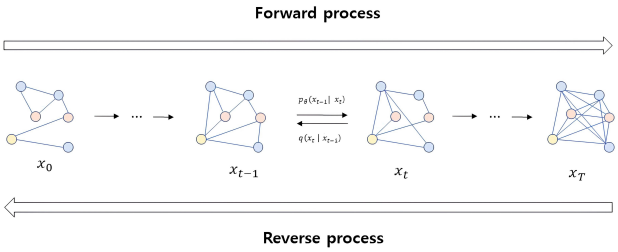


Figure 2.14. Illustration of diffusion models generic framework for deep graph generation.

Denoising Diffusion Probabilistic Models (DDPM).

A denoising diffusion probabilistic model (DDPM) is a generative model that can generate novel data samples from a given data distribution that employs two Markov chains (Ho et al., 2020).

Forward process. First, the original data is transformed into a simpler prior distribution by gradually adding noise over a fixed number of diffusion steps T according to a variance schedule β. This forward pass aims to transform the input data into a distribution that can be easily sampled using standard techniques such as sampling from a Gaussian distribution. Given a data distribution $\mathbf{x}_0 \sim q(\mathbf{x}_0)$, the forward process generates $\mathbf{x}_T$ with transition kernel $q(x_t|x_{t-1})$ and can be defined as follows (Ho et al., 2020):

$$q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1}) \tag{2.1.11.1}$$

$$q(x_t|x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \tag{2.1.11.2}$$

where β_t is a hyper-parameter. To simplify the discussion, it is possible to focus on the use of Gaussian noise as the transition kernels denoted as $\mathcal{N}$ in eq. (2.1.11.2). When $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=0}^{t} \alpha_s$:

$$q(x_t|x_0) := \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I) \tag{2.1.11.3}$$

Reverse process. To generate new samples from the data distribution, the reverse process is performed. The reverse pass makes use of a neural network trained to predict the noise that was added at each step in the forward pass. This pass removes the noise at each step in reverse order until the original data is recovered. To generate $p_\theta(x_0)$ that follows the true data distribution $q(x_0)$, start with $p_\theta(T)$ and optimize the model with the following objective, as described in Ho et al. (2020):

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x_t}) := \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(x_t, t), \sigma_t^2\mathbf{I}) \tag{2.1.11.4}$$

As stated in Yang et al. (2023a), the model's optimization objective can be expressed as:

$$E_{t\sim\mathcal{U}(1,T),\mathbf{x}_0\sim q(\mathbf{x}_0),\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\lambda(t)||\epsilon - \epsilon_\theta(\mathbf{x}_t, t)||^2 \tag{2.1.11.5}$$

Vignac et al. (2022) and Haefeli et al. (2022) adopt DDPM technique for graph diffusion though they add discrete noise instead of continuous Gaussian.

Score-based generative models (SGMs).

Score-based generative models are a class of probabilistic models that employ a score function, also referred to as an energy function or an unnormalized probability density function, to illustrate the probability distribution of the data (Song and Ermon, 2020; Song et al., 2020a).

The score function of a probability density function $p(x)$ is defined as the gradient of the logarithm of the probability density, $\nabla_x \log p(x)$. A noise-conditional score network is a type of deep neural network that is trained to estimate the score function $\nabla_{x_t} \log q(x_t)$, which represents the gradient of the logarithm of the conditional probability density function $q(x_t)$ of the data given the noise. The network is denoted as $s_\theta(x, t)$, where θ represents the learnable parameters of the network and x means the input data at time t. The network is generally trained using maximum likelihood estimation or a variant thereof, such as noise-contrastive estimation or denoising score matching.

Employing denoising score matching and similar notations, as represented in eq. (2.1.11.5), the training objective for the score network is given by:

$$E_{t\sim\mathcal{U}(1,T),\mathbf{x}_0\sim q(\mathbf{x}_0),\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I})}[\lambda(t)||\epsilon + \sigma_t s_0(\mathbf{x}_t, t)||^2] \tag{2.1.11.6}$$

The pioneer score-based graph generation model, EDP-GNN (Niu et al., 2020), leverages GIN (Xu et al., 2018) and makes use of annealed Langevin dynamic sampling to diffuse the edges. In contrast, NVDiff (Chen et al., 2022a), utilizes an attention-based score generative model to exclusively diffuse the latent representation of node vectors.

Stochastic Differential Equations (SDEs).

Stochastic Differential Equations (SDEs) are a class of mathematical models that characterize the development of a system over time under the effect of random noise. A Score-based Stochastic Differential Equation (Score SDE) is a type of SDE where the drift term is defined as the negative gradient of a score function, and the diffusion term is a function of time (Song et al., 2020b).

Forward SDE. In mathematics, a forward SDE is a continuous-time dynamical system that describes the evolution of a state variable over time, where both deterministic and stochastic forces govern the evolution. The forward SDE can be defined as follows (Song et al., 2020b):

$$dx = f(x,t)dt + g(t)dw \tag{2.1.11.7}$$

Reverse SDE. A reverse stochastic differential equation (SDE) is a continuous-time dynamical system that progresses backward. It is generally employed to calculate the score function for the forward SDE, which can subsequently be operated to produce samples from the conditional distribution. The reverse SDE can be defined as follows (Song et al., 2020b):

$$dx = [f(x,t) - g(t)^2 \nabla_x \log p_t(x)]dt + g(t)d\bar{w} \tag{2.1.11.8}$$

The score function is estimated by parameterizing a score model $s_\theta(xt,t)$, which involves generalizing the score matching objective in eq. (2.1.11.9) to continuous time as follows (Yang et al., 2023c):

$$E_{t\sim\mathcal{U}(0,T),\mathbf{x}_t\sim q(\mathbf{x}_t|\mathbf{x}_0)}[\lambda(t)||s_0(\mathbf{x}_t,t) - \nabla_{x_t} \log q_{0_t}(\mathbf{x}_t)|\mathbf{x}_0)||^2] \tag{2.1.11.9}$$

GraphGDP (Huang et al., 2022a), diffuses the graph edges via stochastic differential equations (SDE). For the sampling process tailored to graph data, the authors designed a position-enhanced graph score network employing the structure and position information of graphs and the noisy adjacency matrices. GDSS (Jo et al., 2022) formulated a system of two stochastic differential equations, one for each graph features generation and adjacency matrix generation in parallel for diffusing both nodes and edges. Another SDE based modelGSDM (Luo et al., 2023), addressing the problem of Gaussian noise transforming the adjacency matrix into a dense one quickly, restricts the insertion of Gaussian noise to the eigenvalue matrix of the adjacency matrix

Graph Diffusion Models

Multiple works (You et al., 2018; Simonovsky and Komodakis, 2018; De Cao and Kipf, 2018; Bojchevski et al., 2018; Liu et al., 2018; Kipf and Welling, 2016; Li et al., 2018; Liao et al., 2019; Dai et al., 2020; Chen et al., 2021) have demonstrated various graph generation qualities depending on the sequence of node generation. Autoregressive models, in particular, perform optimally (Huang et al., 2022a). However, they are ineffective in capturing the permutation invariant properties. Diffusion models have come to rescue here, the first score-based model for permutation invariant graph generation, EDP-GNN (Niu et al., 2020), adapts GIN (Xu et al., 2018) and employs annealed Langevin dynamic sampling. It assumes undirected graphs, adding Gaussian noise only to the upper triangular part of the adjacency matrix for edge diffusion, and quantizes the continuous adjacency matrix generated when sampling. On predicting edgewise features, EDP-GNNs are empirically shown to be more expressive compared with vanilla ones. Additionally, sample quality is found to be comparable with state-of-the-art models.

GraphGDP (Huang et al., 2022a) also attempted to address the generation of permutation invariant graphs, where state-of-the-art autoregressive models failed. However, the proposed continuous-time generative diffusion model employed stochastic differential equations (SDE) for edge diffusion. Moreover, for sampling, a score network

is specifically designed utilizing the structure and position information. Although diffusion models show great promise, the unique intrinsic properties of graph data call for a few adaptations in the standard diffusion process to make them fit for graph generation tasks. The following subsection details the modifications thus called for.

Discrete Diffusion

It is observed that the standard diffusion models have difficulties in capturing the structural properties of the graph data because it relies on a continuous Gaussian noise process which creates fully connected, noisy graphs where structural information is not defined (Ingraham et al., 2019). DiGress (Vignac et al., 2022), the first discrete graph generation model, features a Markov process noise model where, similar to the way noise is infused independently in each pixel in image diffusion, all progressive noise addition steps are performed independently on each node or edge. The denoising process involves training a graph transformer network to predict the clean graph from a noisy input. It delivers state-ofthe-art performance on both molecular and non-molecular datasets. Meanwhile, another concurrent work (Haefeli et al., 2022) validates the improvement in the sampling quality and denoising efficiency with the use of discrete noise.

Furthermore, a work (Anand and Achim, 2022) in the protein design domain modifies the diffusion process by modeling it as random masking of a portion of the residues. This fraction is decided by linear interpolation between 0 and 1 depending on the time step t. For sampling, the same procedure is reversed, starting with the completely masked residues at time T. These works use the Markov transition probability matrix to implement discrete noise, and the objective becomes a simple cross-entropy loss.

Low Rank Diffusion

For a typical image diffusion model, data is corrupted by adding full-rank Gaussian noise (Ingraham et al., 2019; Dhariwal and Nichol, 2021). However, diffusing graphs with full-rank noise is detrimental to the training of the score network. In the image diffusion model tasks, the early- and mid-diffusion process stages are not influenced by the inclusion of full-rank Gaussian noise in the image recognition (Song and Ermon, 2020; Song et al., 2020b; Dhariwal and Nichol, 2021). Nevertheless, when perturbing the graph with full-rank Gaussian noise, the fatal noise is injected into the unsupported regions and is detrimental to the training of the score network (Song and Ermon, 2020; Alon and Yahav, 2020). It compromises graphic structure and feature representation. To resolve this issue, rather than doing a full-rank diffusion of the adjacency matrix, GSDM (Luo et al., 2023) restricts the insertion of Gaussian noise to the eigenvalue matrix, which is the spectral decomposition of the adjacency matrix. This approach is effective in both generic graph and molecule generation tasks.

Roto-Translation Equivariance and Invariance

With regards to molecular graph generation specifically, the distribution of molecules should be invariant to rotation and translation, that is, no matter what Euclidean transition is applied to the molecules, their probability distribution should not change (Duvenaud et al., 2015; Jørgensen et al., 2018). Thus, generative models need to make this property true when generating 3D molecular data. The most commonly used method to satisfy these invariances is to employ a roto-translation equivariant

score network. Xu et al. (2022) proved that an invariant density can be produced by dynamics along an equivariant Gaussian Markov kernel beginning from an invariant standard density. This means that it is possible to ensure the invariance property by making neural networks equivariant. Consequently, in GeoDiff (Xu et al., 2022), an equivariant graph convolution network termed as graph field network (GNF) is designed. Inspired by the aforementioned principle, several works (Trippe et al., 2022; Lin et al., 2025; Igashov et al., 2024; Hoogeboom et al., 2022) utilize either equivariant graph neural network (Satorras et al., 2021) or its modification as their score network. Another trick, is that, zero center of mass (CoM) is also explored for this purpose, as it moves the center of atomic coordinates to zero, ensuring translation invariance. Another approach, ConfGF (Shi et al., 2021), formulates the problem as diffusing interatomic distances instead of atomic coordinates. Based on the observation that those two gradient fields are connected via chain rule, their score networks are roto-translation invariant.

2.1.12 Normalizations Layers

The general GNN structure equipped with a normalization layer can be represented as:

$$H^{(k)=F^{(k)}\left(\mathrm{Norm}\left(W^{(k)}H^{(k-1)}Q\right)\right)}, \tag{2.1.12.1}$$

where $F^{(k)}$ is a function that applies to each node separately, Q is an $n \times n$ matrix representing the neighbor aggregation, and $W^{(k)}$ is the weight/parameter matrix in layer k.

The concrete operations of the adaptations of the normalization methods is then described. Consider a batch of graphs $\{G_1, \text{ůůů}, G_b\}$ where b is the batch size. Let n_g be the number of nodes in graph G_g. We generally denote $\hat{h}_{i,j,g}$ as the inputs to the normalization module, e.g., the j-th feature value of node v_i of graph G_g, $i = 1, \text{ůůů}, n_g, j = 1, \text{ůůů}, d, g = 1, \text{ůůů}, b$. The adaptations take the general form:

$$\mathrm{Norm}(\hat{h}_{i,j,g}) = \gamma \cdot \frac{\hat{h}_{i,j,g} - \mu}{\sigma} + \beta \tag{2.1.12.2}$$

where the scopes of mean μ, standard deviation σ, and affine parameters γ, β differ for different normalization methods.

For BatchNorm, normalization and the computation of μ and σ are applied to all values in the same feature dimension across the nodes of all graphs in the batch as in Xu et al. (2018), i.e., over dimensions g, i of $\hat{h}_{i,j,g}$. To adapt LayerNorm to GNNs, we view each node as a basic component, resembling words in a sentence, and apply normalization to all feature values across different dimensions of each node, i.e., over dimension j of $\hat{h}_{i,j,g}$. For InstanceNorm, each graph is regarded as an instance. The normalization is then applied to the feature values across all nodes for each individual graph, i.e., over dimension i of $\hat{h}_{i,j,g}$.

GraphNorm Cai et al. (2021) The current normalization method is modified with a *learnable parameter* to automatically control how much the mean to preserve in the shift operation. Combined with the graph-wise normalization, the method is named Graph Normalization, i.e., GraphNorm. For each graph G, we generally denote value $\hat{h}_{i,j}$ as the inputs to GraphNorm, e.g., the j-th feature value of node

$v_i, i = 1, \text{ůůů}, n, j = 1, \text{ůůů}, d$. GraphNorm takes the following form:

$$\text{GraphNorm}(\hat{h}_{i,j}) = \gamma_j \cdot \frac{\hat{h}_{i,j} - \alpha_j \cdot \mu_j}{\hat{\sigma}_j} + \beta_j \tag{2.1.12.3}$$

where $\mu_j = \frac{\sum_{i=1}^{n} \hat{h}_{i,j}}{n}$, $\hat{\sigma}_j^2 = \frac{\sum_{i=1}^{n} (\hat{h}_{i,j} - \alpha_j \cdot \mu_j)^2}{n}$, and γ_j, β_j are the affine parameters as in other normalization methods. By introducing the learnable parameter α_j for each feature dimension j, we are able to learn how much the information we need to keep in the mean. GraphNorm has stronger expressive power than InstanceNorm (Cai et al., 2021).

NodeNorm Node normalization aims to mitigate oversmoothing by normalizing each node's representation independently across feature dimensions. Given node features $\hat{h}i, j$, NodeNorm computes the ℓ_2 norm of each node embedding, $|\hat{h}i|_2 = \sqrt{\sum_{j=1}^{d} \hat{h}_{i,j}^2}$, and rescales the node features as:

$$\text{NodeNorm}(\hat{h}_{i,j}) = s \cdot \frac{\hat{h}_{i,j}}{|\hat{h}_i|_2}, \tag{2.1.12.4}$$

where s is a learnable or fixed scaling factor. Unlike LayerNorm, which normalizes using per-node mean and variance, NodeNorm standardizes only by the node norm, preserving relative feature proportions while preventing the explosion or collapse of node embeddings during message passing.

2.1.13 Explainability

Considering a supervised task $\mathcal{T}$ with the aim of learning a mapping from $\mathcal{X}$ to $\mathcal{Y}$, and a model $\mathcal{M}$ trained for this task. Given a set of $(\mathbf{x}, y)$ pairs $\subseteq (\mathcal{X}, \mathcal{Y})$ and the model $\mathcal{M}$, generate an explanation $\mathbf{e}$ from a given set $\mathcal{D}_E$ such that $\mathbf{e}$ "explains" the prediction $\hat{y} = \mathcal{M}(\mathbf{x})$.

These explanations can be either local to a single test input $(\mathbf{x}, y)$ or global when they explain prediction over a specific dataset $\mathcal{D}' \subseteq (\mathcal{X}, \mathcal{Y})$. Further, the explanation can be generated either *post-hoc* (i.e., after the model training) or *ante-hoc* where the model itself is *self-interpretable*, i.e., it explains its predictions. With some exceptions, post-hoc explanations usually consider a black-box access to the model while self-interpretable methods update the model architecture and/or training itself. It is possible to further differentiate the explanation methods based on their content, i.e., the explanation set $\mathcal{D}_E$. *Local explanations* only consider the local neighborhood of the given data instance while *global explanations* are concerned about the model's overall behavior and thus, searches for patterns in the model's predictions. On the other hand, explanations can also be *counterfactual*, where the aim is to explain a prediction by providing a contrasting example that changes it.

Counterfactual Explainability

Multiple surveys (Artelt and Hammer, 2019; Guidotti, 2024; Guo et al., 2025; Verma et al., 2024b; Linardatos et al., 2020) defined counterfactual explainability for machine learning models, and in particular Prado-Romero et al. (2023d) has been the first work to provide a uniform graph counterfactual explainability definition.

In rest of the subsection, the notion of counterfactual explanation for machine learning classification is formalized. In agreement with Molnar (2020), it is stated that a counterfactual explanation for a prediction highlights the smallest change to the feature values that changes the prediction to a predefined output. Formal definitions are given in the following paragraphs.

Counterfactual explanation definition. Given a classifier b that outputs the decision $y = b(x)$ for an instance x, a counterfactual explanation consists of an instance x' such that the decision for b on x' is different from y, i.e., $b(x) = y$, and such that the difference between x and x' is minimal.

The classifier b is typically a black-box, i.e., a not interpretable machine learning model such as a neural network, an ensemble, etc. Instances x and x' consist of a set $\{x_1, x_2, ..., x_m\}$ of m attribute-value pairs $x_i = (a_i, v_i)$, where a_i is a feature (or attribute) and v_i is a value from the domain of a_i. Note that the domain of a feature can be continuous or categorical. Therefore, counterfactual explanations, also called counterfactuals, belongs to the family of example-based explanations (Aamodt and Plaza, 1994). Other example-based explanations are prototypes, criticisms, and influential instances (Molnar, 2020). However, these instances are labeled with the same class of the instance x under analysis, i.e., $b(x) = b(x')$. Thus, none of them is able to reveal "why" $b(x) = y$ and not $b(x) \neq y$, while counterfactual explanations can. On the other hand, the not null differences between x and a counterfactual x' reveals exactly what should have been different in x for having a different outcome, i.e.,$\delta_{x,x'} = \{x'_i | \forall i = 1, ..., m s.t. x'_i \neq x_i\}$.

Practically, a counterfactual explanation C, can be composed by a single counterfactual example $C = \{x'\}$, or by a set of counterfactual examples $C = \{x'_1, ..., x'_h\}$. A counterfactual explainer is defined as a function able to return a counterfactual explanation C as follows in the next paragraph.

Counterfactual explainer definition. A counterfactual explainer is a function f_k that takes as input a classifier b, a set X of known instances, and a given instance of interest x, and with its application $C = f_k(x, b, X)$ returns a set $C = \{x'_1, ..., x'_h\}$ of $h \leq k$ of valid counterfactual examples where k is the number of counterfactuals required.

Most of the counterfactual explainers in the literature are designed as f_1 function to return a single valid counterfactual. If $C = \emptyset$, i.e., $h = 0$, it means that the explainer was not able to find any valid counterfactual.

Properties of counterfactual explanations. Research in counterfactual explanations has focused on addressing the problem of finding counterfactual examples guaranteeing some desirable properties. In the following subsection, the most widely used and shared desirable properties of counterfactual explanations and counterfactual explainers are illustrated and formalized: validity, minimality, similarity, plausibility, discriminative power, actionability, causality, and diversity. The most rigorous and inspiring works in this direction are Mothilal et al. (2020); Verma et al. (2024b).

- *Validity.* A counterfactual x is valid if and only if it actually changes the classification outcome with respect to the original one, i.e., $b(x') \neq b(x)$.

- *Minimality (sparsity).* There should not be any other valid counterfactual example x'' such that the number of different attribute value pairs between x and x' is higher than the number of different attribute value pairs between x and x''. It is said that x' is minimal if and only if $\nexists\ x''\ s.t.\ |\delta_{x,x''}| < |\delta_{x,x'}|$, where $|A|$ returns the size of set A.

- *Similarity.* A counterfactual x should be similar to x, i.e., given a distance function d in the domain of x, the distance between x and x' should be as small as possible $d(x, x) < \epsilon$, where ϵ is a predefined maximum distance threshold. Similarity is often referred to as *proximity.*

- *Plausibility.* Given a reference population X, a counterfactual x is plausible if the feature values in x are coherent with those in X. This practically means that the feature values of x should not be higher/smaller than those observable in X, and that x should not be labeled as an outlier with respect to the instances in X. Plausibility helps in increasing trust towards the explanation: it would be hard to trust a counterfactual if it is a combination of features that are unrealistic with respect to existing examples. On the contrary, a plausible counterfactual is "realistic" because it is "similar" to the known dataset and adheres to observed correlations among the features. Plausibility is also named *feasibility* or *reliability.* Various approaches are being proposed to check for plausibility. Laugel et al. (2019) proposes to check that a counterfactual is plausible (justified in the paper) through a concept of ϵ-chain distance with respect to a real record in X. Artelt and Hammer (2020) suggests adding specific constraints to control and measure plausibility in terms of density, i.e., a plausible counterfactual x must lie in a dense area with respect to the instances in X. In Artelt et al. (2021) are presented evaluation measures showing that plausibility also helps for robustness and stability of counterfactual explanations.

- *Discriminative Power.* A counterfactual x should show a high discriminative power for recognizing the reasons for the decision outcome (Guidotti et al., 2019; Kim et al., 2016; Kommiya Mothilal et al., 2021). Indeed, being a counterfactual, an explanation must help in figuring out why a different output can be obtained with x'. In other words, looking at x and x', also as humans would have classified $b(x) = y$ and $b(x') = y$ due to the differences $\delta_{x,x'}$ between x and x'. Note that, with respect to the above definition and the literature cited above, the discriminative power is defined based on a subjective basis that can be difficult to quantify without experiments involving humans. This issue is typically addressed by relying on simple decision models that are supposed to approximate human behaviour.

- *Actionability.* Given a set A of *actionable features*, i.e., features that can be mutated, a counterfactual x' is actionable if and only if all the differences between x and x' refers only to actionable features, i.e., $\nexists\ a_i \in \delta_{x,x'}\ s.t.\ x'_i = (a_i, v_i) \wedge x_i \notin A$. Examples of non-actionable features that cannot be changed in a counterfactual are age, gender, race, etc. Indeed, a counterfactual should never change the non-actionable (immutable) features. Actionability is also referred to as *feasibility.* In Ustun et al. (2019) is used the term *recourse* to indicate a counterfactual that accounts for the actionability of the features changed. It is important to underline that the above formalization of actionability is a soft one and could not be sufficient if the infeasibility of a counterfactual comes from a combination of different factors, i.e., living in

a particular place and having a particular job that is impossible due to the contextual circumstances.

- *Causality.* Let G be a Directed Acyclic Graph (DAG) where every node models a feature and there is a directed edge from i to j if i contributes in causing j. The DAG G describes the known causalities among features. Thus, given a DAG G, a counterfactual x respects the causalities in G if and only if $\forall x'_i = (a_i, \in_i) \in \delta_{x,x'}$ such that the node i in G has at least an incoming/outcoming edge, the value vi maintains any known causal relation between i and the values $v_{j_1}, ..., v_{j_m}$, where the features $j_1, ..., j_m$ identifies the nodes connected with i in G. Indeed, in order to be really plausible and actionable, a counterfactual should maintain any known causal relationship between features. For instance, increasing the number of years for a loan typically implies to increase also the interest rate. Note that this notion of causality is not the same causality captured by counterfactuals. Indeed, counterfactuals model (Pearl, 2009) causality between input and outcome, while the desired property discussed here is among features, and it is also connected to plausibility.
- *Diversity.* Let $C = \{x'_1, ..., x'_k\}$ be a set of k (valid) counterfactuals for the instance x. The counterfactual explanation C should be formed by diverse counterfactuals, i.e., while every counterfactual $x'_i \in C$ should be minimal and similar to x, the difference among all the counterfactuals in C should be maximized (Mothilal et al., 2020; Tsirtsis and Gomez Rodriguez, 2020). For instance, three (similar) counterfactuals saying that a yearly income of 15,000$, of 15,100$, and of 14,800$ is going to change the outcome are less useful than three (different) counterfactuals saying that the outcome can be changed (i) with a yearly income of 15,000$, (ii) by owning a car, or (iii) by first paying back the other debts. Indeed, with the second set of diverse counterfactuals, there are more possible actions to change the classification outcome.

In addition, it is possible to say that a counterfactual explanation is not available if the explanation method failed to find it.

The level of satisfaction of these properties by a certain counterfactual example x, or set of counterfactual examples C, can be used to measure the goodness of an explanation.

Properties of counterfactual explainers. Besides, research in counterfactual explanations (Bodria et al., 2023; Guidotti and Ruggieri, 2019) aimed at guaranteeing some desirable properties for the counterfactual explainers:

- *Efficiency.* An explainer f should return the set C of counterfactuals fast enough to ensure that they can be used in real life applications.
- *Stability.* Given two similar instances x_1 and x_2 obtaining the same classification from the classifier b, i.e., $y = b(x_1) = b(x_2)$, then an explainer f should return two similar set C_1, C_2 of counterfactuals. Thus, the counterfactual explainer f should show *stability* across various explanations such that similar instances would receive similar explanations. Stability is often referred to as *robustness*.
- *Fairness.* A counterfactual explainer is fair if, given a record x, any counterfactual explanation x' for x is valid both in the "actual world" and in the "counterfactual world" when in x' can also be applied changes leading it to

belong to a different demographic group. Various scenarios of counterfactual fair explanations are described in detail in Kusner et al. (2018); Von Kügelgen et al. (2022). The features that can be changed to check the fairness largely correspond to the non-actionable ones.

Graph Counterfactual Explainability

Multi-class minimal counterfactual examples. Let Φ be a prediction model that classifies x into a class $c \in C$ from a set of classes C. Let X' be the set of possible counterfactual examples x' and $S_{inst}(x, x')$ be a similarity measure that tells how similar x' is to x. Then, the set of counterfactual examples with respect to Φ is defined as follows:

$$\begin{aligned} s(c', x) &:= \max_{x' \in X', x \neq x'} \{\mathcal{S}_{inst}(x, x') | \Phi(x') = c'\} \\ \mathcal{E}_\Phi(x) &= \bigcup_{c' \in C - \{c\}} \{x' \in X' | x \neq x', \mathcal{S}_{inst}(x, x') = s(c', x)\} \end{aligned} \tag{2.1.13.1}$$

According to eq. (2.1.13.1), $\mathcal{E}_\Phi(x)$ has the maximally similar counterfactual examples $x' \in X'$ to the original graph x for each class c' that is different to the prediction on x (i.e.,$c' \in C - \{c\}$ where $c = \Phi(x)$). More specifically, those counterfactual examples that must be different from the original instance x can be found such that they maximise a similarity function with respect to the original instance $S_{inst}(x, x')$. Accordingly, it is possible to refer to the counterfactual examples of a specific class $c' \in C - \{c\}$ as $\mathcal{E}_{c',\Phi}(x) = \{x' \in X' | x \neq x', \mathcal{S}_{inst}(x, x') = s(c', x)\}$.

The definition above has two main advantages with respect to the ones previously provided in the literature: i.e., (1) it supports all the graph-based tasks in a multi-class scenario, and (2) it contains all the minimal counterfactual examples for each class $c \in C$. Furthermore, differently from the majority of the formalisations, using a similarity function $S_{inst}(x, x')$ is more beneficial because of its flexibility since it might consider both the attributes and the structure of the graph (i.e vertices, vertex/edge attributes, and edges). Finally, the counterfactual example is constrained to be different from the original instance (i.e., $x' \neq x$).

Global minimal counterfactual example. Let Φ be a prediction model that classifies x into a class $c \in C$. Let X' is the set that contains all the possible counterfactual examples x'. The global minimal counterfactual example $\mathcal{E}^*_\Phi(x)$ of x is defined as follows:

$$\mathcal{E}^*_\Phi(x) = \arg \max_{x' \in X'} \mathcal{S}_{inst}(x, x') \tag{2.1.13.2}$$

It is possible to extend eq. (2.1.13.1) to consider also vertex and edge classification tasks. The only component that changes is the prediction model Φ. Recall that x denotes an instance that can be a vertex or an edge belonging to the original graph G. In this way, the prediction model Φ takes in input the instance x and G to produce a class c (i.e., $\Phi(x, G) = c$). Hence, for a particular counterfactual example graph $G', \Phi(x, G') = c'$. It is advisable that the generated counterfactual $G' = (V', E')$ contains the original instance x: i.e., $x \in V'$ for vertex prediction and $x \in E'$ for edge prediction. In this way, it is possible to understand how x's relations (vertices or edges) in its vicinity have changed in G' with respect to G.

2.1.14 GRETEL

GRETEL(Prado-Romero and Stilo, 2022; Prado-Romero et al., 2023b), Graph Counterfactual Explanation Evaluation Framework, is a unified open-source framework to develop, evaluate, and test graph counterfactual explanations methods in several settings. It is implemented using the Object-Oriented paradigm and the Factory Method design pattern. The main goal is to create a generic platform that allows the researchers to speed up the process of developing and testing new graph counterfactual explanation methods. GRETEL is a highly extensible evaluation framework that promotes Open Science and the reproducibility of the evaluation by providing a set of well-defined mechanisms to integrate and manage easily: both real and synthetic datasets, ML models, state-of-the-art explanation techniques, and evaluation measures.

Overview

GRETEL overview considers the classification task as a reference, but this task can be substituted by any generic ML task (i.e. regression, anomaly detection, etc.). One assumption that must be considered is that the ML Model is a black-box one, and the explanation will be realised through a post-hoc explainability method (from now on, Explainer for simplicity). Thus, as highlighted in grey in fig. 2.15, the model must be already trained on the reference dataset. Now, supposing that the end-user has a new instance, this will be submitted to the ML Method and to the Explainer to obtain respectively the classification and its explanation. Behind the scene, what typically happens is that the Explainer might enquiry the ML Method to understand its behind mechanisms and/or access the original dataset to produce the final explanation. The interactions among the Explainer, the ML Method and the Dataset are highlighted with a dashed line to capture their not mandatory nature that depends on the application scenario and the used Explainer.

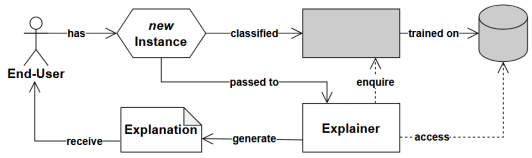


Figure 2.15. Workflow for the explanation of an instance classified by a black-box model in GRETEL.

The framework was designed keeping in mind the point of view of a researcher (academics or industry one) who wants to perform an exhaustive set of evaluations. Moreover, the framework is easily used and extended (in terms of domain, methods, datasets, metrics and ML models) by subsequent researchers. GRETEL was designed by adopting the Object-oriented design approach (Booch, 1982), where the framework's core is constituted mainly by abstract classes which need to be specialised in their implementations. To promote the framework extendability, the design pattern "Factory Method" (Lasater, 2006) was adopted, and the configuration files were leveraged as constituting part of the running framework. Moreover, to enhance the reproducibility of the evaluations, authors not only provide the complete framework with its configurations, but they also provided and allowed storing and loading of the already trained ML Models and the included datasets. Thus, the approach followed was to generate or train a dataset or a model on the fly if it was not already stored and readily available. This will mitigate the efforts needed to evaluate new settings (e.g. same explainer with other datasets or/and measures). Lastly, since the Explainer uses the ML Method agnostically, it is then seen as an Oracle to be enquired. For this reason, in GRETEL framework, as typically happens in the XAI domain, the ML

Method is referred as the Oracle.

Core components

The **Evaluator** is the component responsible for carrying out the evaluation of a specific Explainer. The Explainer must be evaluated on a constituted set of Metrics accordingly to one specific Dataset and a reference Oracle. Thus, the Evaluator e can be seen as a tuple $e =< x, d, o, M >$ where $x \in X$ is one of the explainers that must be evaluated, $d \in D$ is the considered Dataset, $o \in O$ is the reference Oracle and $M \subseteq \mathbb{M}$ is the set of metrics that must be evaluated. Then the Evaluator performs the evaluation by collecting all the performances for each instance of Dataset d.

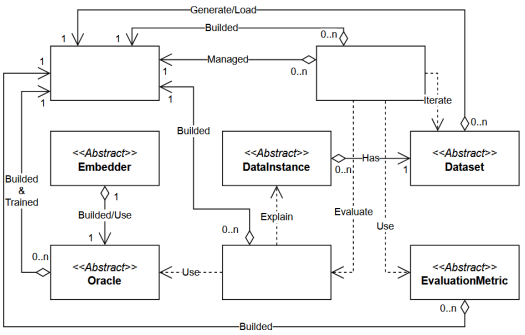


Figure 2.16. Overview of the main classes of the GRETEL Evaluation Framework and their relations.

The aim of the **EvaluatorManager** is to facilitate the task of running experiments specified by the configuration files and instantiating all the components needed to perform the complete set of evaluations. Thus, it starts by reading the configuration file that describes all the evaluations and then generates, accordingly to it, through the factory classes, all the different components of the evaluation without the need to use the constructors of the specific subclasses. Once the sets of Oracles O, Explainers X, Datasets D and Metrics $\mathbb{M}$ are instantiated, the EvaluatorManager proceed to create appropriately the Evaluators E that will be responsible for performing the individual evaluations.

The **DataInstance** class provides an abstract way to interact with data instances. It includes graph representations of the data, class labels and other fundamental information. The DataInstance can be specialised accordingly to the specific necessity that comes from different domains, as it happens for the MolecularDataInstance, which holds specific functionalities to represent the molecular graph in the “"smiles"” format (Weininger, 1988).

The **Dataset** class manages all the details related to generating, reading, writing and transforming the data accordingly to the generating/loading on the fly strategy described before. The Dataset class can be specialised to include specific information, as in the case of the MolecularDataset.

The **Oracle** class provides a generic interface for interacting with ML models. The main methods exposed by this class are `Embed`, `Fit` (to data), and `Predict`. The base class also embeds the logic used to keep track of the number of calls an explainer makes to the oracle, which is an important metric. Each Oracle also provides the specialised mechanisms needed to save and load the trained model on the disk so it can be loaded or trained on the fly. Since some Oracles need to use embedding methods, the Embedder class, which is typically coupled one-to-one with an Oracle, was also defined.

The **Explainer** is the base class used by all explanation methods. It exposes the method `Explain`, which takes an instance and an Oracle as input and returns its explanation. The goal was to keep this class as simple as possible because it is the

one used to encapsulate explanation methods. Moreover, any researcher that wants to test a new explainer in the GRETEL framework needs to extend it by providing its specific implementation.

The **EvaluationMetric** is the base class needed to define a specific metric that will be used to evaluate the quality of the Explainer.

2.2 Literature Review

This section presents a broad survey of graph neural network explainers, unifying state-of-the-art contributions (Yuan et al., 2022; Kakkad et al., 2023) into coherent definitions, taxonomies, formal frameworks, and applications (section 2.2.6).

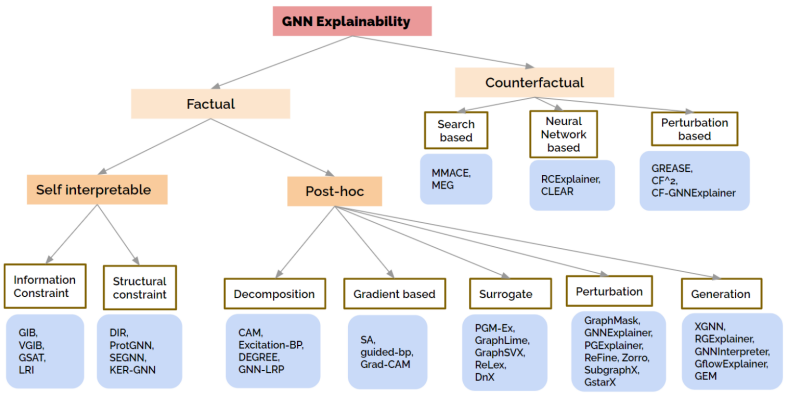


Figure 2.17. Overview of the GNN Explainability Schema.

Main Schema: Factual and Counterfactual Methods. Figure 2.17 provides an overview of the broad categorization of the existing works. Based on the type of explanations, two broad categories are first made: (1) Factual and (2) Counterfactual. Factual methods aim to find an explanation in the form of input features with the maximum influence over the prediction. These explanations can be a set of either node features or a substructure (set of nodes/edges) or both. On the other hand, counterfactual methods provide an explanation by finding the smallest change in the input graph that changes the model's prediction. Hence, counterfactual explanations can be used to find a set of similar features that can alter the prediction of the model.

Organization. In the following subsections, each category is described in detail and the summary of various explainability methods in each category is provided. In section 2.2.1,the factual approaches are described; they are further classified into self-interpretable section 2.2.3 and post-hoc categories section 2.2.2. In section 2.2.4, the counterfactual methods are categorized into perturbation-based, neural network-based and search-based methods. Lastly, section 2.2.5 presents three special categories of explainers such as temporal, global and causality-based.

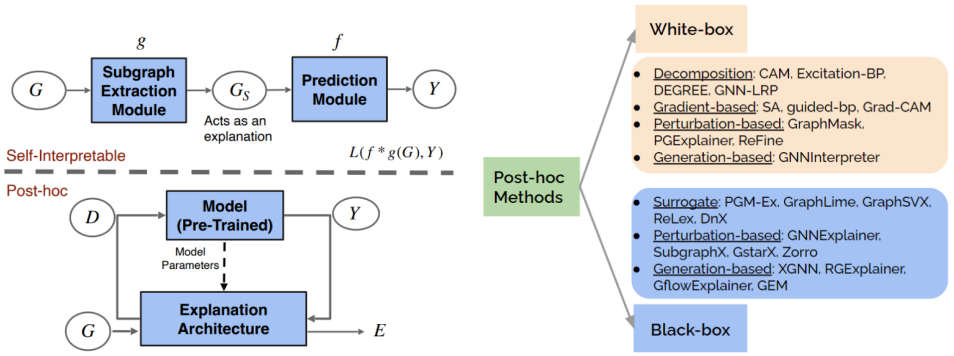
Based on what types of explanations are provided, different techniques are categorized into two main classes: instance-level methods and model-level methods. Instance-level methods provide input-dependent explanations for each input graph. Given an input graph, these methods explain deep models by identifying important input

features for its prediction. Model-level methods explain graph neural networks without respect to any specific input example. The input-independent explanations are high-level and only explain general behaviours. Overall, these two categories of methods explain deep graph models from different views. Instance-level methods provide example-specific explanations while model-level methods provide high-level insights and a generic understanding. To verify and trust deep graph models, it requires human supervision to check the explanations. For instance-level methods, more human supervisions are needed since experts need to explore the explanations for different input graphs. For model-level methods, since the explanations are high-level, less human supervisions are involved. Furthermore, the explanations of instance-level methods are based on real input instances so that they are easier to understand. However, the explanations for model-level methods may not be human-intelligible since the obtained graph patterns may not even exist in the real-world. Overall, these two types of methods can be combined together to better understand deep graph models, and hence it is necessary to investigate both of them.

2.2.1 Factual

Factual explainer methods are broadly classified into two categories based on the nature of the integration of the explainability architecture with the main model as follows.

Figure 2.18. (a) **Self-interpretable and post-hoc architectures**. (b) **White-box and Black-box post-hoc methods**: Methods are shown in the individual categories.



(a) Self-Interpretable & post-hoc. **(b)** Categorization of Post-hoc methods.

- **Post-hoc** methods do not have the explainable architecture inbuilt into the model to attribute a model's prediction to the input. As seen in fig. 2.18a, the explainability architecture (EA) is separated from the model which is pre-trained with fixed weights. For any instance G, post-hoc methods generate an explanation using the model's input $\mathcal{D}$, output Y and sometimes even internal parameters of the model. Note that different EAs use different inputs $\mathcal{D}$ that are fed to the model. Post-hoc methods might not be always accurate as they may end up extracting features that are spuriously correlated with the task (Zhang et al., 2022b; Rudin, 2019; Miao et al., 2022a; Luong et al., 2023).
- **Self-interpretable** explainable methods, contrary to post-hoc methods, design explainability architecture directly inside the model. As seen in fig. 2.18a, these methods usually have two modules. The subgraph extraction module (the function g) uses constraints to find an informative subgraph G_s from the input graph G. Then, the prediction module f uses G_s to predict label Y .

G_s also acts as an explanation. Both modules are trained together with an objective $L(f \circ g(G), Y)$ to minimize the loss between prediction $f \circ g(G)$ and label Y . One major drawback of self-interpretable models is that the good interpretability of the models is often at the cost of the prediction accuracy (Miao et al., 2022a).

2.2.2 Post-hoc

Post-hoc methods are divided based on their approaches used to find explanation into the following categories: Decomposition-based methods (section 2.2.2), Gradient-based methods (section 2.2.2), Surrogate methods (section 2.2.2), Perturbation-based methods (section 2.2.2), Generation-based methods. The post-hoc methods can also be categorized based on their requirement to access the internal parameters of the model. As seen in fig. 2.18b, this division results into the following categories of the methods: white-box and black-box.

White-box methods require access to internal model parameters or embeddings to provide explanations. For instance, all decomposition-based methods (section 2.2.2) require model parameters such as node weights of each layer to compute an importance score of different parts of the input. Even gradient based methods (section 2.2.2) require access to the gradients. Thus, all methods in these categories are considered as white-box methods and they are not suitable in cases where a model's internal parameters are inaccessible.

Black-box methods, contrary to the white-box ones, do not require access to the model's internal parameters. For instance, all approaches in the category of surrogate methods (section 2.2.2) generate a local dataset using the model's input and output. Since these methods do not require access to the model parameters, all of them can be categorized as black-box methods.

Decomposition-based methods

These methods consider the prediction of the model as a score that is decomposed and distributed backwards in a layer by layer fashion till it reaches the input. The score of different parts of the input can be construed as its importance to the prediction. However, the decomposition technique can vary across methods. They also require internal parameters of the model to calculate the score. Hence, these explanation methods are considered as white-box methods. Table 2.1 provides a summary of these methods.

Table 2.1. Key highlights of *decomposition-based* methods.

Method	Parameters	Form of Explanation	Task
CAM (Pope et al., 2019)	Node embedding of last layer and MLP weights	Node importance	Graph Classification
Excitation-BP (Pope et al., 2019)	Weights of all GNN layers	Node importance	Node and Graph Classification
DEGREE (Feng et al., 2023)	Decomposes messages and requires all parameters	Similar nodes	Node and Graph Classification
GNN-LRP (Schnake et al., 2021)	Weights of all layers	Collection of edges	Node and Graph Classification

One of the decomposition-based methods, **CAM** (Pope et al., 2019) aims at constructing the explanation of GNNs that have a Global Average Pooling (GAP) layer and a fully connected layer as the final classifier. Let e_n be the final embedding of node n just before the GAP layer and w^c be the weight vector of the classifier for the class C. The importance score of the node n is computed as $(w^c)^T e_n$. This means that the node's contribution to the class score y^c is taken as the importance. It is

clear that this method is restricted to GNNs that have a GAP layer and perform only the graph classification task.

Another method, **Excitation-BP** (Pope et al., 2019) considers that the final probability of the prediction can be decomposed into excitations from different neurons. The output of a neuron can be intuitively understood as the weighted sum of excitations from the connected neurons in the previous layer combined with a non-linear function where the weights are the usual neural network parameters. With this, the output probability can be distributed to the neurons in the previous layer according to the ratios of these weights. Finally, the importance of a node is obtained by combining the excitations of all the feature maps of that node.

Contrary to other methods, **DEGREE** (Feng et al., 2023) finds explanation in the form of subgraph structures. First, it decomposes the message passing feed-forward propagation mechanism of the GNN to find a contribution score of a group of target nodes. Next, it uses an agglomeration algorithm that greedily finds the most influential subgraph as the explanation. **GNN-LRP** (Schnake et al., 2021) is based on the concept that the function modeled by GNN is a polynomial function in the vicinity of a specific input. The prediction score is decomposed by approximating the higher order Taylor expansion using layer-wise relevance propagation (Bach et al., 2015). This differs from other decomposition-based methods not only in the decomposition technique but also in the score attribution. While other methods attribute scores to nodes or edges, GNN-LRP attributes scores to walks i.e., a collection of edges.

Gradient-based methods

The gradient-based explainer methods follow the following key idea. Gradients represent the rate of change, and the gradient of the prediction with respect to the input represents how sensitive the prediction is to the input. This sensitivity is seen as a measure of importance. Table 2.2 provides a summary of these methods.

Table 2.2. Key highlights of gradient-based methods.

Method	Explanation Type	Task	Explanation Target	Datasets Evaluated
SA (Baldassarre and Azizpour, 2019)	Instance level	Graph classification Node classification	Nodes, Node features Edges, Edge features	Infection, ESOL (Delaney, 2004)
Guided-BP (Baldassarre and Azizpour, 2019)	Instance level	Graph classification Node classification	Nodes, Node features Edges, Edge features	Infection, ESOL (Delaney, 2004)
Grad-CAM (Pope et al., 2019)	Instance level	Node classification	Nodes, Node features	BBBP, BACE, TOX21 (Morris et al., 2020)

Sensitivity Analysis (SA) (Baldassarre and Azizpour, 2019) is one of the earlier methods to use gradients to explain GNNs. Let x be an input, which can be a node or an edge feature vector, $SA(x)$ be its importance, and ϕ be the GNN model, then the importance is computed as $SA(x) \propto ||\nabla_x \phi(x)||^2$. The intuition behind this method is based on the aforementioned sensitivity to the input. **Guided Backpropagation** (Guided-BP) (Baldassarre and Azizpour, 2019), a slightly modified version of the previous method SA, follows a similar idea except the fact that the negative gradients are clipped to zero during the backpropagation. This is done to preserve only inputs that have an excitatory effect on the output. Intuitively, since positive and negative gradients have opposing effect on the output, using both of them could result in less accurate explanations.

The method **Grad-CAM** (Pope et al., 2019) builds upon **CAM** (Pope et al., 2019) (see section 2.2.2), and uses gradients with respect to the final node embeddings to compute the importance scores. The importance score is $(\frac{1}{N}\sum_{n=1}^{N} \nabla_{e_n}(y^c))^T e_n$,

where e_n is the final embedding of node n just before the GAP layer, and w^c is the weight vector of the classifier for class C, $g = \frac{1}{N}\sum_{n=1}^{N} e_n$ is the vector after the GAP layer, and the final class score y^c of C is $w^T g$. This removes the restriction about the necessity of GAP layer. This equation shows that the importance of each node is computed as the weighted sum of the feature maps of the node embeddings, where the weights are the gradients of the output with respect to the feature maps.

All the above methods depend on this particular intuition that the gradients can be good indicators of importance. However, this might not be useful in many settings. Gradients indicates sensitivity which does not reflect importance accurately. Moreover, saturation regions where the prediction of the model does not change significantly with the input, can be seen as another issue in SA and Guided-BP.

Surrogate methods

Within a large range of input values, the relationship between input and output can be complex. Hence, complex functions are needed to model this relationship and the corresponding model might not be interpretable. However, in a smaller range of input values, the relationship between input and output can be approximated by simpler and interpretable functions. This intuition leads to surrogate methods that fit a simple and interpretable surrogate model in the locality of the prediction. Table 2.3 shows different locality-based data extraction techniques and surrogate models used by surrogate methods. This surrogate model can then be used to generate explanations. As seen in fig. 2.19a, these methods adopt a two-step approach. Given an instance G, they first generate data from the prediction's neighbourhood by utilizing multiple inputs D within the vicinity and recording the model's prediction Y . Subsequently, a surrogate model is employed to train on this data. The explanation E provided by the surrogate model serves as an explanation for the original prediction.

Table 2.3. Key highlights of *surrogate methods*

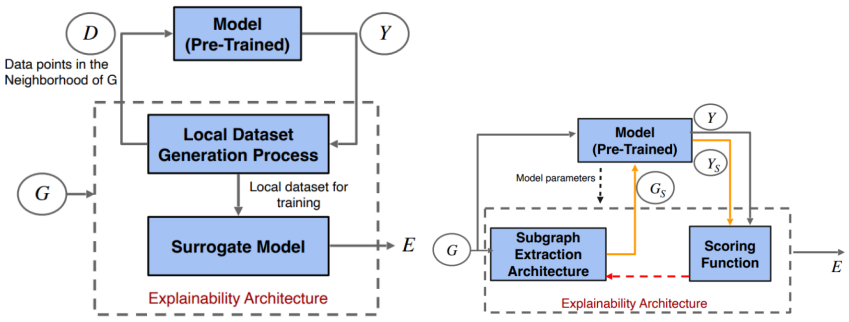
Method	Local Dataset extraction	Surrogate model	Explanation
GraphLime (Huang et al., 2022b)	N-hop neighbor nodes	HSIC Lasso	Weights of the model
PGMExplainer (Vu and Thai, 2020)	Random node feature perturbation	Bayesian network	DAG of conditional dependence
ReLex (Zhang et al., 2021)	Random sampling of connected subgraphs	GCN	Perturbation-based method
DistillnExplain (Pereira et al., 2023)	Entire dataset	Knowledge distilled Simple GCN (Wu et al., 2019)	Convex programming, decomposition
GraphSVX (Duval and Malliaros, 2021)	Input perturbations via mask generator	Weighted Linear Regression (WLR)	Weights of WLR

PGMExplainer (Vu and Thai, 2020) constructs a Bayesian network to explain the prediction. First, it creates a tabular dataset by random perturbations on node features of multiple nodes of the computational graph and records its influence on the prediction. A grow-shrink algorithm is used to select top influential nodes. Using structure learning, a Bayesian network is learnt that optimizes the Bayesian Information Criterion (BIC) scores and the DAG of conditional probabilities act as an explanation. In **GraphLime** (Huang et al., 2022b), the local explanations are based on Hilbert-Schmidt Independence Criterion Lasso (HSIC Lasso) model, which is a kernel-based nonlinear interpretable feature selection algorithm. This method assumes that the node features in the original graph are easily interpretable. The HSIC model takes a node and its N-hop neighbourhood (for some N), and selects a subset of node features that are the most influential to the prediction. These selected features act as the explanation. To construct a local dataset, **GraphSVX** (Duval and Malliaros, 2021) uses a mask generator to jointly perturb the nodes and the features and observes its effects on the predictions. The mask generator isolates the masked nodes and replaces masked features by its expected values. It then fits a

weighted linear regression model (WLR) on the local dataset. The coefficients of WLR act as explanations.

The next two approaches use GNN based models to act as surrogate models. **RelEx** (Zhang et al., 2021) uses a BFS-based sampling strategy to select nodes and then perturb them to create the local dataset. Then, a GCN model with residual connections is used to fit this dataset. In contrast to other methods in this category, the surrogate model of RelEx is not interpretable. Hence, it uses perturbationbased strategy to find a mask that acts as explanation. Note that the surrogate model is more complex compared to other methods and it requires the use of another explanation method to derive explanations from the surrogate model. **DistilnExplain** (DnX) (Pereira et al., 2023) first learns a surrogate GNN via knowledge distillation and then provides an explanation by solving a simple convex program. In contrast to RelEx, DnX uses a simpler surrogate model which is a linear architecture termed as Simplified Graph convolution (SGC) (Wu et al., 2019). SGC does not have any non-linear activation layers and uses a single parameter matrix across layers. The parameters in SGC are learned via knowledge distillation with an objective to minimize the KL-divergence between the predictions by SGC and the model. Furthermore, explanations can be derived from SGC by solving a simple convex program.

Figure 2.19. Generic schemas for (a) **surrogate** and (b) **perturbation-based** methods.



(a) Surrogate methods. **(b)** Perturbation-based methods.

Perturbation-based methods

These methods find *important subgraphs* as explanations by perturbing the input. Figure 2.19b presents two key modules of these methods: the subgraph extraction module and the scoring function module. For an input G, the subgraph extraction module extracts a subgraph G_s. The model predictions Y_s for subgraphs are scored against the actual predictions Y using a scoring function. The feedback from the scoring function can be used to train the subgraph extraction module. In some cases, model parameters are also used as the training input for the subgraph extraction module. These methods provide explanations E in the form of a subgraph structure and some also provide node features as explanations. Section 2.2.2 presents a summary of these methods.

GNNExplainer (Ying et al., 2019) is one of the initial efforts towards the explainability of GNNs. It identifies an explanation in the form of a Subgraph including a subset of node features that have the maximum influence on the prediction. It learns continuous masks for both adjacency matrix and features by optimizing cross entropy between the class label and model prediction on the masked subgraph. In

Table 2.4. Key highlights of the *perturbation-based* methods. Note that MI is mutula information, SV is Shapley value, and Explanation denotes node feature explanation.

Method	Subgraph Extraction Strategy	Scoring function	Constraints	Explanation
GNNExplainer (Ying et al., 2019)	Continuous relaxation	MI	Size	Yes
SubgraphX (Yuan et al., 2021)	Monte Carlo Tree Search	SV	Size, connectivity	No
GraphMask (Schlichtkrull et al., 2020)	Layer-wise parameterized edge selection	L_0 norm	Prediction divergence	No
PGExplainer (Luo et al., 2020)	Parameterized edge selection	MI	Size and/or connectivity	No
Zorro (Funke et al., 2022)	Greedy selection	Fidelity	Threshold fidelity	Yes
ReFine (Wang et al., 2021b)	Parameterized edge attribution	MI	Number of edges	No
GstarX (Zhang et al., 2022a)	Monte Carlo sampling	HN-value	Size	No

a follow-up work, **PGExplainer** (Luo et al., 2020) extends the idea in GNNExplainer by assuming the graph to be a random Gilbert graph, where the probability distribution of edges is conditionally independent. The distribution of each edge is independently modeled as a Bernoulli distribution, i.e., each edge has a different parametric distribution. These parameters are modeled by a neural network (MLP), and the parameters of this MLP is computed by optimizing the mutual information between the explanation subgraph and the predictions of the underlying GNN model. Another masking-related method, **GraphMask** (Schlichtkrull et al., 2020) provides an explanation by learning a parameterized edge mask that predicts the edge to drop at every layer. A single-layer MLP classifier is trained to predict the edges that can be dropped. To keep the topology unaffected, these edges are not dropped but are replaced by a learned baseline vector. The training objective is to minimize the L_0 norm i.e., the total number of edges not masked, such that the prediction output remains within a tolerance level. To make the objective differentiable, it uses sparse relaxations through the reparameterization trick and the hard concrete distribution (Maddison et al., 2016; Jang et al., 2016).

Another approach, **Zorro** (Funke et al., 2022), finds explanations in the form of important nodes and features that maximizes *Fidelity* (see section 4.3.3). It uses a greedy approach that selects the node and the feature at each step with the highest fidelity score. Fidelity is computed as the expected validity of the perturbed input. The approach uses a discrete mask for selecting a subgraph without any backpropagation. A two-staged approach, **ReFine** (Wang et al., 2021b) consists of the edge attribution or pre-training and the edge selection or fine-tuning steps. During pre-training, a GNN and an MLP are trained to find the edge probabilities for the entire class by maximizing mutual information and contrastive loss between classes. During the fine-tuning step, the edge probabilities from the previous stage are used to sample edges and find an explanation that maximizes mutual information for a specific instance.

The next two approaches use cooperative game theoretic techniques. **SubgraphX** (Yuan et al., 2021) applies the Monte Carlo Tree search technique for subgraph exploration and uses the Shapley value (Shapley, 1953) to measure the importance of the subgraphs. For the search algorithm, the child nodes are obtained by pruning the parent graph. In computing the Shapley values, the Monte Carlo sampling helps to find a coalition set and the prediction from the GNN is used as the pay-off in the game. In a subsequent work, **GStarX** (Zhang et al., 2022a) uses a different technique from cooperative game theory known as HN value (Hamiache and Navarro, 2020), to compute importance scores of a node for both graph and node classification tasks. In contrast to the Shapley value, the HN value is a structure-aware metric. Since computing the HN values is expensive, Monte Carlo sampling is used for large graphs. The nodes with the top-k highest HN values act as an explanation.

Generation-based methods

Generation-based approaches either use generative models or graph generators to derive instance-level or model-level explanations. Furthermore, to ensure the validity of the generated graphs, different approaches have been proposed. Table 2.5 provides a summary of the generation-based methods.

Table 2.5. Key highlights of the generation-based methods

Methods	Explanation Type	Optimization	Constraints	Task
XGNN (Yuan et al., 2020)	Model level	RL-policy gradient	Domain specific rules	Graph classification
RG-Explainer (Shan et al., 2021)	Instance level	RL-policy gradient	Size, radius, similarity	Node & Graph classification
GFLOW Explainer (Li et al., 2023)	Instance level	TD flow matching	Connectivity / cut vertex	Node & Graph classification
GNNInterpreter (Wang and Shen, 2022)	Model level	Continuous relaxation	Similarity to mean	Graph Classification
GEM (Lin et al., 2021)	Instance level	Autoencoder	Graph validity rules	Node & Graph Classification

XGNN (Yuan et al., 2020) provides model-level explanations by generating key subgraph patterns to maximize prediction for a certain class. The subgraph is generated using a Reinforcement learning (RL) based graph generator which is optimized using policy gradient. In the setup for the RL agent, the previous graph is the state; adding an edge is an action; and the model prediction along with the validity rules acts as the reward. Unsurprisingly, the validity rules are specified based on domain knowledge. Another RL-based method, **RG-Explainer** (Shan et al., 2021) formulates the underlying problem as combinatorial optimization instead of using continuous relaxation or search methods to find the subgraph. A starting point is selected using an MLP which acts as an input to the graph generator. The graph generator is an RL agent that optimizes for the policy using policy gradient with subgraph as the state, adding neighboring nodes as the action, and the function of the cross entropy loss as the reward.

A non-RL method, **GNNinterpreter** (Wang and Shen, 2022) is a generative model-level explanation method for the graph classification task. Its objective is to maximize the likelihood of predicting the explanation graph correctly for a given class. The similarity between the explanation graph embedding and the mean embedding of all graphs act as an optimization constraint. Intuitively, this ensures that the explanation graph stays closer to the domain and is meaningful. Since the adjacency matrix and sometimes even the features can be categorical, GNNinterpreter uses the Gumbel softmax method (Jang et al., 2016) to enable backpropagation of gradients. Contrary to XGNN with domain-specific hand-crafted rules, GNNinterpreter uses numerical optimization and does not need any domain knowledge.

GFLOW Explainer (Li et al., 2023) uses GFLOWNETs as the generative component. The objective is to construct a TD-like flow matching condition (Bengio et al., 2023) to learn a policy to generate a subgraph by sequentially adding neighbors (nodes) such that the probability of the subgraph of a class is proportional to the mutual information between the label and the distribution of possible subgraphs. A *state* consists of several nodes with the initial state as the single most influential node and the end state that satisfies the stopping criteria. *Action* is adding a node and the *reward* is a function of the cross-entropy loss.

GEM (Lin et al., 2021) uses the principles of Granger causality to generate ground-truth explanations which are used to train the explainer. It quantifies the causal contribution of each edge in the computational graph by the difference in the loss of the model with and without the edge. This distilled ground-truth for the computation graph is used to train the generative auto-encoder based explainer. This explainer provides an explanation for any instance in the form of the subgraph

of the computation graph.

2.2.3 Self-interpretable

In self-interpretable methods, the explainable procedure is intrinsic to the model. Such methods derive explainability by incorporating interpretability constraints. These methods use either information constraints or cardinality (structural) constraints to derive an informative subgraph which is used for both the prediction and the explanation. Based on the design of the explainability, the self-interpretable methods are further classified into two types based on the imposed constraints (fig. 2.20).

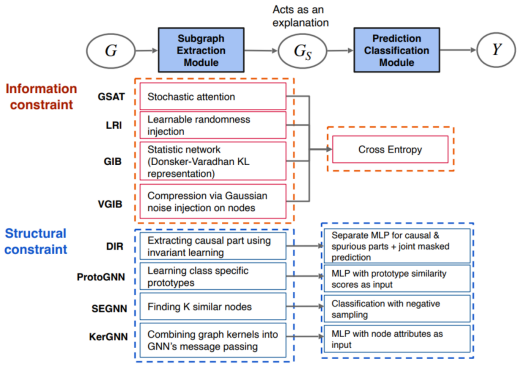


Figure 2.20. Self-interpretable methods. Every self-interpretable method has a s*ubgraph extraction* and a *prediction module*. The subgraph extraction module (the function g) uses constraints to find an informative subgraph G_s from input graph G. The prediction module uses G_s to predict label Y. This also shows the techniques used by each method to implement these individual modules.

Methods with information constraints

One of the major challenges in constructing explanations via subgraphs is that the critical subgraphs may have different sizes and can be irregular. Thus, constraining the size of the explanation may not be appropriate for the underlying prediction task. To address this challenge, the methods based on information constraint use the principle of information bottleneck (IB) (Tishby and Zaslavsky, 2015) to impose constraints on the information instead of the size. For a graph G, subgraph Gs and label Y, the graph information bottleneck (GIB) objective is:

$$\max_{G_s} I(Y, G_s) \text{ such that } I(G, G_S) \leq \gamma \tag{2.2.3.1}$$

where I denotes the mutual information. Using Lagrangian multiplier β, it is possible to write the equation as:

$$\min_{G_s} \; -I(Y, G_s) + \beta \; * \; I(G, G_s) \tag{2.2.3.2}$$

As seen from the equations, GIB objective-based methods have two parts in the objective function and both are intractable. All methods approximate $I(Y, G_s)$ by calculating the cross-entropy loss. However, all methods vary in their approach in making $I(G, G_s)$ tractable i.e., all have different approaches to compressing the graph and finding the informative subgraph G_s. This subgraph is used for both

Table 2.6. Key highlights of the methods with *information constraints.*

Method	Process of calculating $I(G, G_s)$	Injection of Randomness	Subgraph Extractor Architecture
GSAT (Miao et al., 2022a)	Stochastic attention	Bernoulli as Prior for KL divergence	GNN + MLP + Reparameterization
LRI (Miao et al., 2022b)	Learnable Randomness injection	Bernoulli and Gaussian as prior	GNN + MLP + Reparameterization
GIB (Yu et al., 2020)	Donsker-Vardhan KL representation (Donsker and Varadhan, 1975)	No randomness injection	Statistic Network: GNN + MLP
VGIB (Yu et al., 2022)	Compression via Noise injection	Gaussian noise on node features	GNN + MLP + Reparameterization

prediction and interpretation. Table 6 provides the summary of all methods in this category.

GSAT (Miao et al., 2022a) uses a stochastic attention mechanism to calculate the variational upper bound for $I(G, G_s)$. First, it encodes graph G using a GNN to find the representation for each node. Then, for each node pair (u, v), GSAT uses an MLP to calculate P_{uv}. This is used to sample stochastic attention from Bernoulli distribution $Bern(P_{uv})$to extract a subgraph G_s. The variational upper bound is the KL divergence between $Bern(P_{uv})$ and $Bern(\alpha)$ where α is a hyper-parameter. Building on similar concepts, **LRI** (Miao et al., 2022b) uses both Bernoulli and Gaussian distribution as the prior distribution. LRI-Bernoulli provides the existence importance of points and LRI-Gaussian provides the location importance of the points i.e., how perturbing the location of the point in different directions affects the prediction. Another method, **GIB** (Yu et al., 2020) assumes that there is no reasonable prior distribution to solve $I(G, G_s)$ via KL divergence in the graph space. Hence, it uses the Donsker-Vardhan KL representation (Donsker and Varadhan, 1975) in the latent space. It employs a bi-level optimization wherein the statistic network of the Donsker-Varadhan representation is used to estimate $I(G, Gs)$ in the inner loop. This estimate with classification and connectivity loss is used to optimize the GIB objective in the outer loop. This bi-level training process is inefficient and unstable; and hence **VGIB** (Yu et al., 2022) uses a different compression technique. The information in the original graph is dampened by injecting noise into the node representations via a learned probability P_i for each node i. The classification loss will be higher if the informative substructure G_s^* is injected with noise. Hence, G_s^* is less likely to be injected with noise compared to label-irrelevant substructures.

Methods with structural constraints

Imposing structural constraints on the input to derive the most informative subgraph has also been a common approach. The obtained informative subgraph is used for both making predictions and generating explanations. The key difference across the methods is the set up of the structural constraints. Table 2.7 provides the key highlights of these methods.

Table 2.7. Key highlights of explainability methods with *structural constraints.* Note that NC and GC denote node and graph classification respectively.

Method	Subgraph Extraction	Explanation form	Prediction / classification module	Task
DIR (Wu et al., 2022)	Separating pattern that is invariant across interventional distribution	Invariant rationale	Separate MLP for spurious and invariant parts	GC
ProtoGNN (Zhang et al., 2022b)	Computes similarity between graph embedding and several learned diverse prototypes	Prototypes with high similarity	MLP with similarity scores as input	GC
SEGN (Dai and Wang, 2021)	Finds K nodes that have similar structure and node features via contrastive loss	Similar nodes	Classification via negative sampling of nodes	NC
KER-GNN (Feng et al., 2022)	Kernel filters integrated in message passing of GNNs	Learned kernels and output node attributes	MLP with node attributes as input	NC, GC

One of the earlier methods, **DIR** (Wu et al., 2022) finds explanations in the form of invariant causal rationales by learning to split the input into causal (C) and non-causal (S) parts. The objective is to minimize the classification loss such that Y (the prediction) is independent of S given C. To achieve this, it first creates multiple interventional distributions by conducting interventions on the training distribution. The part that is invariant across these distributions is considered a causal part. Moreover, the implementation has three key stages. First, the architecture consists

of a rationale generator (GNN) that splits the input graph into a causal part with top k edges and a non-causal part. Second, a distribution intervener, i.e., a random replacement from a set, creates perturbed distribution to infer the invariant causal parts. Finally, two classifiers are used to generate a joint prediction on causal and non-causal parts.

ProtoGNN (Zhang et al., 2022b) combines prototype learning (Schmidt et al., 2001) with GNNs. Prototype learning is a form of case-based reasoning which makes predictions for new instances by comparing them with several learned exemplar cases also called prototypes. ProtoGNN computes the similarity scores between the graph embedding and multiple learned prototypes. Moreover, these prototypes are projected onto the nearest latent training subgraph during training using the Monte Carlo tree search (Silver et al., 2017; Browne et al., 2012). The similarity scores are used for the classification task where the subgraphs with high similarities can be used for explanation. In another work, for a given unlabeled node, **SEGNN** (Dai and Wang, 2021) finds k nearest labeled nodes that have structural and feature similarities and can be used for both generating predictions and explanations. It uses contrastive loss on node representations for feature similarity and also on edge representations of local neighborhood nodes for structural similarity. Moreover, the classification loss uses negative sampling with approximate k similar nodes. These k nearest nodes can be used to derive an explanation subgraph with threshold importance.

The method, **KER-GNN** (Feng et al., 2022) integrates graph kernels into the message-passing process of GNNs to increase the expressivity of GNNs beyond the 1-WL isomorphism test. In each layer, the node embeddings are updated by computing the similarity between the node's subgraph (the node with its ego-net) and trainable filters in the form of hidden graphs. The learned graph filters can provide important structural information about the data. Moreover, the output node attributes can be used to extract important substructures.

2.2.4 Counterfactual Explanation

Counterfactual methods provides an explanation by identifying the minimal alteration in the input graph that results in a change in the model's prediction. Recently, there have been several attempts to have explanations of graph neural networks via counterfactual reasoning. These explainer methods that find counterfactuals are classified into three major categories based on the type of methods: **Perturbation-based** (section 2.2.4), **Neural framework-based** (section 2.2.4), and **Search-based** (section 2.2.4). The works are discussed in the individual categories below.

Perturbation-based methods

An intuitive way to generate counterfactuals for both the graph classification and the node classification task is to *alter the edges*, i.e., add or delete the edges in the graph such that it would change the prediction of the underlying GNN method. This alteration can be achieved by perturbing either the adjacency matrix or the computational graph of a node. The perturbation-based methods are summarized in table 2.8.

One of the initial efforts, **CF-GNNExplainer** (Lucic et al., 2022) aims to perturb the computational graph by using a binary mask matrix. It uses a binary matrix (all values are 0 or 1) P and modifies the computational graph matrix as $\tilde{A}_v = P \odot A_v$,

Table 2.8. Key highlights of *perturbation-based* methods for counterfactuals.

Method	Explanation Type	Downstream Task	Perturbation Target	Datasets Evaluated
CF-GNNExplainer (Lucic et al., 2022)	Instance level	Node Classification	Computation graph	Tree-Cycles, Tree-Grids, BA-Shapes (Ying et al., 2019)
CF^2 (Tan et al., 2022)	Instance level	Graph Classification Node Classification	Computation graph Original graph	BA-Shapes, Tree-Cycles (Ying et al., 2019) Mutag (Debnath et al., 1991), NCI1 (Wale et al., 2008), CiteSeer (Getoor, 2005)
GREASE (Chen et al., 2022b)	Instance level	Node Ranking	Computation graph	LastFM, Yelp

where A_v is the original computational graph matrix and $\tilde{A}_v$ is computational graph matrix after the perturbation. The matrix P is computed by minimizing a combination of two different loss functions:L_{pred}, and L_{dist}. They are combined using a hyper-parameter in the final loss (L) as $L_{pred} + \beta L_{dist}$. The loss function, L_{pred} quantifies the accuracy of the produced counterfactual, and L_{dist} captures the distance (or similarity) between the counterfactual graph and the original graph. In follow-up work, the method **CF^2** (Tan et al., 2022) extends the method in CF-GNNExplainer (Lucic et al., 2022) by including a contrastive loss that jointly optimizes the quality of both the factual explanation and the counterfactual one. For an input graph, G, it aims to find an optimal subgraph G_s where G_s is a good factual explanation, and $G \backslash G_s$ is a good counterfactual. These objectives are formulated as a single optimization problem with the corresponding loss as $L_{overall} = \alpha L_{factual} + (1 - \alpha) L_{counterfactual}$, where α is a hyperparameter.

Another method, **GREASE** (Chen et al., 2022b) follows the standard technique of using a perturbation matrix to generate a counterfactual, but with two key modifications mainly to accommodate GNNs used for recommendation systems instead of classification tasks. In the recommendation task, GNNs rank the items (nodes) by assigning them a score instead of classifying them. GREASE uses a loss function based on the scores given by the GNN before and after the perturbation. This score helps to rank the items or nodes. The second modification is the perturbation matrix, which acts as the mask, and is used to perturb the computational graph (l-hop neighborhood of the node) instead of perturbing the entire graph. Here l denotes the number of layers in the GNN. Similar to CF^2 (Tan et al., 2022), GREASE also optimizes counterfactual and factual explanation losses, but not jointly.

In summary, all these techniques share similarities in computing the counterfactual similarity and constructing the search space. Similarity is measured by the number of edges removed from input instances and the search space is the set of all subgraphs obtained by edge deletions in the original graph. Because of the unrestricted nature of the search space, these methods might not be ideal for graphs such as molecules, where the validity of the subgraphs has valency restrictions. On the other hand, the mentioned methods differ mainly in the loss function formulations and the perturbation operations for the downstream tasks. For instance, CF-GNNExplainer (Lucic et al., 2022) and GREASE (Chen et al., 2022b) perform node classification and regression, they can use perturbations on the computation graph. However, CF2 (Tan et al., 2022) considers both graph and node classification tasks, hence it uses perturbations on the entire graph, i.e., the adjacency matrix.

Neural framework-based methods

The approaches in this section use neural architectures to generate counterfactual graphs as opposed to the perturbation-based methods where the adjacency matrix of the input graph is minimally perturbed to generate counterfactuals. Table 2.9 summarizes these methods.

The objective of **RCExplainer** (Bajaj et al., 2021) is to identify a resilient subset

Table 2.9. Key highlights of *neural framework-based* methods for counterfactuals.

Method	Explanation Type	Downstream Task	Counterfactual Generator	Datasets Evaluated
RCExplainer (Bajaj et al., 2021)	Instance level	Graph classification Node classification	Edge prediction with Neural Network	Mutag (Debnath et al., 1991), BA-2motifs (Luo et al., 2020), NC11 (Wale et al., 2008) Tree-Cycles, Tree-Grids (Ying et al., 2019), BA-Shapes (Luo et al., 2020), BA-Community (Ying et al., 2019)
CLEAR (Ma et al., 2022)	Instance level	Graph classification Node classification	Graph generation with Variational Autoencoder	Community (Erd, 1959), Ogbg-molhiv, IMDB-M

of edges that, when removed, alter the prediction of the remaining graph. This is accomplished by modeling the implicit decision regions using graph embeddings. Even though the counterfactual graph generated by a neural architecture is used in conjunction with the adjacency matrix of the input graph, the counterfactual itself is not generated through perturbations on the adjacency matrix. RCExplainer addresses the issue of fragility where an interpretation is fragile (or non-robust) if systematic perturbations in the input graph can lead to dramatically different interpretations without changing the label. The standard explainers aim to generate good counterfactuals by choosing the closest counterfactual to the input instance and it might induce over-fitting. RCExplainer reduces this over-fitting by first clustering input graphs using polytopes, and finding good counterfactuals close to the cluster (polytope) instead of individual instances. Another method, **CLEAR** (Ma et al., 2022) generates counterfactual graphs by leveraging a graph variational autoencoder. Two major issues often seen in other explainer methods, namely, generalization and causality are addressed by the related paper.

Both methods use a generative neural model to find counterfactuals, but the generative model is different across the methods. While RCExplainer uses a neural network that takes pairwise node embeddings and predict the existence of an edge between them, CLEAR uses a variational autoencoder to generate a complete graph. This shows that while the former method cannot create nodes that are not present in the original graph, the latter can. In terms of the objective, the primary focus in RCExplainer is the robustness of the generated counterfactual, but CLEAR aims to generate counterfactuals that explain the underlying causality.

Search based methods

These methods usually depend on search techniques over the counterfactual space for relevant tasks or applications (see the highlights in table 2.10). For example, given an inactive molecule in a chemical reaction, the task is to find a similar but active molecule. Here, generative methods or perturbation methods might not be effective, and the perturbations might not even result in a valid molecule. In such cases, a good search technique through the space of counterfactuals could be more useful. An inherent challenge is that the search space of counterfactuals might be exponential in size. Hence, building efficient search algorithms is required.

Table 2.10. Key highlights of *search-based* methods for counterfactuals.

Method	Explanation Type	Downstream Task	Counterfactual Similarity Metric	Datasets Evaluated
MMACE (Wellawatte et al., 2022)	Instance level	Graph classification Node classification	Tanimoto similarity	Blood brain barrier dataset (Martins et al., 2012) Solubility data (Sorkun et al., 2019) HIV drug dataset (Numeroso and Bacciu, 2021)
GCExplainer (Huang et al., 2023)	Global	Graph classification	Graph edit distance	Mutag (Debnath et al., 1991), NCI1 (Wale et al., 2008)
MEG (Numeroso and Bacciu, 2021)	Instance level	Graph classification Node classification	Cosine similarity Tanimoto similarity	Tox21 (Morris et al., 2020), ESOL (Wu et al., 2018)

The major application is finding counterfactual examples for molecules in related tasks. The method **MMACE** (Wellawatte et al., 2022) finds counterfactuals for molecules. In the corresponding graph classification problem, it aims to classify a molecule based on a specific property. Examples include whether a molecule will permeate blood brain barrier and molecule's solubility. The search space

can be generated by a method called *Superfast Traversal, Optimization, Novelty, Exploration and Discovery (STONED)* (Nigam et al., 2021). MMACE uses this method to generate the close neighbourhood and searches with a BFS-style algorithm to find an optimal set of counterfactuals.

Similarly, **MEG** (Numeroso and Bacciu, 2021) also aims to find a counterfactual and the search space consists of molecules. However, instead of searching the space with traditional graph search algorithms, MEG uses a reinforcement learning-based approach to navigate the search space more efficiently. The reward for finding a counterfactual is defined as the inverse of the probability that the candidate molecule found by the agent is not a counterfactual. This method is applied in a classification problem of predicting toxicity of a molecule as well as in a regression problem of predicting solubility of a molecule.

Another approach **GCFExplainer** (Huang et al., 2023) uses a random walk-based method to search the counterfactual space. The objective is not to find an individual counterfactual for each input sample but to find a small set of counterfactuals that explain all or a subset of the input samples. Hence, this is a global method (see section 2.2.5). Here the counterfactual search space is obtained by applying graph edit operations on the training data. The method uses a random walk called **Vertex Reinforced Random Walk** (VRRW) (Pemantle, 1992), which is a modified version of a Markov chain where the state transition probabilities depend on the number of previous visits to that state.

Both MMACE and MEG are developed for GNNs that predict molecular properties while the objective of GCFExplainer is to generate global explanations. However, the search algorithms and the generation mechanisms of the counterfactual space are quite different. For instance, MMACE employs a graph search algorithm to locate the nearest counterfactual instance. In contrast, MEG utilizes reinforcement learning, and GCFExplainer employs random walks to achieve the same.

2.2.5 Others

In this section the explainer methods are described and categorized in three special categories: explainer for temporal GNNs, global explainers and causality-based explainers.

Explainers for Temporal GNNs

In temporal or dynamic graphs, the graph topology and node attributes evolve over time. For instance, in social networks the relationships can be dynamic, or in the citation networks co-authorships change over time. There has been effort towards explaining the GNN models that are specifically designed for such structures.

One of the earlier explainer methods on dynamic graphs is **GCN-SE** (Fan et al., 2021). GCN-SE learns the attention weights in the form of linear combination of the representations of the nodes over multiple snapshots (i.e., over time). To quantify the explanatory power of the proposed method, importance of different snapshots are evaluated via these learned weights. Another method for an explainer framework (He et al., 2022b), designs a two-step process. It uses static explainer such as PGM-explaine (Vu and Thai, 2020) to explain the underlying temporal GNN (TGNN) model (such as TGCN (Zhao et al., 2019)) for each time step separately and then it aims to discover the dominant explanations from the explanations identified by the static one. **DGExplainer** (Xie et al., 2022) also generates explanations for

dynamic GNNs by computing the relevance scores that capture the contributions of each component for a dynamic graph. More specifically, it redistributes the output activation score to the relevance of the neurons of its previous layer in the model. This process iterates until the relevance scores of the input neuron are obtained. Recently, T-GNNExplainer (Xia et al., 2022) has been proposed for temporal graph explanation where a temporal graph constituted by a sequence of temporal events. T-GNNExplainer solves the problem of finding a small set of previous events that are responsible for the model's prediction of the target event. In Liu et al. (2024), the approach involves a smooth parameterization of the GNN predicted distributions using axiomatic attribution. These distributions are assumed to be on a low-dimensional manifold. The approach models the distributional evolution as smooth curves on the manifold and reparameterize families of curves by designing a convex optimization problem. The aim is to find a unique curve that approximates the distributional evolution and will be useful for human interpretation.

The following ones also design explainers of temporal GNNs but with specific objectives or applications. Vu and Thai (2022) studies the limit of perturbation-based explanation methods. The approach constructs some specific instances of TGNNs and evaluate how reliably node-perturbation, edge-perturbation or both can reliably identify specific graph components carrying out the temporal aggregation in temporal GNNs. In Yang et al. (2023b), a novel interpretable model on temporal heterogeneous graphs has been proposed. The method constructs temporal heterogeneous graphs which represent the research interests of the target authors. After the detection task, a deep neural network has been used for the generation process of interpretation on the predicted results. This method has been applied to research interest shift detection of researchers. Another related work (Han and Zhao, 2022) is on explaining GraphRC which is a special type of GNN and popular because of its training efficiency. The proposed method explores the specific role played by each reservoir node (neuron) of GraphRC by using attention mechanism on the distinct temporal patterns in the reservoir nodes.

Global Explainers

Majority of the explainers provide explanation for specific instances and can be seen as *local explainers*. However, global explainers aim to explain the overall behaviour of the model by finding common input patterns that explain certain predictions (Yuan et al., 2020). Global explainers provide a highlevel and generic explanation compared to local explainers. However, local explainers can be more accurate compared to global explainers (Yuan et al., 2020) especially for individual instances. Global explainers are categorized into the following three types.

1. **Generation-based**: These post-hoc methods use either a generator or generative modeling to find explanations. For instance, **XGNN** (Yuan et al., 2020) uses a reinforcement learning (RL) based graph generator optimized using policy gradient. In contrast, **GNNInterpreter** (Wang and Shen, 2022) is a generative global explainer that maximizes the likelihood of explanation graph being predicted as the target class by the model (the details are in section 2.2.2).

2. **Concept-based**: These methods provide concept based explanations. Concepts are small higher level units of information that can be interpreted by humans (Ghorbani et al., 2019). The methods differ in the approaches to find concepts. **GCExplainer** (Magister et al., 2021) adapts an image ex-

planation framework known as automated concept based explanation (ACE) (Ghorbani et al., 2019) to find global explanation for graphs. It finds concepts by clustering the embeddings from the last layer of the GNN. A concept and its importance are represented by a cluster and the number of nodes in it respectively. Another method **GCneuron** (Xuanyuan et al., 2023), which is inspired by Compositional Explanations of Neurons (Mu and Andreas, 2020), finds global explanation for GNNs by finding compositional concepts aligned with neurons. A base concept is a function C on Graph G that produces a binary mask over all the input nodes V. A compositional concept is logical combination of base concepts. This method uses beam search to find compositional concept that minimizes the divergence between the concept and the neuron activation. Lastly, **GLGExplainer** (Azzolin et al., 2022) uses local explanations from PGExplainer (Luo et al., 2020) and projects them to a set of learned prototype or concepts (similar to ProtGNN (Zhang et al., 2022b)) to derive a concept vector. A concept vector is vector of distances between graph explanation and each prototype. This concept vector is then used to train an Entropy based logic explainable network (E-LEN) (Ciravegna et al., 2023) to match the prediction of the class. The logic formula from the entropy layer for each class acts as explanations.

3. **Counterfactual**: **Global counterfactual explainer** (Huang et al., 2023) finds a candidate set of counterfactuals using vertex re-inforced random walk. It then uses a greedy strategy to select the top k counterfactuals from the candidate set as global explanations. It is explained in more detail in section 2.2.4.

Causality-based Explainers

Most of the GNN classifiers learn all statistical correlation between the label and input features. As a result, these may not distinguish between causal and non-causal features and may make prediction using shortcut features (Sui et al., 2022). Shortcut features serve as confounders between the causal features and the prediction. Methods in this category attempt to reduce the confounding effect so that the model exploits causal substructure for prediction and these substructures also act as explanations. They can be categorized into the followings.

Self-interpretable methods. Methods in this category have the explainer architecture inbuilt into the model. One of the methods, **DIR** (Wu et al., 2022) creates multiple interventional distribution by conducting intervention on the training distribution. The invariant part across these distributions is considered as the causal part (see section 2.2.3 and table 2.7). **CAL** (Sui et al., 2022) uses edge and node attention to estimate causal and shortcut features of the graph. Two classifiers are used to make prediction on causal and shortcut features respectively. Loss on causal features is used as classification loss. Moreover, KL divergence is used to push the prediction based on shortcut features to have uniform distribution across classes. Finally, CAL creates an intervention graph in the representation space via random additions. The loss on this intervened graph classification is considered as the causal loss. These three loss terms are used to reduce the confounding effect and find the causal substructure that acts as explanation. **DisC** (Fan et al., 2022) uses a disentangled GNN framework to separate causal and shortcut substructures. It first learns a edge mask generator that divides the input into causal and shortcut substructures. Two separate GNNs are trained to produce disentangled representation of these substructures. Finally, these representations are used to generate unbiased

counterfactual samples by randomly permuting the shortcut representation with the causal representation.

Generation-based methods. These methods use generative modeling to find explanations and are post-hoc. **GEM** (Lin et al., 2021) trains a generative auto-encoder by finding the causal contribution of each edge in the computation graph (details are in section 2.2.2). While GEM focuses on the graph space, **OrphicX** (Lin et al., 2022) identifies the causal factors in the embedding space. It trains a variational graph autoencoder (VGAE) that has an encoder and a generator. The encoder outputs latent representations of causal and shortcut substructures of input. Generator uses both of these representations to generate the original graph and the causal representation to produce a causal mask on original graph. The information flow between the latent representation of the causal substructure and the prediction is maximized to train the explainer. The causal substructure also acts as an explanation.

2.2.6 Applications

This subsection describes the explainers methods that are relevant for specific applications in different domains such as in social networks, biology, and computer security.

Computer Security. He et al. (2022a) focuses on designing an explanation framework for cybersecurity applications using GNN models by identifying the important nodes, edges, and attributes that are contributing to the prediction. The applications include code vulnerability detection and smart contract vulnerability detection. Herath et al. (2022) proposes **CFGExplainer** for GNN oriented malware classification and identifies a subgraph of the malware control flow graph that is most important for the classification. Some other work focuses on the problem of botnet detection. The first method, **BD-GNNExplainer** (Zhu et al., 2022) extracts the explainer subgraph by reducing the loss between the classification results generated by the input subgraph and the entire input graph. The **XG-BoT** detector proposed in Lo et al. (2023) detects malicious botnet nodes in botnet communication graphs. The explainer is based on the GNNExplainer and saliency map in the XG-BoT.

Social Networks. Rath et al. (2021) studies the problem of detecting fake news spreaders in social networks. The proposed method **SCARLET** is a user-centric model that uses a GNN with attention mechanism. The attention scores help in computing the importance of the neighbors. The findings include that a person's decision to spread false information is dependent on its perception (or trust dynamics) of neighbor's credibility. On the other hand, **GCAN** (Lu and Li, 2020) uses sequence models. The aim is to find a fake tweet based on the user profile and the sequence of its retweets. The sequence models and GNNs help to learn representation of retweet propagation and representation of user interactions respectively. A co-attention mechanism is further used to learn the correlation between source tweet and retweet propagation and make prediction. In Ma et al. (2021), a GNN model has been proposed along with the explanation of its prediction for the problem on drug abuse in social networks.

Computational Biology. One of the long standing problems in neuroscience is the understanding of Brain networks, especially understanding the Regions of Interests

(ROIs) and the connectivity between them. These regions and their connectivity can be modelled as a graph. A recent work on explainability, **IBGNN** (Cui et al., 2022) explores the explainable GNN methods to solve the task of identifying ROIs and their connectivity that are indicative of brain disorders. It uses a perturbation matrix to create an edge mask, and extracts important edges and nodes. A few more works also focus on the same task of identifying ROIs, but use different explanation techniques. In Mauri et al. (2022), the method uses a perturbation matrix with feature masks and optimizes mutual information to find the explanations. Zhou et al. (2022) uses Grad-CAM (Pope et al., 2019) to find important ROIs. Abrate and Bonchi (2021) uses a search-based method to extract counterfactuals, which can serve as good candidates for important ROIs. The method uses graph edit operations to navigate from input graph to a counterfactual, but it optimizes this by using a lookup database to select edges that are the most effective in discriminating between different predicted classes. As another interesting application,Pfeifer et al. (2022) explores is related to the extraction of subgraphs in protein-protein interaction (PPI) network, where the downstream task is to detect the relevance of a protein to cancer.

Chemistry. GNNs are being used to study molecular properties extensively and often requires explanations to better understand the model's predictions. Henderson et al. (2021) focuses on improving self-interpretability of GCNs by imposing orthogonality of node features and sparsity of the GCN's weights using Gini regularization. The intuition behind the orthogonality of features is driven by the assumption that atoms in a molecule can be represented by a linear combination of orthonormal basis of wavefunctions. Another method, **APRILE** (Xu et al., 2021) aims at finding the parts of a drug molecule responsible for side effects. It uses perturbation techniques to extract an explanation. In drug design, the method in Jiménez-Luna et al. (2021) uses integrated gradients (Sundararajan et al., 2017) to assign importance to atoms (nodes) and the atomic properties (node features) to understand the properties of a drug.

Pathology. In medical diagnosis a challenging task is to understand the reason behind a particular diagnosis, whether it is made by a human or a machine learning system. To this end, explainer frameworks for the machine learning models become useful. In many cases the diagnosis data can be represented by graphs. Wu et al. (2021a) builds a graph using words, entities, clauses and sentences extracted from a patient's electronic medical record (EMR). The objective is to extract the entities most relevant for the diagnosis by training an edge mask, and is achieved by minimizing the sum of the elements in the mask matrix. Another method by Jaume et al. (2021), focuses on generating explanations for histology (micro-anatomy) images. It first converts the image into a graph of biological entities, where the nodes could be cells, tissues or some other task specific biological features. Afterwards the standard explainer techniques described in gradient (section 2.2.2) or perturbation (section 2.2.2 based methods are used to generate the explanations. Another work in this field by Yu et al. (2021), modifies the objective to optimise for both necessity and sufficiency. The explanation is generated in such a way that the mutual information between explanation subgraph and the prediction is maximized. Additionally, the mutual information between the remaining graph after removing the explanation subgraph and the prediction is minimized.

2.3 Related Work

This subsection introduces the baselines used to compare the method proposed in this work.

It is possible to distinguish between inherently explainable and black-box methods, focusing on counterfactual explanations for graph classification. While counterfactuals are well-established in images and text (Vermeire et al., 2022; Zemni et al., 2023), their application to graphs is less common (Liu et al., 2021; Ma et al., 2022; Nguyen et al., 2022; Numeroso and Bacciu, 2021; Tan et al., 2022). As categorized in Prado-Romero et al. (2023c), graph counterfactual explanation methods fall into heuristic search and learning-based approaches. This work focuses on instance-level, learning-based explainers that identify minimal perturbations to flip a prediction, providing actionable, data-point-specific insights.

Learning-based strategies often use perturbation matrices (Tan et al., 2022), reinforcement learning (RL) (Numeroso and Bacciu, 2021), or generative models (Ma et al., 2022; Prado-Romero et al., 2024). **MEG** Numeroso and Bacciu (2021) is a reinforcement learning approach that generates counterfactuals for input molecules. Its reward function integrates task-dependent regularization, influencing the policy to select actions that lead to valid molecules. Some notable methods include **CF-GNNExplainer** (Lucic et al., 2022), which learns a binary perturbation matrix to sparsify the adjacency matrix of the original graph G, guided by a sparse neural network.; **CF^2** (Tan et al., 2022), which balances factual and counterfactual reasoning through multi-objective optimization; counterfactuals are generated by removing the factual subgraph from the input and focusing on simplicity; and **CLEAR** (Ma et al., 2022), which uses a Variational Autoencoder (VAE) to generate counterfactuals as complete graphs with stochastic edge weights, conditioned on the input graph and a desired class. During decoding, a graph matching step is required to address vertex reordering, an NP-hard problem Livi and Rizzi (2013). **RSGG-CE** Prado-Romero et al. (2024) relies on a modified learning approach of Generative Adversarial Networks (GANs) and a partial-order sampling strategy on the learned edge distribution to generate robust graph counterfactual candidates. It exploits the generator to learn a graph representation, enabling stochastic estimations of the graph's topology, thus allowing the generation of counterfactuals in zero-shot. **D4Explainer** Chen et al. (2023) employs discrete denoising diffusion, progressively perturbing graphs and then filtering out irrelevant changes. D4Explainer was completely ported into GRETEL during this thesis work. Additional methods are **INDUCE** (Verma et al., 2024a) which is based on a RL-based inductive approach; **COMBINEX** (Giorgi et al., 2025) and **C2Explainer** (Ma et al., 2025). Emerging research on time-related graph counterfactuality (Prenkaj et al., 2024; Qu et al., 2024) is outside the scope of this work.

Differently from the state-of-the-art, the proposed method perform graph- and node-level explanations. In the former, changes needed to alter the prediction for an entire graph are identified. In the latter, loss information from individual nodes are leveraged, allowing the method to perturb both the local graph structure and node features with the targeted objective of changing a single node's label. Here, an oracle performs node-wise classification, and the proposed method focuses on modifying the inputs that directly influence the prediction of the targeted node.

Chapter 3

Method

This chapter reports the main theoretical definitions, methodologies, and techniques adopted in this thesis work. It is organized as follows:

- The **Problem Formulation** section (section 3.1) defines the explainability problem formulation.
- The **Proposed Method** section (section 3.2) introduces and formally defines XPlore.
- The **Algorithmic Implementation** section (section 3.3) provides the reproducibility pipeline for the proposed method.
- The **Oracles Models** section (section 3.4) reports the implemented architectures definitions and details.
- The **Theoretical Foundations of Gradient-Guided Counterfactual Perturbations** section 3.5 lays out the formal support for the proposed explanation method.

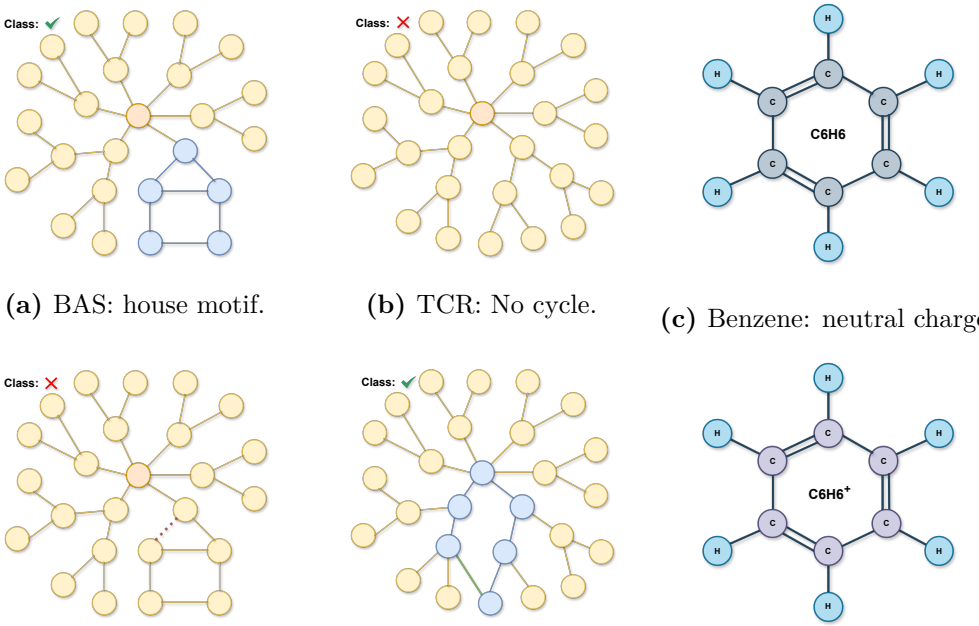
3.1 Problem Formulation

Graph Counterfactual Explanation. Suppose to have a well-trained GNN serving as an oracle $\Phi : \mathcal{G} \to \mathcal{Y}$ and an original graph instance $G \in \mathcal{G}$, with predicted label $\Phi(G)$. The objective of counterfactual explanation is to find a perturbed graph G', obtained by a counterfactual model $\mathcal{E} : \mathcal{G} \to \mathcal{G}$, that minimally deviates from G while ensuring that the oracle's prediction changes, i.e., $\Phi(G') \neq \Phi(G)$ (Prado-Romero et al., 2023c). Let $\Delta(G, G')$ denote the distance function measuring the difference between G and G'; then the problem can be stated as:

$$G^* = \underset{G' \in \mathcal{G}'}{\arg\min}\, \Delta(G, G') \text{ s.t } \Phi(G) \neq \Phi(G'). \tag{3.1.0.1}$$

As done in Lucic et al. (2022), counterfactuals are generated by reformulating the above hard-constrained formulation into a soft-unconstrained optimization by minimizing the loss function:

$$L(G, G') = L_{\text{pred}}(G, G' \mid \Phi) + \beta\, L_{\text{dist}}(G, G'). \tag{3.1.0.2}$$



(a) BAS: house motif. **(b)** TCR: No cycle. **(c)** Benzene: neutral charge.

(d) BAS: edge deletion. **(e)** TCR: edge insertion. **(f)** Benzene: positive charge.

Figure 3.1. Illustration of counterfactual explanations in graph classification. *a* and *d*: The class changes only after an **edge deletion** (red, dashed). *b* and *e*: the predicted class changes only after an **edge addition** (green). *c* and *f*: Feature perturbation of the carbon-group illustrates benzene converting from neutral to cationic by electron loss. XPlore enables these types of perturbations, alone or in combination, offering a broader search space than edge deletion only methods.

Here, L_{pred} is a prediction loss that encourages $\Phi(G') \neq \Phi(G)$ and L_{dist} is a distance loss that promotes similarity between G' and G, with the trade-off controlled by β. Thus, the optimal counterfactual example for G is obtained by solving Equation (3.1.0.2).

Node Counterfactual Explanation. The proposed method can be extended to node-level counterfactual explanations. In this scenario, rather than modifying only the target node, the entire graph is perturbed, including all node features. Moreover, the loss formulation can be adapted to encompass a set of nodes by imposing a soft constraint that preserves the class labels of nodes other than the target through the use of binary cross-entropy with logits as the loss function. This framework naturally generalizes to multi-node classification tasks, where the objective is to generate counterfactual explanations for a set of nodes rather than a single node.

Table 3.1. Notation used in the **??**.

Symbol	Description
$\mathcal{E}, \Phi$	Counterfactual and oracle model
G, G'	Original and counterfactual graphs
$A, \hat{A}$	Adjacency matrices (original and counterfactual)
$P, \bar{P}$	Original and new perturbation matrices for edges
$N, \bar{N}$	Binary and continuous node feature perturb. matrices
X, W	Node feature and weight matrices
$\bar{D}$	Degree matrix
Γ	Probability weight matrix with values $\gamma_{i,j} \in [0, 1]$
$\sigma(\cdot)$	Sigmoid activation function
softmax$(\cdot)$	Softmax function
$\mathcal{T}_\alpha(\cdot)$	Entry-wise threshold: $\mathcal{T}_\tau(X) = \mathbb{1}\{\sigma(X) \geq \alpha\}$
$\odot$	Element-wise (Hadamard) product
$\mathbf{1}\{\cdot\}$	Indicator function
K	Number of optimization iterations (e.g. 50, see section 4.3.2)
L_{pred}, L_{dist}	Prediction and distance losses

3.2 Proposed Method

XPlore is proposed to sieve the less-constrained counterfactual search space compared to the base model (Lucic et al., 2022), aiming at better explanations while reducing the out-of-distribution effect shown in Chen et al. (2023). XPlore, not only drops instance edges but also adds them, enabling the discovery of counterfactual explanations for which a class change is unattainable by edge removal alone (e.g. TCR in section 4.2; see fig. 3.1) . Moreover, by modifying node features, the proposed approach can identify counterfactuals where a class change is unachievable solely through edge modifications (or any amount thereof), but rather via feature adjustments. Two modalities for handling features are explored: i.e., **(1)** applying a gating mechanism to retain or discard node features, similarly to the edge-dropping process introduced in Lucic et al. (2022); **(2)** allowing for continuous adjustment in node features, enabling their values to increase/decrease smoothly. Formal guarantees on convergence are provided, on ℓ_1-minimality of perturbations and semantic fidelity in 3.5.

Original Adjacency matrix perturbations. Let $A \in \{0,1\}^{n\times n}$ be the adjacency matrix – a square matrix whose elements indicate whether pairs of vertices are adjacent or not in the graph. The perturbation is initially defined similarly to Lucic et al. (2022) , having the CF generated matrix $\hat{A} = P \odot A$, with $P = \mathbf{1}_{n\times n}$, a binary perturbation matrix full of ones, with same dimensionality of A; and $\odot$ denoting element-wise (Hadamard) product (a real-valued $\bar{P}$ is first generated, then a threshold with a entry-wise sigmoid transformation $\sigma(\cdot)$ is performed to obtain a binary P, either discarding or retaining edges: $\mathcal{T}_{0.5}(\bar{P}) = \mathbb{1}\{\sigma(\bar{P}) \geq 0.5\}$). To include self message passing, self-loops are added in the form of the identity matrix during CF search. X is defined as the node feature matrix (so that $G = (A, X)$), W as the weight matrix and $\bar{D}$ as the degree matrix, a diagonal matrix containing the number of edges attached to each vertex, based on $P \odot A + I$. The original goal is to only remove edges zeroing out entries in the adjacency matrix, so find P (which acts as a gate) that minimally perturbs A and use it to compute $\hat{A}$. Hence, the original counterfactual generating model, parameterized by P, is:

$$\mathcal{E}(A, X, W; P) = \text{softmax}[\bar{D}^{-\frac{1}{2}}(P \odot A + I)\bar{D}^{-\frac{1}{2}}XW], \tag{3.2.0.1}$$

note that here, differently than Lucic et al. (2022), the CF search is not performed on a subgraph neighbourhood of a node, but rather on the whole graph. For undirected graphs, only the upper triangular part of $\bar{P}$ is parametrized and it is symmetrized at each step to obtain $\hat{A}$. This ensures that adjacency perturbations remain symmetric.

Edge-masking with Free Insertions The original formulation of the framework has some downsides when it comes to updating P: when the oracle prediction is obtained with the associated loss, and new edges need to be added. Essentially, gradients for adjacency and perturbation matrix entries are computed and flow back, but edges not present in A ($A_{ij} = 0$) cause respective gradients of P to be zeroed out:

$$\frac{\partial \mathcal{L}}{\partial P_{ij}} = \frac{\partial \mathcal{L}}{\partial \hat{A}_{ij}} \cdot \frac{\partial \hat{A}_{ij}}{\partial P_{ij}};\ \hat{A}_{ij} = A_{ij} \cdot P_{ij}, \frac{\partial \hat{A}_{ij}}{\partial P_{ij}} = A_{ij} \Rightarrow \frac{\partial \mathcal{L}}{\partial P_{ij}} = \frac{\partial \mathcal{L}}{\partial \hat{A}_{ij}} \cdot A_{ij} \tag{3.2.0.2}$$

– *As a results edges not initially present in A cannot be inserted during CF search by design choice, and gradients of missing edges indicating the direction of optimal*

change, even if are back-propagated until P, will not lead to any update to it, thus introducing confusion for other updates that may be co-adapted/correlated to them.

To enable the inclusion of any edge, the roles of A and P are swapped: initializing $\bar{P} \leftarrow A$, hence making it store the original adjacencies, but making it able to be updated by the iterative gradients; and making $\bar{A} = \mathbf{1}_{n \times n}$ be a matrix full of ones, fully connecting the graph. It is possible to account for the original missing edges by removing them from $\bar{P}$ (setting their values to zero), now $\hat{A}$ is computed as $\hat{A} = \bar{P} \odot \bar{A}$ plus self loops. By this swap, the explainer gets the possibility to freely add edges by updating the whole graph accordingly to the loss. $\bar{A}$ now becomes a (redundant) matrix of ones whereas $\bar{P}$ stores the graph connections, ready to be perturbed. It is then added to P a small Gaussian noise ($std : \sigma = 0.1; \mu = 0$) to break the symmetry in the gradient updates.

Table 3.2. XPlore' steps to perturb edges and node features.

Step	Action
1	Initialize $\bar{P} \leftarrow A + \mathcal{N}(0, 0.1^2)$ and optionally add Γ.
2	Initialize $\bar{N} \leftarrow \mathbf{1}_{n \times f}$
3	Compute $\hat{A} = max(\mathcal{T}_{0.5}(\bar{P}), I)$
4	Compute contribution of $\hat{A}$ and $\bar{N}$ to L_{pred} through the oracle's prediction (Equations (3.2.0.5) and (3.2.0.6)).
5	Update $\bar{P}$ and $\bar{N}$ via gradient descent.

Moreover, a probability weight is assigned to missing edges of $\bar{P}$: defining a matrix $\Gamma \in [-1, 1]^{n \times n}$. Γ is added to $\bar{P}$ at initialization: $\bar{P} \leftarrow \bar{P} + \Gamma$. The idea is to manually adjust each $\gamma_{ij} \in [-1, 1]$ so that by adding Γ it is possible to induce edge presence or absence, therefore Γ behaves like a mask. For example, some may be interested in a counterfactual explanation that has specific connections: it is possible to directly encode them in Γ. Note that P and A were swapped to be coherent with the original formulation: as A now consists of a matrix of ones, it is redundant for updating $\bar{P}$ and can be simply omitted. Now, the explainer is as in Equation (3.2.0.3):

$$\mathcal{E}(X, W; \bar{P}) = \mathcal{S}[\bar{D}^{-1/2} \hat{A} \bar{D}^{-1/2} X W], \tag{3.2.0.3}$$

where $\mathcal{S}$ is either the element-wise sigmoid or the softmax function, depending on whether the task is binary or multi-class; and $\bar{D}$ is based on $\hat{A}$, slightly different from Equation (3.2.0.1), ensuring that values in the diagonal are ones, specifically when self-loop are already present – due to $\hat{A}$ having no more the diagonal entries fixed at zero as it depends on $\bar{P}$ for storing perturbed adjacencies and not on A any more.

Node Features Perturbation. Similarly to what has been done for the edge perturbation matrix $\bar{P}$, it is possible to introduce a perturbation matrix $\bar{N} = \mathbf{1}_{n \times f}$, f being the number of node features, to perform continuous perturbation on the node features. Two different mechanisms can be applied (Algorithm 1): i.e., **(1)** gate the features values with a sigmoid, by applying the sigmoid and then the threshold to have a binary mask, and **(2)** let them freely change, updating them proportionally to the gradient information of the loss. In results, these mechanisms added on top of the edges addition, are defined as XPlore w/ gating and XPlore w/ freedom. Hence, the final formulation of the proposed explainer considering all perturbations is:

$$N = \mathcal{T}_{0.5}(\bar{N}) \text{ if } gate \text{ else } \bar{N} \tag{3.2.0.4}$$

$$\mathcal{E}(X, W; \bar{P}, \bar{N}) = \mathcal{S}[\bar{D}^{-1/2}\hat{A}\bar{D}^{-1/2}(X \odot N)W]. \quad (3.2.0.5)$$

Loss function optimization. $\bar{P}$ is generated by minimizing Equation (3.1.0.2), defining the prediction loss L_{pred} as in Equation (3.2.0.6): for L_{logits} the cross-entropy (CE) loss for single-label classification tasks is employed, encompassing binary and multi-class scenarios, and the binary cross-entropy with logits loss for multi-label classification tasks:

$$L_{\text{pred}}(G, G' \mid \Phi) = -\mathbf{1}[\Phi(G) = \Phi(G')] \cdot L_{\text{logits}}(\Phi_{\ell-1}(G), \Phi_{\ell-1}(G')), \quad (3.2.0.6)$$

where $\Phi_{\ell-1}(G)$ outputs the logits given the input G, and ℓ is the number of layers. G' is $\mathcal{E}(G)$. The L_{dist} in Equation (3.1.0.2) is the element wise distance between G and G', corresponding to the sums of the two L_p-norms of A and A', and X and X':

$$L_{\text{dist}}(G, G') = \|A - A'\| + \|X - X'\|. \quad (3.2.0.7)$$

3.3 Algorithmic Implementation

Algorithm 1 XPlore.

Input graph $G = (A, X)$, trained oracle Φ, explainer $\mathcal{E}$, loss function L, learning rate α, number of iterations K, matrix Γ, node features *gate* flag, distance function d.
$y \leftarrow \Phi(G)$ {▶ Oracle initial prediction}
$\bar{P} \leftarrow A + \mathcal{N}(0, 0.01) + \Gamma$ {▶ Noise + mask}
$\bar{N} \leftarrow \mathbf{1}_{n \times f}$ {▶ Node perturbation matrix}
$G^* = [\,]$
for K iterations **do**
  $G', G^* =$ `GET_CF_EXAMPLE()`
  $L \leftarrow L(G, G', \Phi)$ {▶ Eqns. 3.2.0.6-3.2.0.7}
  $\bar{P} \leftarrow \bar{P} - \alpha \nabla_{\bar{P}} L$ {▶ Update $\bar{P}$}
  $\bar{N} \leftarrow \bar{N} - \alpha \nabla_{\bar{N}} L$ {▶ Update $\bar{N}$}
return G^*

func `GET_CF_EXAMPLE()`
  $\hat{A} \leftarrow max(\mathcal{T}_{0.5}(\bar{P}), I)$ {▶ Perturbed adj.}
  if *gate* **then** {▶ Gate node features}
    $N \leftarrow \mathcal{T}_{0.5}(\sigma(\bar{N}))$
  else {▶ Freely perturb node features}
    $N \leftarrow \bar{N}$
  $N \leftarrow N \odot X$
  $G'_{cand} \leftarrow (\hat{A}, N)$
  if $\Phi(G) \neq \Phi(G'_{cand})$ **then**
    $G' \leftarrow G'_{cand}$
    if not G^* **then**
      $G^* \leftarrow G'$ {▶ First counterfactual}
    else if $d(G, G') \leq d(G, G^*)$ **then**
      $G^* \leftarrow G'$ {▶ Closer counterfactual}
  return G', G^*

The details of the proposed method XPlore[1] are summarized in Algorithm 1. Given a graph in the test set G, its prediction is obtained from the GNN oracle Φ. $\bar{P}$ is initialized as the adjacency matrix A and summed with Γ, giving missing edges a probability value of $\gamma_{i,j} \in [-1, 1]$. $\bar{N}$ is assigned to a matrix of ones. XPlore is run for K iterations (4.3.2), at each one, an optimization step is performed to find a valid counterfactual and improve it, there is no stopping criterion as the same CF is iteratively modified and potentially improved across successive iterations, as conceived by Lucic et al. (2022). Equation (3.2.0.5) is used to find a counterfactual example. P is computed by applying a sigmoid transformation on $\bar{P}$ and then a threshold to obtain a binary matrix.

Although hard threshold $\mathcal{T}$ is applied in forward pass, gradients are back-propagated through this non-differentiable step via the straight-through estimator (STE) (Bengio et al.,

[1] https://github.com/ric-desa/GRETEL.git

2013), which treats thresholding as identity during backpropagation (cf. 3.5.2). As stated earlier, the element wise multiplication of P with A to get A' could simply be omitted, but it is here retained for consistency with the previous version of the algorithm. Self-loops are added by summing the identity matrix; this is done to let node representation updates to include self-representations from previous layers. According to the node perturbation mechanism, node features are either gated or simply multiplied by the node features perturbation matrix $\bar{N}$. The candidate graph produced G'_{cand} is fed to the GNN oracle Φ. If the output prediction is different from the initial node, a valid counterfactual is found. The closer counterfactual is returned as the optimal CF example G^* after K iterations upon success. $\bar{P}$ and $\bar{N}$ are updated based on the loss, which is calculated according to Equations (3.1.0.2) and (3.2.0.5) to (3.2.0.7). This setting allows to perform updates freely by perturbing edges and node features and adding missing ones. The optimal explanation is retrieved as $\Delta^*_G = G - G^*$. The algorithm is linear in the number of edges and in the multiplication of features with number of nodes (i.e. $O(|E|\, d + n\, d\, f)$, see 3.5.7).

3.4 Oracle Models

This section describes the state-of-the-art oracles implemented for the empirical experimental evaluation.

3.4.1 Full-Graph Reformulation

Perturbation-based counterfactual generation in GNNs requires gradients not only for existing edges but also for *missing* ones. However, standard message-passing layers (e.g., GCNConv, GATConv) operate exclusively on the edges explicitly provided through `edge_index`. As a consequence, no computation, and therefore no gradient, flows through non-existent edges. This makes it impossible to estimate how adding or removing an edge would affect the model's prediction.

To overcome this limitation, each input graph was reformulated as a **full graph**, where *all possible node pairs* (i, j) are included in the edge set. Connectivity is no longer encoded by the presence or absence of an edge, but instead by its **edge weight**:

- existing edges keep their original weight;
- missing edges are assigned weight zero.

This dense representation allows the GNN to treat every potential edge as a differentiable parameter. Message passing becomes a function of the full adjacency matrix, and the computational graph now includes the contribution of all edges. As a result, gradients correctly propagate to missing edges, enabling the explainer to assess the effect of hypothetical edge insertions and deletions.

For the model to correctly reflect how each (real or missing) edge influences the prediction, the oracle must *use the edge weights during its convolution or aggregation operations*. Otherwise, even if connectivity is encoded in the weights, the GNN would ignore them, and no meaningful sensitivity or gradient signal could be obtained. This requirement is model-agnostic: whatever architecture is used, its message-passing mechanism must incorporate the provided edge weights for the full-graph reformulation to be effective.

Although this representation increases computational cost, it removes the structural bias introduced by sparse message passing and ensures gradient availability for the entire edge space, an essential requirement for perturbation-based counterfactual explanations.

3.4.2 Graph Edge Convolution Network

The baseline GCN layer introduced by Kipf and Welling (2017) is given in section 2.1.4. Recall the standard layer computation (eq. (2.1.4.1)):

$$H^{(l+1)} = \sigma\Big(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\Big). \tag{2.1.4.1}$$

where, $\tilde{A} = A + I_N$ is the adjacency matrix with added self-connections. $\tilde{D}$ is the degree matrix with $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. No edge features or edge weights participate in the computation.

To account for edge weights influence in the prediction, the classical architecture is extended by incorporating explicit edge weights into the message passing process and by recording per-edge contributions for gradient computation. The extended architecture is thus named Graph Edge Convolution Network (GECN). For each edge $(j \to i)$, the message is:

$$m_{(j\to i)} = e_{ij}W^{(l)}h_j, \tag{3.4.2.1}$$

where:

- e_{ij} is the edge weight,
- h_j is the source node feature vector,
- $W^{(l)}$ is the layer weight matrix.

The aggregated updates become:

$$h_i^{(l+1)} = \sum_{j\in\mathcal{N}(i)} \alpha_{ij}e_{ij}W^{(j)}h_j, \tag{3.4.2.2}$$

with the usual GCN normalization factor: $\alpha_{ij} = \left(\tilde{D}_{ii}\tilde{D}_{jj}\right)^{-1/2}$. Unlike the original GCN formulation, which aggregates normalized transformed node features without considering edge attributes, this variant modulates each message by the corresponding edge weight and stores the resulting edge-wise messages. GCNE stores the raw edge-scaled messages for every $(j \to i)$ as

$$M_{ij}^{(l)} = e_{ij} \cdot h_j. \tag{3.4.2.3}$$

By retaining this tensor during computation the gradients $\frac{\partial\mathcal{L}}{\partial e_{ij}}$ can be computed during backpropagation and therefore by XPlore as the direct gradient flow through edge weights is enabled. Additionally, a residual connection is further added to stabilize training across varying layer widths:

$$H^{(l+1)} = \hat{H}^{(l+1)} + Rh^{(l)}, \tag{3.4.2.4}$$

where

$$R = \begin{cases} I, & \text{if in} = \text{out}, \\ Linear(in, out), & \text{otherwise.} \end{cases} \tag{3.4.2.5}$$

3.4.3 Graph Edge Attention Network

The baseline GAT layer introduced by Velickovic et al. (2017) is given in section 2.1.5. Recall the standard computation of attention coefficients on edge $(j \to i)$ (eq. (2.1.5.9)) as:

$$\alpha_{ij} = \frac{\exp\Big(\text{LeakyReLU}\Big(\vec{a}^T[\mathbf{W}\vec{h}_i||\mathbf{W}\vec{h}_j]\Big)\Big)}{\sum_{k\in\mathcal{N}_i}\exp\Big(\text{LeakyReLU}\Big(\vec{a}^T[\mathbf{W}\vec{h}_i||\mathbf{W}\vec{h}_k]\Big)\Big)}, \tag{2.1.5.9}$$

the node update (eq. (2.1.5.10)), where no edge weighting or edge-level masking is present, is:

$$\vec{h}_i^{(l+1)} = \sigma\Bigg(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\mathbf{W}\vec{h}_j\Bigg). \tag{2.1.5.10}$$

The Graph Edge Attention Network (GEAT) layer extends the original GAT architecture by introducing an explicit **edge-weight masking mechanism** inside the **attention** computation. Instead of computing attention solely from node features, the attention logits are multiplied by a learned or externally provided edge weight e_{ij}, which directly modulates the importance of each edge during message passing. This modification allows gradients to back-propagate through the attention scores to each edge, enabling optimization of edge weights for XPlore's counterfactual graph editing. Additional normalization by the source-node degree stabilizes training. Apart from this masking mechanism, the rest of the GAT pipeline (multi-head attention, softmax normalization, dropout) remains unchanged.

Before softmax normalization, the modified attention coefficient is:

$$\tilde{\alpha}_{ij} = e_{ij}\,\text{LeakyReLU}\Big(a^T[\mathbf{W}\vec{h}_i||\mathbf{W}\vec{h}_j]\Big) \tag{3.4.3.1}$$

For node-pair attention logits from both directions, the masked version is:

$$\tilde{\alpha}_{ij} = e_{ij}\,\text{LeakyReLU}\Big(\alpha^{(j)} + \alpha^{(i)}\Big). \tag{3.4.3.2}$$

To stabilize message scaling, the masked logits are normalized by the source-degree:

$$\tilde{\alpha} \leftarrow= \frac{\tilde{\alpha}}{\deg(j)+\epsilon} \tag{3.4.3.3}$$

The final normalized attention weights become:

$$\alpha_{ij} = \text{softmax}_{j\in\mathcal{N}(i)}(\alpha_{ij}). \tag{3.4.3.4}$$

This makes the attention mechanism fully differentiable with respect to each edge weight e_{ij}. The node update (per head) is:

$$h_i^{(l+1)} = \sum_{j\in\mathcal{N}_i}\alpha_{ij}Wh_j. \tag{3.4.3.5}$$

with multi-head concatenation identical to the original GAT.

3.4.4 Graph Isomorphism Network Variants

The baseline GIN introduced by Xu et al. (2018) is given in section 2.1.6. Recall the standard GIN update (eq. (2.1.6.1)), in which no edge features or attention weighting are used, as:

$$h_i^{(l+1)} = \text{MLP}^{(k)}\left(\left(1+\epsilon^{(l)}\right)\cdot h_i^{(l)} + \sum_{j\in\mathcal{N}()} h_j^{(l)}\right). \tag{2.1.6.1}$$

Two different GIN variants are implemented:

- **Attention-GIN** (A-GIN): conceptually similar to the GEAT previously introduced section 3.4.3, it reuses the attention formulation.
- **Gated Multi-Head GIN** (G-GIN): independent formulation, designed for dense graphs.

Attention Graph Isomorphism Network

A-GIN extends the Graph Isomorphism Network by introducing an attention mechanism that scores each neighbour using both transformed node features (Wh_i, Wh_j) and edge weights ($\phi(e_{ij})$). The attention vector takes the concatenation:

$$z_{ij} = [Wh_i^{(l)} || Wh_j^{(l)} || \phi(e_{ij})], \tag{3.4.4.1}$$

and produces raw multi-head scores:

$$\alpha_{ij}^{(k)} = \text{LeakyReLU}(\mathbf{a}_k^{\mathsf{T}} z_{ij}). \tag{3.4.4.2}$$

The softmax normalization across neighbours per node and head is:

$$\alpha_{ij}^{(k)} = \text{softmax}_{j\in\mathcal{N}(i)}(\alpha_{ij}^{(k)}). \tag{3.4.4.3}$$

Messages are reweighted by these attention coefficients and aggregated across heads before passing through the GIN MLP. Each head computes:

$$m_{ij}^{(k)} = \alpha_{ij}^{(k)}(h_j^{(l)} + \phi(e_{ij})), \tag{3.4.4.4}$$

And AGIN averages over heads:

$$m_i = \frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}(i)} m_{ij}^{(k)}. \tag{3.4.4.5}$$

with the final update being:

$$h_i^{l+1} = \text{MLP}(m_i) + \text{Res}(h_i^{(l)}). \tag{3.4.4.6}$$

This allows AGIN to perform structure-aware, edge-dependent neighbor selection, unlike the sum-only aggregation in the original GIN.

Gated Multi-Head Graph Isomorphism Network

This architecture (GGIN) augments the GIN using **multiplicative edge gating** instead of attention, resulting in a well-suited paradigm for dense graphs with many zero-weight edges.

For each head, a learned scalar gate $g_{ij}^{(k)} = \sigma(w_k^{\mathsf{T}} \phi(e_{ij}))$, where $w_k \in \mathbb{R}^{d_e}$ is a head-specific linear filter over edge weights, modulates the contribution of each neighbour j. Each head transmits:

$$m_{ij}^{(k)} = g_{ij}^{(k)} h_j^{(j)} \tag{3.4.4.7}$$

Messages are sum-aggregated in the GIN fashion eq. (3.4.4.8) and averaged across heads eq. (3.4.4.9) before passing through the MLP and residual connection eq. (3.4.4.10):

$$m_i^{(k)} = \sum_{j \in \mathcal{N}(i)} m_{ij}^{(k)}, \tag{3.4.4.8}$$

$$m_i = \frac{1}{K} \sum_{k=1}^{K} m_i^{(k)}, \tag{3.4.4.9}$$

$$h_i^{(l+1)} = \mathrm{MLP}(m_i) + \mathrm{Res}(h_i^{(l)}). \tag{3.4.4.10}$$

Unlike A-GIN or GAT-based models, G-GIN performs no softmax normalization, making it particularly suitable for dense graphs where many edges must be softly suppressed.

3.4.5 Edge Gated Recurrent Unit

The layer extends the classical Gated Graph Neural Network update (Li et al., 2015) given in section 2.1.7 by injecting edge information directly into both the message-passing and gating mechanisms. Messages are produced from concatenated neighbour and edge weights, then modulated by a learned edge-gate:

$$g_{ij} = \sigma(\psi(e_{ij})), \quad g_{ij} \in (0, 1), \tag{3.4.5.1}$$

where $\psi(\cdot)$ is a MLP applied to the edge weights e_{ij}. The raw message uses concatenated neighbour and edge weights:

$$m_{ij} = \phi\Big([h_j^{(l)} || e_{ij}]\Big), \tag{3.4.5.2}$$

$\phi(\cdot)$ being an MLP. The final edge-gated message is: $\tilde{m}_{ij} = g_{ij} m_{ij}$. Messages are summed as in GIN / GGNN:

$$m_i = \sum_{j \in \mathcal{N}(i)} \tilde{m}_{ij}. \tag{3.4.5.3}$$

Aggregated messages are used as the input to a GRUCell, which performs a recurrent update of node states and incorporates a residual connection to handle dimensionality changes.

$$h_i^{(l+1)} = \mathrm{GRU}\Big(m_i, \mathrm{Res}(h_i^{(l)})\Big), \tag{3.4.5.4}$$

where $\mathrm{Res}(\cdot)$ is, as in previous architectures, either identity or a linear projection when dimensions differ. The GRU update expands to:

$$z_i = \sigma\Big(W_z m_i + U_z h_i^{(l)}\Big), \quad (3.4.5.5)$$

$$r_i = \sigma\Big(W_r m_i + U_r h_i^{(l)}\Big), \quad (3.4.5.6)$$

$$\tilde{h}_i = \tanh\Big(W_h m_i + U_h\Big(r_i \odot \mathbf{h_i^{(l)}}\Big)\Big), \quad (3.4.5.7)$$

$$\mathbf{h}_i^{(l+1)} = (1 - \mathbf{z}_i) \odot \mathbf{h}_i^{(l)} + \mathbf{z}_i \odot \tilde{h}_i, \quad (3.4.5.8)$$

with gating coefficients z_i, r_i controlling information flow. This architecture allows E-GRU to selectively amplify or suppress edges and accumulate information over layers in a stable, gated fashion.

3.4.6 Edge Principal Neighbourhood Aggregation

The Principal Neighbourhood Aggregation operator (Corso et al., 2020) introduced in section 2.1.10, is extended by incorporating edge-refined messages and edge-conditioned updates, enabling the model to track and modulate message strengths based on edge weights. The resulting architecture is named E-PNA.

Each message is created from neighbour features and refined edge embeddings; given an edge $(j \to i)$ with edge weight (or feature) e_{ij}, edge embeddings are computed as $\tilde{e}_{ij} = \psi(e_{ij})$, where $\psi(\cdot)$ is, as in previous architectures, a MLP applied to the edge weights e_{ij}. Messages concatenate neighbour features with refined edge features:

$$m_{ij} = \phi\Big([h_j^{(l)} || \tilde{e}_{ij}]\Big), \quad (3.4.6.1)$$

with $\phi(\cdot)$ being an MLP. A small additive correction from the edge refinement $\tilde{m}_{ij} = m_{ij} + \tilde{e}_{ij}$ is applied to enhance the use edge weights and obtain a better gradient signal. For each target node i, the full PNA suite of aggregator are computed over incoming messages:

$$\text{mean}_i = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \tilde{m}_{ij}, \quad (3.4.6.2)$$

$$\max_i = \max_{j \in \mathcal{N}(i)} \tilde{m}_{ij}, \quad (3.4.6.3)$$

$$\min_i = \min_{j \in \mathcal{N}(i)} \tilde{m}_{ij}, \quad (3.4.6.4)$$

$$\text{std}_i = \sqrt{\frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} ||\tilde{m}_{ij} - \text{mean}_i||^2 + \epsilon}. \quad (3.4.6.5)$$

The aggregated feature vector is the concatenation:

$$A_i = [\text{mean}_i || \max_i || \min_i || \text{std}_i] \quad (3.4.6.6)$$

The resulting statistics are modulated by degree-based scalers before being projected to the latent dimension. Let $d_i = |\mathcal{N}(i)|$ be the node degree. The PNA scaling functions are then applied:

- Identity: $S_i d(A_i) = A_i$,
- Amplification: $S_{amp}(A_i) = A_i \cdot \log(d_i + 1)$,
- Attenuation: $S_{att}(A_i) = A_i / \log(d_i + 2)$,

yielding the degree-aware aggregated output: $\hat{A}_i = S(A_i)$. Finally, the aggregated vector is projected back to the latent dimension:

$$h_i^{(l+1)} = W_{comb}\, \hat{A}_i + \text{Res}(h_i^{(l)}), \quad (3.4.6.7)$$

where Res(·) is, as in previous architectures, either identity or a linear projection when dimensions differ. This design injects edge-sensitivity into all stages of PNA, message construction, neighbourhood statistics, and update, yielding a strictly more expressive variant that tracks edge-level influence across layers.

3.4.7 Edge Graph-GPS

The GPS Graph Transformer (Rampášek et al., 2022) introduced in section 2.1.9 is extended to replace the standard local message-passing block with an edge-aware GNN, enabling the GPS framework to incorporate edge information directly into the local structural updates. E-GPS is a hybrid message-passing layer combining local, edge-aware structural reasoning with global graph-wide attention, feed-forward refinement, residual connections and layer normalization.

The local block injects edge-weight information into node updates, enhancing relational sensitivity, by applying an edge-sensitive GNN operator $\mathcal{M}$ (here an edge-weighted GCN section 3.4.2):

$$h_i^{local} = \mathcal{M}(h_i^{(l)}, \mathcal{N}(i), e_{ij}). \tag{3.4.7.1}$$

In the case of an edge-weighted GCN:

$$h_i^{local} = \sum_{j \in \mathcal{N}(i)} \frac{w_{ij}}{\sqrt{d_i d_j}} W h_j^{(l)}. \tag{3.4.7.2}$$

A residual update with layer normalization (section 2.1.12):

$$H^{(1)} = \text{LayerNorm}\Big(H^{(l)} + H^{local}\Big). \tag{3.4.7.3}$$

The introduction of explicit edge weights w_{ij} distinguishes EGPS from the default GPS layer, whose local block is not inherently edge-aware. The global transformer then propagates contextual information across distant nodes, and a feed-forward block refines node embeddings. Nodes interact through full-graph multi-head attention:

$$H^{gloabl} = \text{MultiHeadAttention}(H^{(1)}, H^{(1)}, H^{(1)}), \tag{3.4.7.4}$$

with residual connection and normalization:

$$H^{(2)} = \text{LayerNorm}(H^{(1)} + H^{global}) \tag{3.4.7.5}$$

This component captures long-range dependencies complementary to the local GNN. Finally, a position-wise feed-forward network further processes each node:

$$H^{(l+1)} = \text{LayerNorm}\Big(H^{(2)} + \text{FFN}(H^{(2)})\Big). \tag{3.4.7.6}$$

Overall, E-GPS preserves the modular GPS design while enabling edge-level influence throughout the local processing stage.

3.4.8 Diffusion Graph Edge Neural Network

A Graph Edge Convolution Network section 3.4.2 is extended to operate inside a denoising diffusion probabilistic model on graphs section 2.1.11. This architecture (GECN-DDPM) performs graph classification through diffusion-based distributional modeling of the label. Each node is augmented with a time embedding and a diffused label embedding, while edges are treated as continuous variables subject to diffusion, yielding a fully connected, time-varying weighted adjacency in each step. The GECN is modified to condition on: noisy time step t, noisy labels y_t, fully-connected diffused edge weights w_t, classifier-free masked conditioning. The GECN thus becomes an edge-aware neural operator conditioned on the diffusion time step. Each node receives a concatenation of:

$$x_t^{(t)} = [h_i || y_t^{(g(i))} || \tau(t)], \tag{3.4.8.1}$$

where h_i is the original node feature, $y_t^{(g(i))}$ is the diffused label embedding for the graph containing node i, $\tau(t) \in \mathbb{R}^{d_t}$ is the sinusoidal time embedding. Thus, the GECN input dimension is $d_{in} = d_h + d_{label} + d_t$. Each GECN layer is an edge weighted GCN of the form:

$$h_i^{l+1} = \sigma\Big(\sum_{j \in \mathcal{N}(i)} \frac{w_{ij}}{\sqrt{d_i d_j}} W h_j^{(l)} + R h_i^{(l)} \Big), \tag{3.4.8.2}$$

where w_{ij} is the diffused edge weight at time t, R is either the residual projection or identity if dimensions match (as in previous architectures), σ is the ReLU. It is possible to inject diffusion-independent structural feature such as network topology to hone the representational power. The DDPM forward corruption if followed:

$$*_t = \sqrt{\bar{\alpha}_t}\, *_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon_*, \quad * \in \{x, w, y\} \tag{3.4.8.3}$$

The model predicts all three noise components:

$$\epsilon_x^{\text{pred}} = fx(h^L),\ \epsilon_w^{\text{pred}} = f_W([h_i^{(L)} || h_j^{(L)}]),\ \epsilon_y^{\text{pred}} = f_Y(u_g). \tag{3.4.8.4}$$

With Loss comprising the distance Mean Square Error (MSE) term, the classification Cross Entropy (CE):

$$\mathcal{L}_{diff} = \lambda_{cls}\text{CrossEntropy}(\text{CLS}(u_g), y_0) + \sum_{*\in\{x,w,y\}} \lambda_* ||\epsilon_*^{pred} - \epsilon_*||^2 \tag{3.4.8.5}$$

Finally, a classifier-free guidance (Ho and Salimans, 2022) is adopted to stabilize training and dstribution learning:

$$\tilde{x} = \begin{cases} \text{original graph } x_0, w_0, y_0 & \text{with probability } P_{clean} \\ \text{noised graph } x_t, w_t, y_t & \text{with probability } 1 - P_{clean} \end{cases} \tag{3.4.8.6}$$

3.5 Theoretical Foundations of Gradient-Guided Counterfactual Perturbations

This section is meant to give a mathematical justification for why our gradient-based procedure converges and finds small (ℓ_1-minimal) local perturbations to the graph and features.

Section Roadmap. This section first establishes convergence of projected gradient descent (section 3.5.1), then handles non-smooth thresholding (section 3.5.2), proves the ℓ_1-minimality bound (section 3.5.3), derives the edge-insertion/deletion condition (section 3.5.4), shows that the prediction loss is L-smooth (section 3.5.5), chooses sparsity–accuracy trade-offs (section 3.5.6), analyzes computational complexity (section 3.5.7), extends to weighted and directed graphs (section 3.5.8), and finally discusses search-space expressivity (section 3.5.9).

?? describes a soft, differentiable objective $L(\cdot)$ (Equation (3.1.0.2)) that trades off between (a) changing the model's prediction and (b) paying an ℓ_p "cost" for every edit. Ir remains to be shown that if this objective is optimized with a natural algorithm, a valid counterfactual that also doesn't make too-large edits if found. Hence, here it is justified why the joint gradient-based optimization over edge masks P and feature perturbations N (cf. Algorithm 1 and eqs. (3.1.0.2) and (3.2.0.5)) converges to a meaningful counterfactual and yields nearly ℓ_1-minimal changes.

3.5.1 Convergence to Local Minimizers

It is possible to stack all continuous perturbation variables into a single vector $\theta = (\mathrm{vec}(P), \mathrm{vec}(N))$, and write the soft objective Equation (3.1.0.2) (later for brevity we use $L(\theta) = L(G;\theta)$)

$$L(G;\theta) = L_{\text{pred}}\big(\Phi(G), E(G;\theta)\big) + \beta L_{\text{dist}}(\theta), \tag{3.5.1.1}$$

where

$$L_{\text{pred}}\big(\Phi,\, E(G;\theta)\big) = \text{CE}\big(\Phi(G),\, \Phi\big(E(G;\theta)\big)\big) \tag{3.5.1.2}$$

for multi-class classification or

$$L_{\text{pred}}\big(\Phi,\, E(G;\theta)\big) = \text{BCE-Logits}\big(\Phi(G),\, \Phi\big(E(G;\theta)\big)\big) \tag{3.5.1.3}$$

for multi-label classification, and

$$L_{\text{dist}}(\theta) = \big\|A - \widehat{A}(\theta)\big\|_1 \;+\; \big\|X - X'(\theta)\big\|_1, \tag{3.5.1.4}$$

as in Equations (3.1.0.2), (3.2.0.6) and (3.2.0.7). To reflect the proposed methodology, it is possible to optimize by projected gradient descent, as P is constrained to $[0,1]$, while leaving N unconstrained:

$$\theta^{t+1} \leftarrow \begin{cases} P^{t+1} = \Pi_{[0,1]^{n^2}}\left(P^t - \eta\, \nabla_P L(P^t, N^t)\right), \\ N^{t+1} = N^t - \eta\, \nabla_N L(P^t, N^t) \quad \text{(no projection)}. \end{cases} \tag{3.5.1.5}$$

where $\Theta = [0,1]^{n\times n} \times \mathbb{R}^{n\times f}$, closed and convex parameter domain, enforces $P \in [0,1]^{n\times n}$ and $N \in \mathbb{R}^{n\times f}$.

Assumption 3.5.1. *Assume the loss $L : \Theta \to \mathbb{R}$ satisfies:*

1. *Differentiability: L is smooth plus convex-nonsmooth (i.e. L_{pred} is C_1, and L_{dist} is proper, closed, convex, and can be smoothed by a fixed smooth approximation $L_{dist,\epsilon}(\theta) = \lambda \sum_i \sqrt{\theta_i^2 + \epsilon^2}$ for a fixed ϵ).*
2. *L-smoothness: $\|\nabla L(x) - \nabla L(y)\| \le L\|x - y\|$ for all $x, y \in \Theta$. (This is the smoothness constant L).*

3. Coercivity: $\{ x \in \Theta : L(x) \leq c\}$ *is bounded for any* $c \in \mathbb{R}$.

Theorem 3.5.1 (Convergence of PGD)**.** *Under Assumption 3.5.1, if the step-size satisfies* $\eta \leq 1/L$*, then the iterates* $\theta_{t+1} = \Pi_\Theta(\theta_t - \eta \nabla L(\theta_t))$ *obey*

$$\sum_{t=0}^{\infty} \|\theta_{t+1} - \theta_t\|^2 < \infty \quad \Longleftrightarrow \quad \lim_{t\to\infty} ||\theta^{t+1} - \theta^t|| = 0, \tag{3.5.1.6}$$

and every limit point is Clarke-stationary.

and any limit point $\theta^* = (P^*, N^*)$ fulfils the first-order optimality condition

$$0 \in \begin{pmatrix} \nabla_P L(P^*, N^*) \\ \nabla_N L(P^*, N^*) \end{pmatrix} + \begin{pmatrix} NC_{[0,1]^{n^2}}(P^*) \\ \{0\} \end{pmatrix}, \tag{3.5.1.7}$$

where NC is the normal-cone block encoding our constraint on P: $v \in NC_{[0,1]^{n^2}}(P^*) \Longleftrightarrow \langle v,\ P - P^* \rangle \leq 0 \quad \forall P \in [0,1]^{n^2}$.
This means θ^* is a stationary point of the soft objective (stationarity in N (zero gradient) and projected-stationarity in P. Because L_{dist} has kinks at integer p entries (ℓ_1), we actually converge to a Clarke-stationary point. Clarke-stationary point means PGD still converges to a point where no small perturbation – even through the hard threshold – can locally decrease the loss. All standard PGD convergence guarantees for convex–nonsmooth+smooth composite objectives (e.g. see Bertsekas (1997)) carry over.

Proof (derived from Bertsekas (1997) (ch. 2.3))

Θ is closed and convex. Moreover, under our loss design the level sets $\{\theta | L(\theta) \leq L(\theta_0)\}$ are bounded (or equivalently L is coercive), ensuring the standard projected-gradient-descent convergence arguments carry through.
A function $L : \mathbb{R}^d \to \mathbb{R}$ is said to be L-smooth if its gradient is Lipschitz continuous with constant L. Concretely, for all $\theta, \theta' \in \mathbb{R}^d$,

$$||\nabla L(\theta') - \nabla L(\theta)|| \leq L||\theta' - \theta||.$$

Equivalently, this implies the Taylor-remainder bound

$$L(\theta') \leq L(\theta) + \nabla L(\theta)^\top (\theta' - \theta) + \frac{L}{2}||\theta' - \theta||^2,$$

for the present purposes, L is L-smooth over the convex set Θ ($\theta, \theta' \in \Theta$). This means that it is possible to upper-bound the change in L by its first-order (linear) term plus a quadratic "error" that scale with $||\theta' - \theta||^2$.

Now set $\theta' = \Pi_\Theta(\theta - \eta \nabla L(\theta))$ as the PGD update, that is: start at θ, step "downhill" by $\eta \nabla L(\theta)$, then project back into the box Θ).
We call

$$d(\theta) = \theta - \Pi_\Theta(\theta - \eta \nabla L(\theta))$$

the projected gradient (or "proximal residual"). It is the step we actually take, equivalently one can view $\theta' - \theta = -d(\theta)$ as the update direction. When $||d(\theta)||$ goes to zero, then θ is a constrained stationary point, where "gradient+projection" induce no change.

Now, plugging $\theta' = \Pi_\Theta(\theta - \eta\nabla L(\theta))$ into the smoothness bound gives two pieces:
1. Linear piece: $\nabla L(\theta)^\top(\theta' - \theta) = -\nabla L(\theta)^\top d(\theta) \leq -\frac{1}{\eta}||d(\theta)||^2$:
For any closed convex set Θ and any point u, the projection $\theta' = \Pi_\Theta(u)$ satisfies

$$(u - \theta')^\top(y - \theta') \leq 0 \quad \forall y \in \Theta,$$

as θ' is defines as the unique minimizer of the convex problem $\min_{y\in\Theta} \frac{1}{2}||y-u||^2$, and the first-order (KKT) optimality condition for this strongly convex problem is exactly the equation above.
Since $\theta \in \Theta$, we may plug $y = \theta$ into this inequality:

$$(u - \theta')^\top(\theta - \theta') \leq 0,$$

and substitute $u = \theta - \eta\nabla L(\theta)$ and $\theta - \theta' = d(\theta)$:

$$\begin{aligned}(\theta - \eta\nabla L(\theta) - \theta')^\top d(\theta) \leq 0 \\ \implies \big(d(\theta) - \eta\nabla L(\theta)\big)^\top \leq 0.\end{aligned}$$

$$\begin{aligned}\big(d(\theta) - \eta\nabla L(\theta)\big)^\top &= d(\theta)^\top d(\theta) - \eta\nabla L(\theta)^\top d(\theta) \\ &= ||d(\theta)||^2 - \eta\nabla L(\theta)^\top d(\theta) \leq 0.\end{aligned}$$

$$\eta\nabla L(\theta)^\top d(\theta) \geq ||d(\theta)||^2 \implies \nabla L(\theta)^\top d(\theta) \geq \frac{1}{\eta}||d(\theta)||^2.$$

Finally

$$\begin{aligned}\nabla L(\theta)^\top(\theta' - \theta) &= \nabla L(\theta)^\top(-d(\theta)) \\ &= -\nabla L(\theta)^\top d(\theta) \leq -\frac{1}{\eta}||d(\theta)||^2.\end{aligned}$$

This is the "linear piece" of the smoothness inequality giving us a guaranteed decrease proportional to $||d(\theta)||^2$.
2. Quadratic piece: $\frac{L}{2}||\theta' - \theta||^2 = \frac{L}{2}||d(\theta)||^2$
Putting them together in the smoothness inequality:

$$L(\theta') \leq L(\theta) - \frac{1}{\eta}||d(\theta)||^2 + \frac{L}{2}||d(\theta)||^2.$$

Now by choosing the step size $\eta \leq 1/L$, then

$$-\frac{1}{\eta} + \frac{L}{2} \leq -\frac{1}{2\eta},$$

so overall

$$L(\theta') \leq L(\theta) - \frac{1}{2\eta}||d(\theta)||^2,$$

and rearranging and renaming $\theta' = \theta^{t+1}, \theta = \theta^t$:

$$L(\theta^t) - L(\theta^{t+1}) \ \geq \ \frac{1}{2\eta}||d(\theta^t)||^2.$$

Telescoping (summing the inequalities) this descent bound over $t = 0, \ldots, T$ yields:

$$\sum_{t=0}^{T} \Big[L(\theta^t) - L(\theta^{t+1}) \Big] \geq \frac{1}{2\eta} \sum_{t=0}^{T} \Big[||d(\theta^t)||^2 \Big].$$

On the left the telescoping sum

$$L(\theta^0) - L(\theta^1) \geq \frac{1}{2\eta} ||d(\theta^0)||^2,$$

$$L(\theta^1) - L(\theta^2) \geq \frac{1}{2\eta} ||d(\theta^1)||^2,$$

$$\ldots$$

collapses to $L(\theta^0) - L(\theta^{T+1})$, which is bounded above (since L is lower bounded). Hence the right-hand sum of $||d(\theta^t)||^2$ is finite, forcing $||d(\theta^t)|| \to 0$ as $t \to \infty$: so the projected gradient $d(\theta^t) = \theta^t - \Pi(\theta^t - \eta \nabla L(\theta^t))$ vanishes as $t \to \infty$.
Any limit point θ^* thus satisfies

$$\lim_{t \to \infty} \frac{1}{\eta} (\theta^t - \Pi_\Theta(\theta^t - \eta \nabla L(\theta^t))) = 0 \in \nabla L(\theta^*) + NC_\Theta(\theta^*),$$

i.e. the first-order (Karush-Juhn-Tucker) condition for stationarity on Θ: $NC_\Theta(\theta^*)$ is the normal cone of the convex set Θ at θ^* that collects all vectors:

$$\mathbf{v} \in NC_\Theta(\theta^*) \quad \Longleftrightarrow \quad \mathbf{v}^\top (\mathbf{y} - \theta^*) \leq 0 \quad \forall \mathbf{y} \in \Theta,$$

there exists some normal vector $\mathbf{v} \in NC_\Theta(\theta^*)$ such that $\nabla L(\theta^*) + \mathbf{v} = 0$, or equivalently, it is not possible to find any feasible direction $\mathbf{y} - \theta^*$ along which the directional derivative of L, $\nabla L(\theta^*)^\top d$, satisfies $\nabla L(\theta^*)^\top d < 0$, hence no feasible direction would decrease the loss further. In other words, this is a constrained stationary point under the box constraint Θ – no small perturbation inside Θ can locally lower the objective.

Here $\eta \leq 1/L$ was assumed. In practice, however, computing L exactly for our GNN's prediction-loss is infeasible. Instead, we select the step size (α is η in Algorithm 1) via a grid search in $\{10^{-3}, 10^{-2}, 10^{-1}, 1\}$. All choices empirically respected the surrogate smoothness bound.

3.5.2 Handling non-smooth Thresholding via Surrogate Gradients

Let $P \in [0,1]^{n \times n}$ be our continuous edge-importance matrix. We obtain a hard adjacency

$$\hat{A} = 1[P > 0.5] \in \{0,1\}^{n \times n}$$

for the forward GNN pass. To propagate gradients back to P, we use the common PyTorch "detach" trick (an instance of the straight-through estimator (Bengio et al., 2013))

```
mask   = (P >= 0.5).float() # no gradient

# forward uses mask; backward flows into P
P_hard = P+(mask - P).detach()

outputs = GNN(P_hard, features)
```

Listing 3.1. STE-based hard-thresholding in PyTorch

In effect, the backward pass treats the binarization as if it were the identity function. This can be seen as a limiting case of using a steep sigmoid surrogate $\sigma_\alpha(x) = 1/(1 + e^{-\alpha(x-0.5)})$ with $\alpha \to \infty$. Under this STE, the composite loss

$$L(P) = L_{\text{pred}}(\hat{A}, X) + \beta||P - P_0||_1$$

is differentiable almost everywhere in P, and convergence to a Clarke-stationary point follows from standard projected-gradient arguments. P_0 is the original adjacency mask.
Note that STE is a heuristic used in practice, and the previous convergence proofs refer to the continuous/relaxed problem, not the exact gradients through hard thresholding.

Note that we replace the non-differentiable hard threshold by the identity in the backward pass (i.e. STE). Using the identity in the backward pass induces a bias (dependent on the size of the downstream gradient) by propagating nonzero gradients where the true hard-threshold has zero derivative, whereas zeroing those gradients would irreversibly freeze masked entries – hence, to mitigate this, one may use a temperature-controlled sigmoid σ_α (or even an annealed α) that smoothly trades off bias and trainability.

3.5.3 ℓ_1-Minimality Bound of Perturbations

Let again,

$$\theta = (\text{vec}(P), \text{vec}(N)), \quad \theta_0 = (\text{vec}(A), \text{vec}(1_{n\times f})),$$

and the well-known soft objective

$$L(G;\theta) = L_{\text{pred}}\big(\Phi(G), E(G;\theta)\big) + \beta||\theta - \theta_0||_1,$$

Lemma 3.5.1. *Assume L_{pred} is bounded below and set $\Delta = L_{\text{pred}} - \inf_\Theta L_{\text{pred}}$, the difference of L_{pred} with the infimum (greatest lower bound) of the prediction loss L_{pred} over all feasible $\theta \in \Theta = [0,1]^{n\times n} \times \mathbb{R}^{n\times f}$, the best possible point in Θ achievable by θ^*. Then any stationary point θ^* of L satisfies*

$$||\theta^* - \theta_0||_1 \leq \frac{\Delta}{\beta},$$

where θ_0 is the original unperturbed (P, N)-vector.

Proof. Since $\theta_0 \in \Theta$, stationarity (or just minimality) of θ^* gives

$$L(\theta^*) \leq L(\theta_0) \Longrightarrow L_{\text{pred}}(\theta^*) + \beta||\theta^* - \theta_0||_1 \leq L_{\text{pred}}(\theta_0).$$

Rearranging,

$$\begin{aligned}\beta||\theta^* - \theta_0||_1 &\leq L_{\text{pred}}(\theta_0) - L_{\text{pred}}(\theta^*) \\ &\leq L_{\text{pred}}(\theta_0) - \inf_\Theta L_{\text{pred}} = \Delta\end{aligned}$$

Dividing by β, we get the clean bound

$$||\theta^* - \theta_0||_1 \leq \frac{\Delta}{\beta}.$$

Thus, choosing $\beta \geq \Delta/\epsilon$ forces $||\theta^* - \theta_0||_1 \leq \epsilon$, (i.e. the size of our edits (in ℓ_1) is at most the ratio of "how much we can lower the prediction loss" over "how costly each unit of edit is."

□

Practical choice of β. In theory, the bound

$$\|\theta^* - \theta_0\|_1 \;\le\; \frac{\Delta}{\beta}, \quad \Delta = L(\theta_0) - L(\theta^*)$$

guides β to achieve a target ℓ_1-norm (sparsity). In practice Δ is unknown, so we select β empirically – e.g. via grid search or cross-validation – by monitoring the resulting $\|\theta^*\|_1$ (or edge count) and choosing the smallest β that attains the desired sparsity level ϵ.

3.5.4 Edge-Insertion and Edge-Deletion Condition

This subsection now derives a simple, quantitative criterion under which a zero-entry $p_{ij} = 0$ will be driven strictly positive, and a one-entry $p_{ij} = 1$ will be driven to zero, by projected gradient descent on the soft objective Equation (3.5.1.1).
Recall that L_{dist} penalizes deviations of both edges and node features, but here we focus on its effect on the edge variable p_{ij}.

We consider the soft objective

$$L(P, N) \;=\; L_{\text{pred}}(P, N) \;+\; \beta\, L_{\text{dist}}(P, N),$$

where

- $P = [p_{ij}] \in \{0,1\}^{n\times n}$ is the *final* binary adjacency mask,
- N is the node-feature matrix,
- $\bar{P} = [\bar{p}_{ij}] \in [0,1]^{n\times n}$ is the *original* continuous adjacency (here we assume no noise, in practice we add a small Gaussian noise to $\bar{P}$, which does not materially change the ± 1 subgradients at the boundaries),
- $L_{\text{dist}}(P, N)$ includes the edge penalty $\sum_{i,j} |p_{ij} - \bar{p}_{ij}|$, Equation (3.2.0.7) (plus any feature-penalty terms).

To decide whether an edge coordinate p_{ij} will switch its value under projected gradient descent, we inspect the one-dimensional update

$$p_{ij}^{(t+1)} = \Pi_{\{0,1\}}\Big(p_{ij}^{(t)} - \eta\Big(g_{ij} + \beta\, d_{ij}\Big)\Big),$$

where

$$g_{ij} = \frac{\partial L_{\text{pred}}}{\partial p_{ij}}, \quad d_{ij} = \frac{\partial}{\partial p_{ij}}|p_{ij} - \bar{p}_{ij}| = \operatorname{sign}(p_{ij} - \bar{p}_{ij}).$$

By the sign-definition,

$$d_{ij} = \begin{cases} +1, & p_{ij} > \bar{p}_{ij}, \\ -1, & p_{ij} < \bar{p}_{ij}, \end{cases} \quad d_{ij} \in [-1, 1] \text{ if } p_{ij} = \bar{p}_{ij}.$$

Insertion. At the insertion boundary $p_{ij} = 0$ with continuous $\bar{p}_{ij} > 0$, we have $d_{ij} = -1$ if $\bar{p}_{ij} > 0$, but to capture the switch from 0 to 1 we consider the subgradient at the boundary:

$$p_{ij}^{(t+1)} = 1 \Longleftrightarrow 0 - \eta(g_{ij} + \beta\, d_{ij}) \le -1 \Longleftrightarrow -g_{ij} > \beta\, d_{ij}.$$

Since at the 0-boundary we take $d_{ij} = +1$ for the "hardest" subgradient,

$$-g_{ij} > \beta$$

is required for insertion.

Deletion. At the deletion boundary $p_{ij} = 1$ with $\bar{p}_{ij} < 1$, we similarly take $d_{ij} = -1$ and find

$$p_{ij}^{(t+1)} = 0 \Longleftrightarrow 1 - \eta(g_{ij} + \beta\, d_{ij}) \geq 1 \Longleftrightarrow g_{ij} > \beta.$$

In both cases, an edge flips exactly when its *marginal benefit* or *cost* in the prediction loss exceeds the fixed penalty β, yielding a transparent sparsity threshold.

Role of the Γ Matrix

Rather than using a uniform ℓ_1 penalty $\beta \sum_{i,j} |\bar{P}_{ij} - A_{ij}|$, we shift the target by Γ and write

$$\min_{\bar{P}} \; L_{\text{pred}}(\bar{P}) \; + \; \beta \sum_{i,j} |\bar{P}_{ij} - (A_{ij} + \gamma_{ij})|.$$

By simple algebraic manipulation, this is equivalent (up to an additive constant) to

$$\min_{\bar{P}} \; L_{\text{pred}}(\bar{P}) \; + \; \beta \sum_{i,j} w_{ij}\, |\bar{P}_{ij} - A_{ij}|, \quad w_{ij} \; \equiv \; \frac{1}{1 + \gamma_{ij}}.$$

In the latter form, the KKT subgradient for each (i, j) is

$$\underbrace{\frac{\partial L_{\text{pred}}}{\partial \bar{P}_{ij}}}_{\text{model term}} + \beta\, w_{ij}\, \text{sign}(\bar{P}_{ij} - A_{ij}),$$

so a larger $\gamma_{ij} \Rightarrow$ smaller $w_{ij} \Rightarrow$ smaller magnitude of the regularizer $\Rightarrow$ cheaper to flip edge (i, j), all **without** changing the global β.

3.5.5 Smoothness of the Prediction Loss

We next show, at a high level, that our prediction loss $L_{\text{pred}}(P, N)$ admits an L-Lipschitz continuous gradient (i.e. is L-smooth) in the perturbation variables (P, N).

Lemma 3.5.2. *Let f_θ be a GNN with fixed weights θ, built from layers that are each Lipschitz continuous (e.g. linear transforms, neighbor-aggregation, and elementwise activations such as ReLU). Define*

$$L_{\text{pred}}(P, N) = L(f_\theta(A \odot P,\, X + N),\, y),$$

where L is a twice-differentiable loss (e.g. cross-entropy). Then there exists a constant $L > 0$ depending only on θ, the GNN architecture, and L, such that for all (P, N) and (P', N'),

$$\begin{aligned} &\|\nabla L_{\text{pred}}(P, N) - \nabla L_{\text{pred}}(P', N')\| \\ &\quad \leq L \,\|\, (P, N) - (P', N')\|. \end{aligned}$$

Hence L_{pred} is L-smooth.

High-Level. Note that each GNN layer can be written as the composition of:

- A linear map in (P, N) (adjacency mask enters via $A \odot P$, features via $X + N$), which is Lipschitz.

- An elementwise activation (e.g. ReLU or smooth ReLU), which is 1-Lipschitz.
- A final differentiable loss L, whose gradient is Lipschitz in the model's output.

By the chain rule, the gradient of the overall map $(P, N) \mapsto L(f_\theta(\cdot), y)$ is Lipschitz, with constant

$$L \;\leq\; L_L \prod_{\ell=1}^{L} L_{\text{layer},\ell},$$

where L_L is the loss's Lipschitz constant and each $L_{\text{layer},\ell}$ upper-bounds the Lipschitz constant of layer ℓ. □

Remark. In practice, we do not require the exact value of L, only the existence of such a bound to invoke standard convergence results for projected gradient methods on smooth + non-smooth objectives.

3.5.6 Choosing the Trade-off

In practice, β controls sparsity versus fidelity. A grid search typically selects $\beta \in [0.1, 1]$ to yield few edits while guaranteeing $\Phi(G') \neq \Phi(G)$. The step size (α is η in Algorithm 1) is as well selected via a grid search in $\{10^{-3}, 10^{-2}, 10^{-1}, 1\}$.

3.5.7 Computational Complexity.

Each iteration of Algorithm 1 performs:

- one forward+backward pass through the GNN, which on a graph with n nodes, $|E|$ edges, hidden-dimensionality d and feature-dimensionality f takes

  $$O(|E|\, d + n\, d\, f),$$

  since message-passing scales linearly in edges and feature multiplications scale in $n\, f \times d$;
- elementwise thresholding of the perturbation masks P and N, costing

  $$O(|E| + n\, f),$$

  as we only inspect each edge entry and each feature entry once. And the STE back-pass adds no asymptotic overhead.

Thus, the dominant per-iteration cost is

$$O(|E|\, d + n\, d\, f),$$

i.e., linear in the graph size under standard GNN architectures, ensuring the method scales to large graphs. Recall that K total iterations are performed in Algorithm 1. Treating d as a small constant, when the graph is dense $|E| \sim n^2$, the complexity reduces to $O(n^2)$.

3.5.8 Extension to Weighted and Directed Graphs

Our theoretical developments extend straightforwardly to weighted or directed graphs. For weighted graphs, replace the binary mask $P \in \{0,1\}^{n\times n}$ by a continuous mask $P \in [a,b]^{n\times n}$ (e.g. $[0,1]$), and substitute the unweighted ℓ_1 distance

$$\sum_{i,j} |p_{ij} - \bar{p}_{ij}|$$

with a weighted version

$$\sum_{i,j} w_{ij}\, |p_{ij} - \bar{p}_{ij}|,$$

where $w_{ij} > 0$ can reflect edge-specific costs. All projected gradient and subgradient arguments carry over by projecting onto the box $[a,b]$ and using the weighted sign subgradient for the ℓ_1 term. For directed graphs, simply treat (i,j) and (j,i) as distinct entries in P, and the same "flip when $|\partial L_{\text{pred}}/\partial p_{ij}| > \beta\, w_{ij}$" threshold holds. Convergence to a KKT-stationary point and the ℓ_1-minimality bound remain valid with these modifications, since they rely only on box-constraints and separable ℓ_1 penalties.

3.5.9 Search-Space Expressivity

By jointly parameterizing the adjacency perturbation $\bar{P} \in [0,1]^{n\times n}$ and feature perturbation $\bar{N} \in \mathbb{R}^{n\times f}$ (see **??** and algorithm 1), our gradient-based explainer can, in principle, reach *any* discrete graph–feature configuration (once binarized via sigmoid and thresholding). In practice, however, the non-convex loss landscape can trap plain gradient descent in a local minimum within the basin around the original graph, favoring small, local edits. To counteract this, we add a small amount of Gaussian noise to $\bar{P}$ at initialization. These random perturbations diversify the initial gradient directions, helping the optimizer "hop" across low-gradient regions and leading to a richer exploration of structurally distinct counterfactuals without sacrificing convergence.

Hence, even when the optimization converges, the explainer may fail to reach a counterfactual due to either local minima or an ill-conditioned decision boundary in the oracle – e.g., the true label region may be disjoint, vanishingly thin, or entirely absent near the input. This is reflected in our experiments: some generated counterfactuals either fail to flip the label or result in out-of-distribution inputs. These issues could be mitigated by improving the oracle's inductive bias and generalization capacity.

Chapter 4

Experimental Evaluation

This chapter describes and discuss the experiments performed: the explanation framework setup, datasets used, the detailed configuration adopted, results and their discussion, analysis of the out-of-distribution effect and ablation studies. It is organized as follows:

- The **setup** section (section 4.1) covers the general framework settings used for the explainability experiments.
- The **datasets** section (section 4.2) provides a detailed description of the eighteen datasets used to empirically asses the performance of the proposed method.
- The **configuration** section (section 4.3) defines the detailed settings, hardware and metrics used for the experiments.
- The **results** section (section 4.4) illustrates, compares and explains the empirical results obtained by the proposed method.
- The **residual out-of-distribution influence** section (section 4.5) analyses the semantic property and the interpretable qualitative behaviour of the proposed method.
- The **Interpretability** section (section 4.6) presents valuable intelligibility results of XPlore's inspection of the oracle's models.
- The **ablation studies** section (section 4.7) provides additional insights into the configuration, results, and behaviour of the proposed method.

4.1 Setup

To obtain high performance classification oracles, separate graph neural networks have been trained for each dataset using an 80%-20% train-test dataset split, with architectural and hyperparameter details provided in section 4.3.1 and section 4.3.2, respectively. XPlore is compared against six different explainer baselines: CF-GNNExplainer (Lucic et al., 2022), CF^2 (Tan et al., 2022), CLEAR (Ma et al., 2022), RSGG-CE (Prado-Romero et al., 2024), D4Explainer (Chen et al., 2023), and iRand (Prado-Romero et al., 2023a). For iRand, the edge perturbation probability (p) was set to 0.01 and the number of iterations (t) was set to 3.

Following the protocol from Prado-Romero et al. (2023c), a comprehensive evaluation suite was used, including *Oracle Accuracy*, *Validity*, *Fidelity*, *Sparsity*, *Graph Edit Distance (GED)*, *Oracle Calls* and *Runtime*. Additionally, the cosine similarity (CS), which measures the semantic similarity between original and counterfactual graphs, is used. Unlike structural metrics such as graph edit distance or sparsity, cosine similarity captures alignment in graph meaning by comparing embeddings. Definitions for standard metrics are detailed in 4.3.3.

4.2 Datasets

To evaluate the proposed approach against state-of-the-art graph counterfactual explanation methods, experiments were conducted on eighteen datasets (thirteen real world datasets and five synthetic ones) that cover diverse domains and involve multi-class graph classification tasks. Table 4.1 and table 4.2 illustrate the characteristics of the datasets used in this thesis for graph and node classification-explanation, respectively. Common characteristics are reported in table 4.1.

Table 4.1. Graph explanation datasets characteristics.

Dataset	Avg # Nodes	Node Attr.	Avg # Edges	# Graphs	# Classes	Motif	Category	Oracle Acc.
TCR (explainers)	28	–	27.75	5000	2	Cycle	Synthetic	100.00%
TCR (oracles)	10.45	–	13.04	50000	2	Cycle	Synthetic	Table 4.6
TG	64	1	65.01	5000	2	Grid	Synthetic	99.72%
BAS	64	4	64.01	5000	2	House	Synthetic	100.00%
MUTAG	17.93	–	19.79	188	2	–	Molecular	88.30%
BZR	35.75	3	38.36	405	2	–	Molecular	99.01%
COX2	41.22	3	43.45	467	2	–	Molecular	98.72%
AIDS	15.69	4	16.20	2000	2	–	Molecular	99.80%
BBBP	25.95	–	24.06	2039	2	–	Molecular	99.41%
ENZYMES	32.63	18	62.14	600	6	–	Bioinf.	95.67%
PROTEINS	39.06	29	72.82	1113	2	–	Bioinf.	98.02%
Fingerprint	7.06	2	5.76	2149	15	–	Comp.Vis.	74.97%
COLLAB	74.49	–	2457.78	5000	3	–	Social net.	69.60%
COLOR-3	61.31	4	91.03	10500	11	Color	Synthetic	91.68%
TRIANGLES	20.85	–	32.74	45000	10	Triangle	Synthetic	99.18%
MSRC	77.52	–	198.32	563	20	–	Comp.Vis.	42.27%
DBLP	10.48	–	19.65	19456	2	–	Social net.	80.83%
IMDB	19.77	–	96.53	1000	2	–	Social net.	77.80%
TWITTER	4.03	–	4.98	144033	2	–	Social net.	91.32%

Table 4.2. Node explanation datasets characteristics.

	BAS	BZR	ENZYMES	MSRC	TWITTER
# of Node labels	2	10	3	22	1323
Oracle Test. Acc.	100.00%	99.90%	87.81%	92.81%	3.41%

AIDS (Riesen and Bunke, 2008) consists of graphs representing molecular compounds. These graphs are derived from the AIDS Antiviral Screen Database of Active Compounds. This data set consists of two classes (active and inactive), which represent molecules with or without activity against HIV. The molecules are converted into graphs in a straightforward manner by representing atoms as nodes and the covalent bonds as edges. Nodes are labeled with the number of the corresponding chemical symbol and edges by the valence of the linkage. There are 2,000 elements in total (1,600 inactive elements and 400 active elements).

BAShapes (Ying et al., 2019) is a synthetic dataset consisting of a base graph and motifs connected on the base. The base graph is a Barabasi-Albert (BA) graph with house-shaped motif attached to it. The resulting graph is further perturbed by adding $0.1N$ random edges. Following the generation done in Lucic et al. (2022),

there are 8 nodes on the base graph with 5 edges connecting them. Each base graph has 7 motives connected to it.

BBBP (Blood-Brain Barrier Penetration) (Martins et al., 2012) is a dataset widely used in drug discovery and neurological research to develop machine learning models that predict blood-brain barrier permeability. The blood-brain barrier is a protective membrane that shields the central nervous system by regulating the passage of solutes. Its presence is a critical consideration in drug development, whether for designing molecules that target the central nervous system or for identifying compounds that should be restricted from crossing the barrier. BBBP contains binary labels for 2,053 curated molecules, indicating whether a compound can penetrate the blood-brain barrier. Specifically, 1,570 molecules can penetrate the barrier, while 483 cannot.

BZR and **COX2** (Sutherland et al., 2003) have different molecular compounds, 467 cyclooxygenase-2 (COX-2) inhibitors and 405 benzodiazepine receptor (BZR) ligands, respectively. These datasets are widely used in quantitative structure-activity relationships (QSAR) studies, which attempt to correlate the biological activities of compounds with their structural attributes, to help elucidate the mechanism by which they act and to predict the activities of novel derivatives.

COLLAB (Yanardag and Vishwanathan, 2015) is a social network dataset comprising 5000 scientific collaborations derived from 3 public collaboration datasets (Leskovec et al., 2005), namely, High Energy Physics, Condensed Matter Physics and Astro Physics. Ego-networks of different researchers from each field were generated, and each graph was labeled as the field of the researcher. The task is then to determine whether the egocollaboration graph of a researcher belongs to High Energy, Condensed Matter or Astro Physics field.

COLORS-3 (Knyazev et al., 2019) is a synthetic dataset consisting of 10500 random graphs with 11 classes where features of each node are assigned to one of the three colors (red, green or blue), $\mathbf{p} \in \mathbb{R}^3$. The dataset is extended to higher n-dimensional cases $\mathbf{p} \in \mathbb{R}^n$.

DBLP (Pan et al., 2013) dataset consists of bibliography data in computer science. Each record is associated with a number of attributes such as abstract, authors, year, venue, title, and reference ID. The graph stream is built by selecting the list of conferences and using the papers published in these conferences (in chronological order) to form a binary-class graph stream. The classification task is to predict whether a paper belongs to DBDM (database and data mining) or CVPR (computer vision and pattern recognition) field, by using the references and the title of each paper.

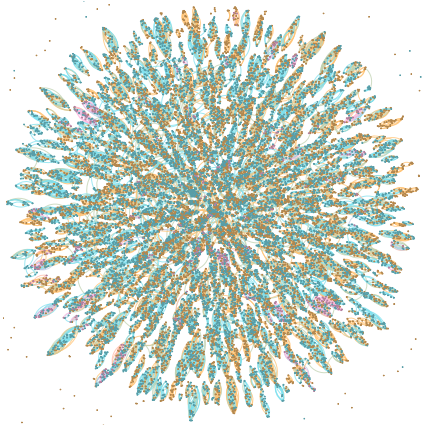
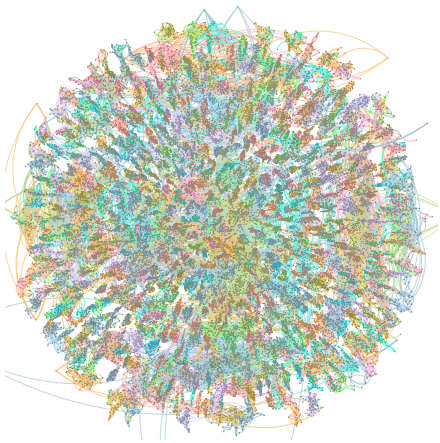


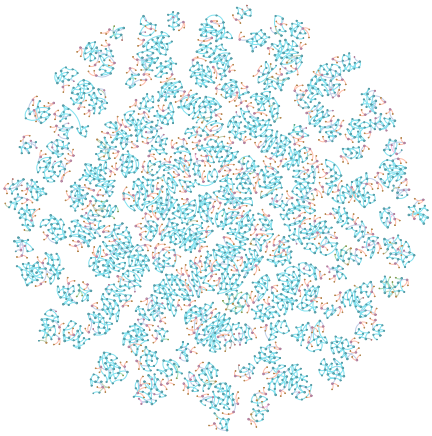
Figure 4.1. ENZYMES dataset.

ENZYMES (Borgwardt et al., 2005) is a bioinformatics dataset consisting of a dataset of 600 enzymes, constructed from the Protein Data Bank (Berman et al., 2000) and labeled with their corresponding enzyme class labels from the BRENDA enzyme database (Schomburg et al., 2004). It includes 100 proteins from each of six classes (EC 1, EC 2, EC 3, EC 4, EC 5, EC 6), which represent proteins out of the six enzyme commission top level hierarchy (EC classes).

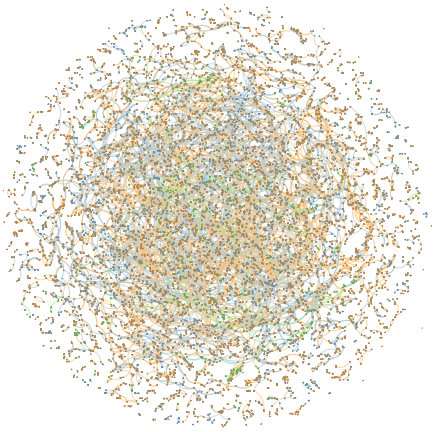
The proteins are converted into graphs by representing the secondary structure elements of a protein with nodes and edges of an attributed graph. Nodes are labeled with their type (helix, sheet, or loop) and their amino acid sequence. Every node is connected with an edge to its three nearest neighbours in space. Edges are labelled with their type and the distance they represent in angstroms.



(a) MSRC21 dataset.



(b) MUTAG dataset.



(c) PROTEINS dataset.

Fingerprint (Riesen and Bunke, 2008) is a computer vision dataset consisting of 2149 fingerprints that are converted into graphs by filtering the images and extracting regions that are relevant (Neuhaus and Bunke, 2005). In order to obtain graphs from fingerprint images, the relevant regions are binarized and a noise removal and thinning procedure is applied. This results in a skeletonized representation of the extracted regions. Ending points and bifurcation points of the skeletonized regions are represented by nodes. Additional nodes are inserted in regular intervals between ending points and bifurcation points. Finally, undirected edges are inserted to link nodes that are directly connected through a ridge in the skeleton. Each node is labeled with a two-dimensional attribute giving its position. The edges are attributed with an angle denoting the orientation of the edge with respect to the horizontal direction.

IMDB is a movie collaboration dataset in which actors/actresses and genre information of different movies on IMDB are collected. For each graph, nodes represent actors/actresses, and there is an edge between them if they appear in the same movie. Collaboration graphs on Action and Romance genres were generated and ego-networks for each actor/actress derived. Note that a movie can belong to both genres at the same time, therefore movies from Romance genre already included to the Action genre were discarded. Similarly to COLLAB dataset, each ego-network was labeled with the genre graph to which belongs to. The task is then simply to identify which genre an ego-network graph belongs to.

MSRC21 (Neumann et al., 2016) is derived from the a state-of-the-art dataset in semantic image processing originally introduced by Winn et al. (2005). Each image is represented by a conditional Markov random field graph. The nodes of each graph are derived by oversegmenting the images using the quick shift algorithm (Vedaldi and Soatto, 2008), resulting in one graph among the superpixels of each image. Nodes are con-

nected if the superpixels are adjacent, and each node can further be annotated with a semantic label. Semantic (ground-truth) node labels are derived by taking the mode ground-truth label of all pixels in the corresponding superpixel. Semantic labels are objects name plus a label void to handle objects that do not fall into one of the given classes. Images consisting of solely one semantic label were removed, leading to a classification task among 20 classes for MSRC21 dataset.

MUTAG (Debnath et al., 1991) is a widely used dataset consisting of 188 nitroaromatic chemical compounds divided into two classes according to their mutagenic effect on a Salmonella typhimurium bacterium. Input graphs represent chemical compounds, vertices stand for atoms and are labeled by the atom type (represented by one-hot encoding), while edges between vertices represent bonds between the corresponding atoms.

PROTEINS (Borgwardt et al., 2005) is a bio-informatics dataset comprising 1113 proteins from the dataset of enzymes (59%) and non-enzymes (41%) (Dobson and Doig, 2003). Proteins are modeled as feature vectors which indicate for each amino acid its fraction among all residues, its fraction of the surface area, the existence of ligands, the size of the largest surface pocket and the number of disulphide bonds.

Tree-Cycles Rand (TCR) (Ying et al., 2019) is an emblematic synthetic dataset. Each instance constitutes a graph comprising a central tree motif and multiple cycle motifs connected through singular edges. The dataset encompasses two distinct classes: i.e., one for graphs without cycles (0) and another for graphs containing cycles (1). The TC also allows control of the number of nodes, the number of cycles, and the number of nodes in them.

Tree-Grid (TG) is a synthetic dataset similar to *Tree-Cyles*, in which n-by-n grid motifs are attached to the main tree motif in place of cycle motifs. We used 5000 graphs with 64 nodes and randomly attached a 3-by-3 grid.

TRIANGLES (Knyazev et al., 2019) is a synthetic dataset comprising 45000 graphs. The task is to count the number of triangles in the graph, node degree features as one-hot vectors are added to all graphs, so that the oracle model can exploit both graph structure and features.

TWITTER (Pan et al., 2014)(Pan et al., 2015) dataset is extracted from twitter sentiment classification. Because of the inherently short and sparse nature, twitter sentiment analysis (i.e., predicting whether a tweet reflects a positive or a negative feeling) is a difficult task. To build a graph dataset, each tweet is represented as a graph by using tweet content, with nodes in each graph denoting the terms and/or smiley symbols and edges indicating the co-occurrence relationship between two words or symbols in each tweet. To ensure the quality of the graph, only tweets containing 20 or more words were used. Tweets from April 6 to June 16 were selected to generate 140,949 graphs (in a chronological order).

4.3 Configuration

The detailed configuration adopted for the empirical evaluation is detailed in this section.

4.3.1 Oracles and training

For the explainers baselines analysis, the oracle model adopted is the Graph Convolutional Network (GCN), which were trained for different datasets with varying architectures and hyperparameters, all using RMSprop optimization and CrossEntropyLoss, with an 80%-20% train-test split. Most datasets (BAS, AIDS, BZR, COX2, TCR, TG, and ENZYMES) used 3 convolutional layers and 1 dense layer, BBBP (5 convolutional, 3 dense), COX2, MUTAG (2 convolutional, 2 dense), TRIANGLES (5 convolutional, 2 dense), COLORS-3 (2 convolutional, 1 dense), COLLAB, PROTEINS (3 convolutional, 2 dense), Fingerprint (5 convolutional, 5 dense) . Learning rates varied between 0.001 and 0.01, with training epochs ranging from 20 to 1000, and batch sizes typically 32, except BBBP, ENZYMES, PROTEINS (64), Fingerprint (128), COLLAB (256), TRIANGLES (1024). The experiments were conducted on an 8GB NVIDIA RTX 4060 GPU with an Intel Core i9-13900HX and 32GB RAM, while node explanation experiments used a 4GB NVIDIA RTX 3050 GPU with an Intel Core i7-11800H and 16GB RAM.

For XPlore study across the oracles baselines, the following training configuration was used: 3 convolutional layers, 1 dense layer, 1.8 linear decay, loss used is CrossEntropyLoss, the batch size is 512, and 100 training epochs with early stopping threshold set at ($1e^{-4}$). Optimizer is RMSprop for all models expect for GIN variants and for the diffusion GECN which use AdamW. Learning rate is set to 0.001 for all models except for GECN (0.01) and for the diffusion GECN ($1e^{-4}$).

4.3.2 Hyperparameters search

Hyperparameters search was performed according to these combination of parameters: $\alpha \in \{0.001, 0.01, 0.1, 1\}, \beta \in \{0, 0.5, 1\}, K \in \{50, 100, 500, 5000\}, \gamma \in \{0, 0.1, 0.01, 0.001\}$. The optimal configuration found was $\alpha = 0.1, \beta = 0.5, K = 500$, $\gamma = 0$, with no random noise initialization.

4.3.3 Metrics

Cosine Similarity (CS) measures the semantic information retained by the counterfactual produced. A counterfactual may be structurally and feature-wise very different from the original instance, nonetheless it could show high semantic similarity to it. Given a set of graph embedders E ($|E| = M$), vector embeddings $\mathbf{e_{ji}} = E_j(G_i)$ and $\mathbf{e'_{ji}} = E_j(G'_i)$ are computed for each graph G_i ($|G| = N$) and its counterfactual G'_i, using embedder $E_j \in E$. Cosine similarity is defined as:

$$\mathrm{CS}(\mathbf{e_{ji}}, \mathbf{e'_{ji}}) = \frac{1}{MN} \sum_{j=1}^{M} \sum_{i=1}^{N} \frac{\mathbf{e_{ji}} \cdot \mathbf{e'_{ji}}}{|\mathbf{e_{ji}}||\mathbf{e'_{ji}}|} \in [-1, 1] \tag{4.3.3.1}$$

The set E includes following embedders: Feather-G (Rozemberczki and Sarkar, 2020), Graph2Vec (Narayanan et al., 2017), NetLSD (Tsitsulin et al., 2018), WaveletCharacteristic (Wang et al., 2021a), IGE (Galland and Lelarge, 2019), LDP (Cai and Wang, 2022), GeoScattering (Gao et al., 2019), GL2Vec (Chen and Koga, 2019), SF (de Lara and Pineau, 2018), and FGDS (Verma and Zhang, 2017).

Oracle Accuracy tells us how accurate the model Φ is at predicting the ground truth labels,

$$\frac{1}{N} \sum_{i=1}^{N} \mathbf{1}[\Phi(G_i) = y_i] \tag{4.3.3.2}$$

Validity illustrates the capability of the explainer to cross the decision boundary of Φ given an input G_i,

$$\frac{1}{N}\sum_{i=1}^{N}\mathbf{1}[\Phi(G_i') \neq \Phi(G_i)] \tag{4.3.3.3}$$

Fidelity Fidelity measures how faithful the explainer's generated counterfactuals are to the original instance's true labels, not just to the oracle's learned decision boundary,

$$\frac{1}{N}\sum_{i=1}^{N}\max(\mathbf{1}(\Phi(G_i) = y_i) - \mathbf{1}[\Phi(G_i') = y_i], 0) \tag{4.3.3.4}$$

Note that if the oracle misclassifies an instance, the explainer may produce a counterfactual that changes the predicted class to the original class; in such cases, accuracy would indicate success even though the counterfactual is not a true solution, whereas fidelity correctly reflects this.

Sparsity measures the ratio between the number of structural features modified to obtain a valid counterfactual explanation and the number of structural features in the original instance.

Graph Edit Distance (GED) tracks the modification applied to the graph of the valid counterfactual explanation, i.e., number of added/removed nodes/edges.

Oracle Calls is the number of times the explainer calls Φ to generate a valid counterfactual explanation.

Runtime to counterfactual: the time in seconds required by the explainer to generate a valid counterfactual explanations.

4.4 Results

This section analyses XPlore results: in section 4.4.1, first benchmarks XPlore's behaviour against the other state-of-the art explainers baselines. Subsequently, Section 4.4.2 studies XPlore's performances across the the state-of-the-art oracles models.

4.4.1 Explainer Comparison

XPlore on average achieves +15.1% percentage improvement in validity, and +14.0% on fidelity on across the board vs. the second-best in graph classification. Table 4.3 reports the performance of XPlore and all state-of-the-art baselines across ten runs on each dataset. See table 4.5 for a broader analysis. The validity, fidelity, sparsity, and oracle calls are shown since they describe the goodness of the explainer in finding valid counterfactuals that are also cheap in terms of querying the underlying predictor. Note that XPlore is the best in 17/18 datasets in terms of validity and 17/18 regarding fidelity. It is clearly illustrated in Figure 4.3 and fig. 4.4. This shows that a simple modification to the learning objective and node feature manipulation enhances the faithfulness of the explainer. XPlore maintains a relatively low number of oracle calls, indicating that in at most 14.692 (see TG) iterations, the optimal counterfactual is found (see line 6 of Algorithm 1). Additionally, the relationship between GED and CS is shown in Figure 4.9.

Table 4.3. Comparison of XPlore with SoTA methods (validity ↑ – up; fidelity ↑ – down). **Bold** is best-performing; underline is second-best. **XPlore is best in 17/18 on validity, and 17/18 on fidelity (of which 1/17 on par with RSGG-CE).** Table 4.5 shows a detailed comparison on other metrics. † indicates the second-best explainer.

	TCR	BAS	BZR	AIDS	ENZYMES	Fingerprint	COLORS-3	TG	MUTAG	COX2	BBBP	PROTEINS	COLLAB	TRIANGLES	DBLP	IMDB	TWITTER	MSRC
iRand	27.92	50.70	27.16	0.00	26.67	0.09	42.99	36.16	2.66	69.81	19.76	18.87	4.60	6.38	1.17	3.70	0.00	95.74
CF^2	50.04	45.78	19.75	0.10	68.33	24.52	52.07	49.86	0.00	24.20	25.26	16.35	52.66	37.13	5.76	50.60	38.82	90.94
CLEAR	50.68	50.96	60.49	16.75	83.17	72.73	0.00	58.40	35.11	22.06	22.90	0.00	0.00	89.99	0.68	56.20	7.65	23.98
RSGG-CE †	67.90	91.04	21.23	19.80	98.33	90.46	94.57	89.28	56.91	**99.36**	22.90	58.67	0.00	99.84	57.51	**86.10**	48.36	**100.0**
D4Explainer	44.82	35.16	20.00	0.10	68.00	24.52	0.00	49.86	9.57	22.06	21.24	0.00	0.00	31.11	7.02	49.60	23.45	90.94
CF-GNNExpl	50.04	44.18	19.75	0.10	68.33	24.52	52.07	49.86	10.11	22.06	22.31	16.35	52.66	37.13	5.76	50.60	38.82	90.94
XPlore	**100.0**	**100.0**	**100.0**	**32.30**	**100.0**	**100.0**	**100.0**	**100.0**	**67.55**	**99.36**	**81.51**	**65.41**	**100.0**	**100.0**	**91.84**	78.80	**50.71**	**100.0**
iRand	0.279	0.507	0.262	0.000	0.238	0.001	0.300	0.356	0.005	0.677	0.275	0.183	0.006	0.053	0.06	0.026	0.000	0.391
CF^2	0.500	0.457	0.188	0.001	0.633	0.133	0.398	0.499	0.005	0.229	0.253	0.151	0.262	0.364	0.027	0.366	0.345	0.371
CLEAR	0.507	0.510	0.595	0.168	0.797	0.515	0.000	0.584	0.309	0.208	0.229	0.000	0.000	0.892	0.005	0.420	0.039	0.012
RSGG-CE †	0.679	0.910	0.202	0.198	0.930	0.653	0.824	0.888	0.516	**0.968**	0.229	0.556	0.000	0.987	0.450	**0.664**	0.434	**0.423**
D4Explainer	0.448	0.446	0.190	0.001	0.632	0.133	0.000	0.499	0.021	0.208	0.212	0.000	0.000	0.302	0.031	0.358	0.161	0.371
CF-GNNExpl	0.500	0.442	0.188	0.001	0.633	0.133	0.398	0.499	0.005	0.208	0.223	0.151	0.262	0.364	0.027	0.366	0.345	0.371
XPlore	**1.000**	**1.000**	**0.980**	**0.323**	**0.942**	**0.730**	**0.896**	**0.994**	**0.548**	**0.968**	**0.803**	**0.627**	**0.570**	**0.988**	**0.748**	0.606	**0.613**	**0.423**

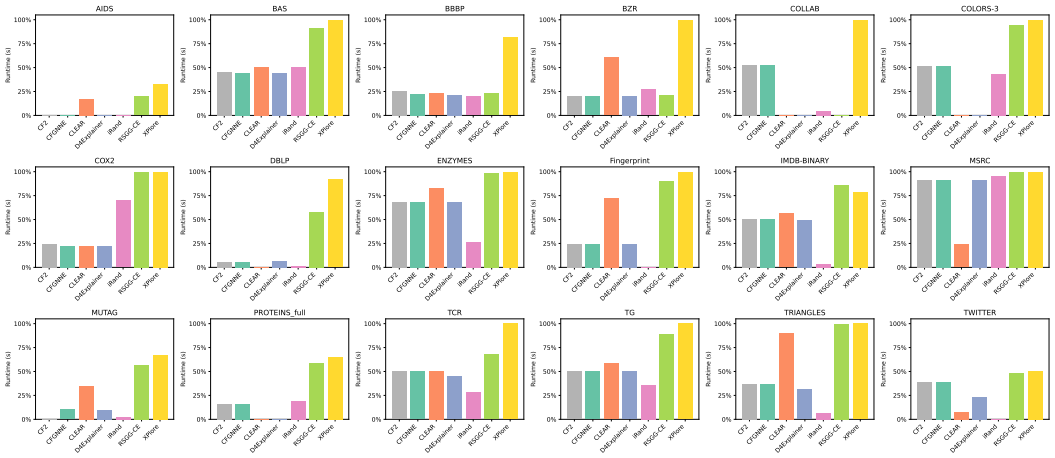


Figure 4.3. Validity: empirical comparison illustration.

However, the interest lies in those explainers that might induce more edit changes to the counterfactual yet maintaining high semantic similarity. Note that XPlore generally exhibits high GED values although it consistently achieves strong CS scores, underlining the intuition that semanticity plays an important role in counterfactuality. More thorough experimentation is reserved for future work. Lastly, the runtime (s) is shown at inference time for all methods to highlight that XPlore is at least on par with SoTA explainers (Figure 4.5).

Table 4.4. Validity and Fidelity results for node explanation.

	BAS		BZR		ENZYMES		MSRC		TWITTER	
Method	Val. ↑	Fid. ↑	Val. ↑	Fid. ↑	Val. ↑	Fid. ↑	Val. ↑	Fid. ↑	Val. ↑	Fid. ↑
CF-GNNExpl	2.42	0.024	7.21	0.071	12.24	0.001	7.23	0.000	26.50	0.000
XPlore	**4.74**	**0.047**	**99.91**	**0.998**	**81.06**	**0.695**	**81.40**	**0.753**	**100.0**	**0.034**

XPlore outperforms CF-GNNExpl on node classification, improving per-class explainability by +62.30 validity points on average. Both methods are compared on five datasets, selecting one per domain. Despite prior claims (Lucic et al., 2022), generating effective node-level counterfactuals remains difficult,

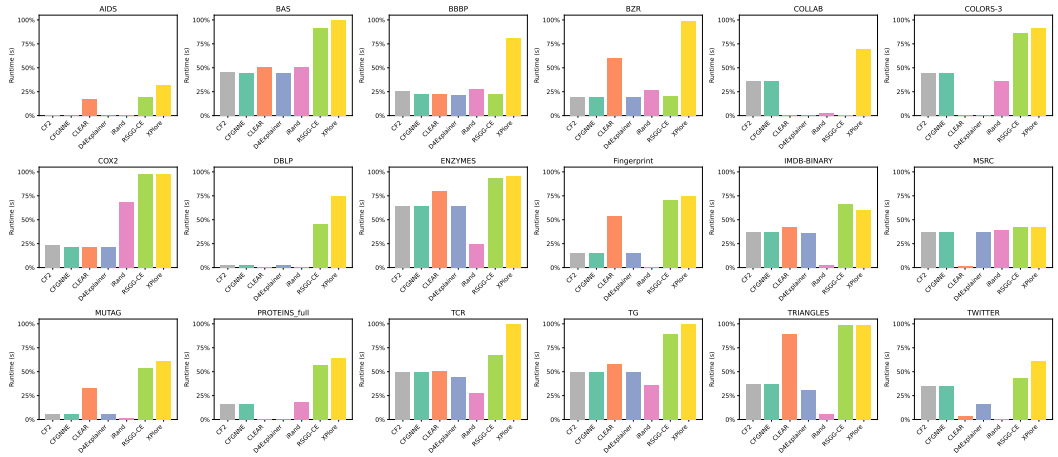


Figure 4.4. Fidelity: empirical comparison illustration.

particularly in class-specific scenarios. As summarized in Table 4.4, XPlore shows consistent improvements, reflecting its ability to finely manipulate node-level features.

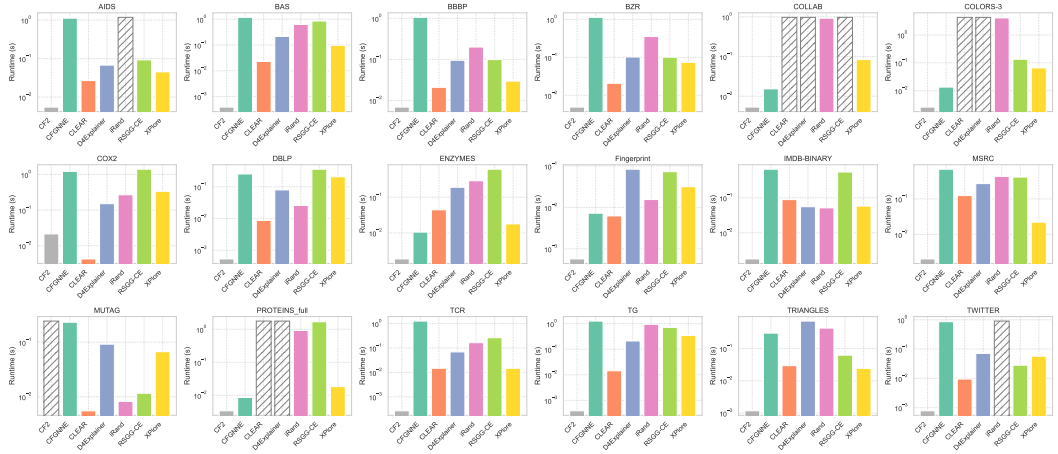


Figure 4.5. Runtime (s) at inference across multiple datasets. XPlore **maintains a competitive runtime, making it the most reliable explainer in practice.** Placeholders are used for explainers with 0% accuracy.

Table 4.5 shows all the performances of XPlore against the state-of-the-art methods. Figure 4.5 shows the runtime of all explainers, showcasing that XPlore is lightweight and only second to CLEAR and CF^2. However, it shows that XPlore is more efficient than its baseline CF-GNNExplainer which is, noticeably, more restrictive in perturbing the input graph (i.e., only allowing edge deletions). Contrarily, XPlore allows for both edge additions and removals as well as node perturbations. Figure 4.11 presents the cosine similarity and graph edit distance distributions, showing that XPlore retains high semantic meaning in the counterfactuals it generates. XPlore performs better in that it can also identify counterfactuals for hard instances where competitors fail. Consequently, it achieves cosine similarity scores that are comparable to or better than those of other baselines on easy samples, while maintaining strong semantic meaning for hard instances not captured by competitors.

Table 4.5. Extended comparison of XPlore with state-of-the-art methods on the used datasets (section 4.2). Standard deviation is reported for each metric. Bold values are the best-performing; underlined values are the second-best ones. Highlighted columns depict the most important metrics as they evaluate the faithfulness of the explainer to produce counterfactuals. When an explainer cannot produce valid counterfactuals in a dataset, it does not make sense to evaluate the other metrics (− symbol is used as placeholder).

Dataset	Method	Validity↑	Fidelity↑	Sparsity↓	Oracle Calls↓
TCR	iRand	27.92% ± 44.86%	0.279 ± 0.449	**0.062** ± 0.023	10.666 ± 0.084
	CF^2	50.04% ± 50.00%	0.500 ± 0.500	0.491 ± 0.000	**0.000** ± 0.000
	CLEAR	50.68% ± 50.00%	0.507 ± 0.500	3.037 ± 0.193	**0.000** ± 0.004
	RSGG-CE	67.90% ± 46.69%	0.679 ± 0.467	0.305 ± 0.057	8.890 ± 0.169
	D4Explainer	44.82% ± 49.73%	0.448 ± 0.497	0.491 ± 0.000	**0.000** 0.012
	CF-GNNExpl	50.04% ± 50.00%	0.500 ± 0.500	0.745 ± 0.000	46.000 ± 0.399
	XPlore	**100.00%** ± 0.00%	**1.000** ± 0.000	2.708 ± 0.906	3.593 ± 0.009

Dataset	Method	Validity↑	Fidelity↑	Sparsity↓	Oracle Calls↓
BAS	iRand	50.70% ± 50.00%	0.507 ± 0.500	**0.112** ± 0.044	43.600 ± 0.293
	CF^2	45.78% ± 49.82%	0.457 ± 0.499	0.496 ± 0.000	**0.000** ± 0.001
	CLEAR	50.96% ± 49.99%	0.510 ± 0.500	6.007 ± 0.325	**0.000** ± 0.035
	RSGG-CE	91.04% ± 28.56%	0.910 ± 0.286	0.331 ± 0.108	23.585 ± 1.069
	D4Explainer	44.56% ± 49.70%	0.446 ± 0.497	0.499 ± 0.002	**0.000** ± 0.014
	CF-GNNExpl	44.18% ± 49.66%	0.442 ± 0.497	0.748 ± 0.000	46.000 ± 0.283
	XPlore	**100.00%** ± 0.00%	**1.000** ± 0.000	7.600 ± 7.557	2.514 ± 0.047

Dataset	Method	Validity↑	Fidelity↑	Sparsity↓	Oracle Calls↓
BZR	iRand	27.16% ± 44.48%	0.262 ± 0.451	**0.104** ± 0.045	25.373 ± 0.213
	CF^2	19.75% ± 39.81%	0.188 ± 0.403	0.516 ± 0.007	**0.000** ± 0.002
	CLEAR	60.49% ± 48.89%	0.595 ± 0.501	4.937 ± 0.843	**0.000** ± 0.012
	RSGG-CE	21.23% ± 40.90%	0.202 ± 0.414	0.258 ± 0.003	2.000 ± 0.015
	D4Explainer	20.00% ± 40.00%	0.190 ± 0.405	0.523 ± 0.006	**0.000** ± 0.015
	CF-GNNExpl	19.75% ± 39.81%	0.188 ± 0.403	0.758 ± 0.003	46.000 ± 0.224
	XPlore	**100.00%** ± 0.00%	**0.980** ± 0.198	6.595 ± 3.616	2.244 ± 0.029

Dataset	Method	Validity↑	Fidelity↑	Sparsity↓	Oracle Calls↓
AIDS	iRand	0.00% ± 0.00%	0.000 ± 0.000	–	–
	CF^2	0.10% ± 3.16%	0.001 ± 0.032	3.870 ± 0.011	**0.000** ± 0.004
	CLEAR	16.75% ± 37.34%	0.168 ± 0.373	13.442 ± 1.834	**0.000** ± 0.008
	RSGG-CE	19.80% ± 39.85%	0.198 ± 0.398	**0.258** ± 0.004	2.000 ± 0.027
	D4Explainer	0.10% ± 0.032%	0.001 ± 0.032	0.488 ± 0.031	**0.000** ± 0.001
	CF-GNNExpl	0.10% ± 3.16%	0.001 ± 0.032	0.745 ± 0.005	46.000 ± 0.059
	XPlore	**32.30%** ± 45.96%	**0.323** ± 0.460	2.445 ± 2.002	5.851± 0.058

Dataset	Method	Validity↑	Fidelity↑	Sparsity↓	Oracle Calls↓
ENZYMES	iRand	26.67% ± 44.22%	0.238 ± 0.445	**0.074** ± 0.046	29.069 ± 0.046
	CF^2	68.33% ± 46.52%	0.633 ± 0.502	0.657 ± 0.034	**0.000** ± 0.004
	CLEAR	83.17% ± 37.42%	0.797 0.402	27.942 ± 24.021	**0.000** ± 0.017
	RSGG-CE	98.33% ± 12.80%	0.930 ± 0.292%	0.306 ± 0.152	19.400 ± 2.801
	D4Explainer	68.00% ± 46.65%	0.632 ± 0.499	0.663 ± 0.499	**0.000** ± 0.066
	CF-GNNExpl	68.33% ± 46.52%	0.633 ± 0.502	0.657 ± 0.034	2.000 ± 0.002
	XPlore	**100.00%** ± 0.00%	**0.942** ± 0.291	2.394 ± 1.538	2.615 ±

Dataset	Method	Validity↑	Fidelity↑	Sparsity↓	Oracle Calls↓
Fingerprint	iRand	0.09% ± 3.05%	0.001 ± 0.030%	**0.031** ± 0.000	3.500 ± 0.007
	CF^2	24.52% ± 43.02%	0.133 ± 0.400	0.412 ± 0.052	**0.000** ± 0.000
	CLEAR	72.73% ± 44.53%	0.515 ± 0.537	15.517 ± 14.400	**0.000** ± 0.001
	RSGG-CE	90.46% ± 29.38%	0.653 ± 0.566	0.329 ± 0.142	6.479 ± 0.048
	D4Explainer	24.52% ± 43.02%	0.133 ± 0.400	0.412 ± 0.052	**0.000** ± 0.014
	CF-GNNExpl	24.52% ± 43.02%	0.133 ± 0.400	0.412 ± 0.052	2.000 ± 0.002
	XPlore	**100.00%** ± 0.00%	**0.730** ± 0.487	0.872 ± 0.591	4.168 ± 0.052

Dataset	Method	Validity↑	Fidelity↑	Sparsity↓	Oracle Calls↓
COLLAB	iRand	4.60% ± 20.95%	0.006 ± 0.213	**0.018** ± 0.021	58.696 ± 2.505
	CF^2	52.66% ± 49.93%	0.262 ± 0.624	0.886 ± 0.061	**0.000** ± 0.003
	CLEAR	0.00% ± 0.00%	0.000 ± 0.000	–	–
	RSGG-CE	0.00% ± 0.00%	0.000 ± 0.000	–	–
	D4Explainer	0.00% ± 0.00%	0.000 ± 0.000	–	–
	CF-GNNExpl	52.66% ± 49.93%	0.262 ± 0.624	0.886 ± 0.061	2.000 ± 0.004
	XPlore	**100.00%** ± 0.00%	**0.570** ± 0.709	0.810 ± 0.068	3.604 ± 0.021

Dataset	Method	Validity↑	Fidelity↑	Sparsity↓	Oracle Calls↓
IMDB	iRand	3.70% ± 18.88%	0.026 ± 0.192	**0.024** ± 0.013	10.865 ± 0.048
	CF^2	50.60% ± 50.00%	0.366 ± 0.674	0.803 ± 0.038	**0.000** ± 0.001
	CLEAR	56.20% ± 49.61%	0.420 ± 0.696	34.773 ± 15.650	**0.000** ± 0.022
	RSGG-CE	**86.10%** ± 34.60%	**0.664** ± 0.802	0.462 ± 0.122	34.890 ± 1.795
	D4Explainer	49.60% ± 50.00%	0.358 ± 0.669	0.797 ± 0.037	**0.000** ± 0.015
	CF-GNNExpl	50.60% ± 50.00%	0.366 ± 0.674	0.902 ± 0.019	46.000 ± 0.230
	XPlore	78.80 ± 40.87%	0.606 ± 0.780	0.390 ± 0.398	5.042 ± 0.047

Dataset	Method	Validity↑	Fidelity↑	Sparsity↓	Oracle Calls↓
COLORS-3	iRand	42.99% ± 49.51%	0.300 ± 0.571	**0.076** ± 0.064	66.175 ± 62.151
	CF^2	52.07% ± 49.96%	0.398 ± 0.568	0.598 ± 0.046	**0.000** ± 0.001
	CLEAR	0.00% ± 0.00%	0.000 ± 0.000	–	–
	RSGG-CE	94.57% ± 22.66%	0.824 ± 0.471	0.312 ± 0.079	4.040 ± 0.848
	D4Explainer	0.00% ± 0.00%	0.000 ± 0.000	–	–
	CF-GNNExpl	52.07% ± 49.96%	0.398 ± 0.568	0.598 ± 0.046	2.000 ±0.007
	XPlore	**100.00%** ± 0.00%	**0.871** ± 0.453	0.536 ± 0.057	12.204 ± 0.155

Dataset	Method	Validity↑	Fidelity↑	Sparsity↓	Oracle Calls↓
TG	iRand	36.16% ± 48.05%	0.356 ± 0.485	**0.175** ± 0.086	69.177 ± 0.0542
	CF^2	49.86% ± 50.00%	0.499 ± 0.500	0.496 ± 0.000	**0.000** ± 0.001
	CLEAR	58.40% ± 49.29%	0.584 ± 0.493	6.353 ± 0.400	**0.000** ± 0.003
	RSGG-CE	89.28% ± 30.94%	0.888 ± 0.324	0.316 ± 0.091	19.152 ± 0.911
	D4Explainer	49.86% ± 50.00%	0.499 ± 0.500	0.500 ± 0.001	**0.000** ± 0.013
	CF-GNNExpl	49.86% ± 50.00%	0.499 ± 0.500	0.784 ± 0.000	46.000 ± 0.414
	XPlore	**100.00%** ± 0.00%	**0.994** ± 0.106	0.248 ± 0.004	14.692 ± 0.236

Dataset	Method	Validity↑	Fidelity↑	Sparsity↓	Oracle Calls↓
MUTAG	iRand	2.66% ± 16.09	0.005 ± 0.163	**0.035** ± 0.011	4.200 ± 0.004
	CF^2	0.00% ± 0.00%	0.005 ± 0.318	–	–
	CLEAR	35.11% ± 47.73%	0.309 ± 0.506	2.438 ± 0.375	**0.000** ± 0.001
	RSGG-CE	56.91% ± 49.52%	0.516 ± 0.550	0.264 ± 0.004	2.000 ± 0.002
	D4Explainer	9.57% ± 29.42%	0.021 ± 0.309	0.526 ± 0.016	**0.000** ± 0.007
	CF-GNNExpl	10.11% ± 30.14%	0.005 ± 0.318	0.758 ± 0.005	46.000 ± 0.147
	XPlore	**67.55%** ± 46.82%	**0.548** ± 0.613	0.459 ± 0.036	2.866 ± 0.089

Dataset	Method	Validity↑	Fidelity↑	Sparsity↓	Oracle Calls↓
COX2	iRand	69.81% ± 45.91%	0.677 ± 0.490	**0.087** ± 0.049	22.482 ± 0.192
	CF^2	24.20% ± 42.83%	0.229 ± 0.435	4.354 ± 0.235	**0.000** ± 0.005
	CLEAR	22.06% ± 41.46%	0.208 ± 0.421	0.512 ± 0.001	**0.000** ± 0.002
	RSGG-CE	**99.36%** ± 7.99%	**0.968** ± 0.238	0.761 0.366	92.914 ± 1.102
	D4Explainer	22.06% ± 41.46%	0.208 ± 0.421	0.518 ± 0.003	**0.000** ± 0.025
	CF-GNNExpl	22.06% ± 41.46%	0.208 ± 0.421	0.756 ± 0.001	46.000 0.198
	XPlore	**99.36%** ± 7.99%	**0.968** ± 0.238	7.682 3.704	11.388 ± 0.568

Dataset	Method	Validity↑	Fidelity↑	Sparsity↓	Oracle Calls↓
BBBP	iRand	19.76% ± 39.82%	0.275 ± 0.460	**0.059** ± 0.037	12.928 ± 0.190
	CF^2	25.26% ± 43.45%	0.253 ± 0.434	0.510 ± 0.024	**0.000** ± 0.002
	CLEAR	22.90% ± 42.02%	0.229 ± 0.420	51.889 ± 32.182	**0.000** ± 0.022
	RSGG-CE	22.90% ± 42.02%	0.229 ± 0.420	0.258 ± 0.006	2.000 8.131
	D4Explainer	21.24% ± 40.90%	0.212 ± 0.409	0.524 ± 0.010	**0.000** ± 0.038
	CF-GNNExpl	22.31% ± 41.64%	0.223 ± 0.416	0.758 ± 0.005	46.000 ± 0.185
	XPlore	**81.51%** % 38.82%	**0.803** ± 0.412	2.217 ± 1.206	4.055 ± 0.030

Dataset	Method	Validity↑	Fidelity↑	Sparsity↓	Oracle Calls↓
PROTEINS	iRand	18.87% ± 39.13%	0.183 ± 0.394	**0.107** ± 0.078	75.524 ± 4.036
	CF^2	16.35% ± 36.98%	0.151 ± 0.375	0.651 ± 0.025	**0.000** ± 0.001
	CLEAR	0.00% ± 0.00%	0.000 ± 0.000	–	–
	RSGG-CE	58.67% 49.24%	0.556 ± 0.527	0.664 ± 0.477	78.757 ± 3.667
	D4Explainer	0.00% ± 0.00%	0.000 ± 0.000	–	–
	CF-GNNExpl	16.35% ± 36.98%	0.151 ± 0.375	0.651 ± 0.025	2.000 ± 0.002
	XPlore	**65.41%** ± 47.57%	**0.627** ± 0.511	0.569 ± 0.039	2.798 ± 0.006

Dataset	Method	Validity↑	Fidelity↑	Sparsity↓	Oracle Calls↓
MSRC	iRand	95.74% ± 20.20%	0.391 ± 0.656	**0.027** ± 0.039	24.731 ± 0.383
	CF^2	90.94% ± 28.70%	0.371 ± 0.523	0.719 ± 0.009	**0.000** ± 0.000
	CLEAR	23.98% ± 42.70%	0.012 ± 0.158	10.630 ± 1.095	**0.000** ± 0.034
	RSGG-CE	**100.00%** ± 0.00%	**0.423** ± 0.538	0.368 ± 0.050	7.536 ± 1.666
	D4Explainer	90.94% ± 28.70%	0.371 ± 0.523	0.721 ± 0.009	**0.000** ± 0.047
	CF-GNNExpl	90.94% ± 28.70%	0.371 ± 0.523	0.859 ± 0.004	46.000 ± 0.078
	XPlore	**100.00%** ± 0.00%	**0.423** ± 0.569	5.059 ± 0.756	2.089 ± 0.006

Dataset	Method	Validity↑	Fidelity↑	Sparsity↓	Oracle Calls↓
DBLP	iRand	1.17% ± 10.74%	0.006 ± 0.108	**0.032** ± 0.019	8.780 ± 0.018
	CF^2	5.76% ± 23.29%	0.027 ± 0.240	0.530 ± 0.158	**0.000** ± 0.001
	CLEAR	0.68% ± 8.21%	0.005 ± 0.082	3.528 ± 1.037	**0.000** ± 0.003
	RSGG-CE	57.51% ± 49.43%	0.450 ± 0.686	0.419 ± 0.225	21.848 ± 0.844
	D4Explainer	7.02% ± 25.55%	0.031 ± 0.265	0.599 ± 0.089	**0.000** ± 0.021
	CF-GNNExpl	5.75% ± 23.29%	0.027 ± 0.240	0.765 ± 0.079	46.000 ± 0.035
	XPlore	**91.84%** ± 27.37%	**0.748** ± 0.765	0.569 ± 0.278	36.115 ± 0.097

Dataset	Method	Validity↑	Fidelity↑	Sparsity↓	Oracle Calls↓
TWITTER	iRand	0.00% ± 0.00%	0.000 ± 0.000	–	–
	CF^2	38.82% ± 48.74%	0.345 ± 0.545	0.476 ± 0.111	**0.000** ± 0.001
	CLEAR	7.65% ± 26.58%	0.039 ± 0.277	2.352 ± 0.246	**0.000** ± 0.003
	RSGG-CE	48.36% ± 49.97%	0.434 ± 0.579	0.238 ± 0.056	2.000 ± 0.003
	D4Explainer	23.45% ± 42.37%	0.161 ± 0.476	0.386 ± 0.157	**0.000** ± 0.012
	CF-GNNExpl	38.82% ± 48.74%	0.345 ± 0.545	0.476 ± 0.111	1.000 ± 0.061
	XPlore	**50.71%** ± 50.00%	**0.613** ± 0.626	**0.217** ± 0.143	1.000 ± 0.027

Dataset	Method	Validity↑	Fidelity↑	Sparsity↓	Oracle Calls↓
TRIANGLES	iRand	6.38% ± 24.44%	0.053 ± 0.247	**0.123** ± 0.080	71.421 ± 0.346
	CF^2	37.13% ± 48.31%	0.364 ± 0.488	0.621 0.032	**0.000** ± 0.001
	CLEAR	89.99% ± 30.01%	0.892 ± 0.311	53.517 ± 36.786	**0.000** ± 0.019
	RSGG-CE	99.84% ± 4.00%	0.987 ± 0.141	0.325 ± 0.080	5.161 ± 0.378
	D4Explainer	31.11% ± 46.29%	0.302 ± 0.468	0.619 ± 0.038	**0.000** ± 76.890
	CF-GNNExpl	37.13% ± 48.31%	0.364 ± 0.488	0.621 ± 0.032	46.000 ± 0.056
	XPlore	**100.00%** ± 0.00%	**0.988** ± 0.138	0.532 ± 0.053	3.366 ± 0.026

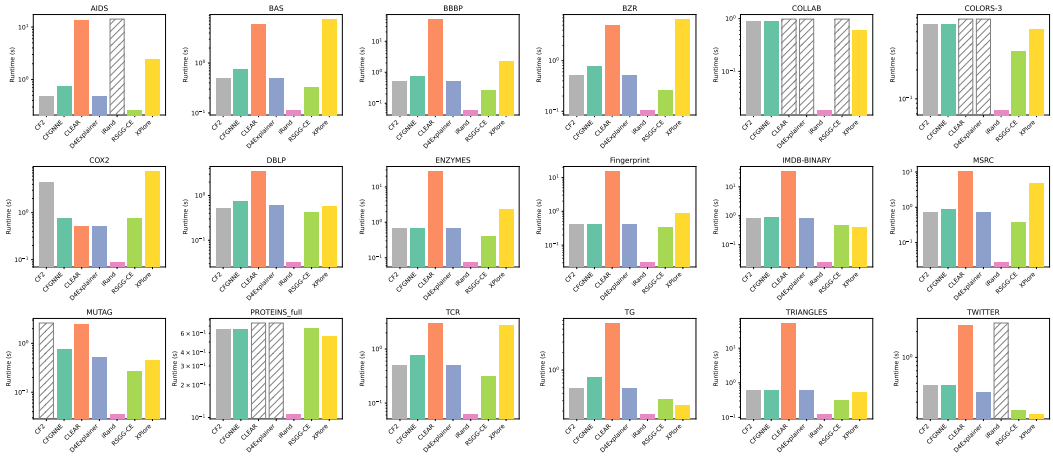


Figure 4.6. Sparsity: empirical comparison illustration.

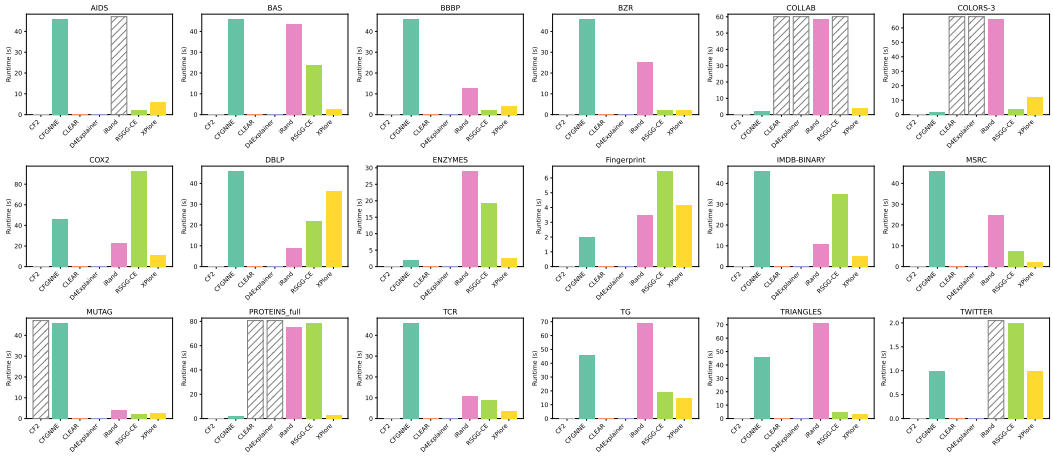


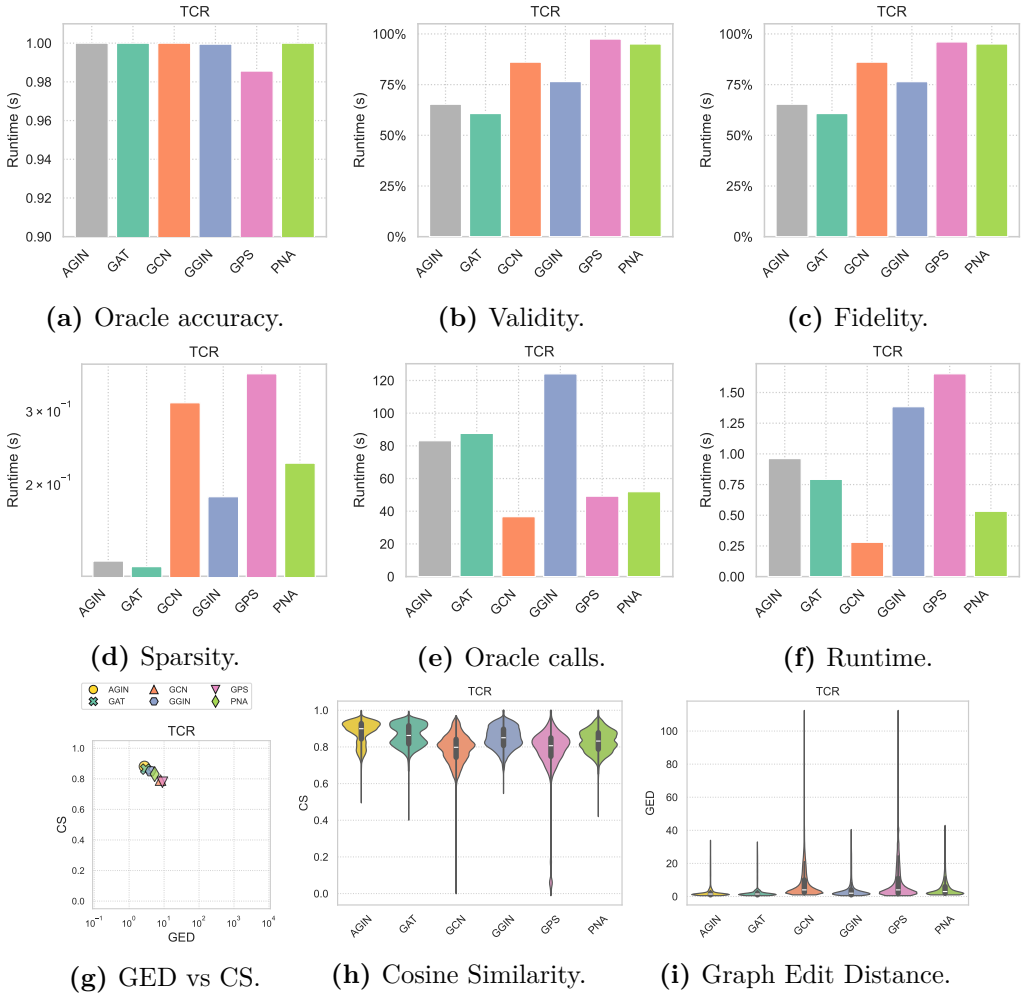
Figure 4.7. Oracle Calls: empirical comparison illustration.

4.4.2 Oracle Comparison

This section reports quantitative results for XPlore's counterfactual explanations across for TCR dataset. To investigate XPlore semantic quality, eight state-of-the-art oracles baselines are used, covering most of the present graph classification techniques and methodologies.

Table 4.6. Extended comparison of XPlore with state-of-the-art oracle models on the TCR dataset. Standard deviation is reported for each metric. Bold values are the best-performing; underlined values are the second-best ones.

Oracle	Oracle Accuracy	Validity ↑	Fidelity ↑	Sparsity ↓	Oracle Calls ↓	Graph Edit Distance↓	Runtime ↓	Cosine Similarity ↑
GECN	**100%** ± 0.00%	86.08% ± 34.62%	0.861 ± 0.346	0.312 ± 0.285	36.661 ± 0.501	7.663 ± 7.959	**0.280** ± 62.671	0.791 ± 107
GEAT	**100%** ± 0.45%	60.75% ± 48.83%	0.607 ± 0.488	**0.126** ± 0.200	87.618 ± 1.188	3.054 ± 5.046	0.792 ± 111.875	0.863 ± 0.085
A-GIN	**100%** ± 0.00%	65.32% ± 47.59%	0.653 ± 0.476	0.130 ± 0.179	83.140 ± 1.568	**2.940** ± 4.600	0.962 ± 121.453	**0.881** ± 0.081
G-GIN	99.95% ± 2.28%	76.50% ± 42.40%	0.764 ± 0.426	0.185 ± 0.223	124.055 ± 1.576	4.125 ± 5.513	1.384 ± 126.653	0.850 ± 0.084
E-GRU	**100%** ± 0.00%	62.26% ± 48.47%	0.623 ± 0.346	0.266 ± 0.291	**26.603** ± 0.973	5.890 ± 7.143	0.496 ± 48.662	0.801 ± 0.107
E-PNA	**100%** ± 0.00%	95.03% ± 21.74%	0.950 ± 0.217	0.223 ± 0.230	51.954 ± 0.767	5.084 ± 5.730	0.532 ± 69.966	0.831 ± 0.090
E-GPS	98.56% ± 11.92%	**97.47%** ± 15.72%	**0.960** ± 0.282	0.366 ± 0.453	49.210 ± 2.898	8.817 ± 11.820	1.652 ± 82.816	0.778 ± 0.186
GECN-DDPM	**100%** ± 0.00%	86.08% ± 34.62%	0.861 ± 0.346	0.312 ± 0.285	36.661 ± 0.501	7.663 ± 7.959	**0.280** ± 62.671	0.791 ± 107



(a) Oracle accuracy. **(b)** Validity. **(c)** Fidelity.

(d) Sparsity. **(e)** Oracle calls. **(f)** Runtime.

(g) GED vs CS. **(h)** Cosine Similarity. **(i)** Graph Edit Distance.

Figure 4.8. Illustration of the XPlore counterfactual explanations comparison with state-of-the-art oracle models for TCR dataset.

Table 4.6 and fig. 4.8 highlight that according to the oracle utilized, XPlore manifests

a different behaviour. Validity and Fidelity may range significantly. Cosine similarity is within the same range for all oracle models, therefore it seems that XPlore is able to find comprehensible explanations for each of them. Section 4.6 will show that this may not always be case, confirming that higher values of cosine similarity correspond to better and more interpretable counterfactuals.

A rapid architectural comparison test will help to chose which architecture may be the best fit for the given requirements, for example preferring interpretability (i.e. higher CS scores) to validity. This trade-off could be avoided by bigger and more powerful architectures, future works will explore this area.

4.5 Residual Out-of-distribution Influence

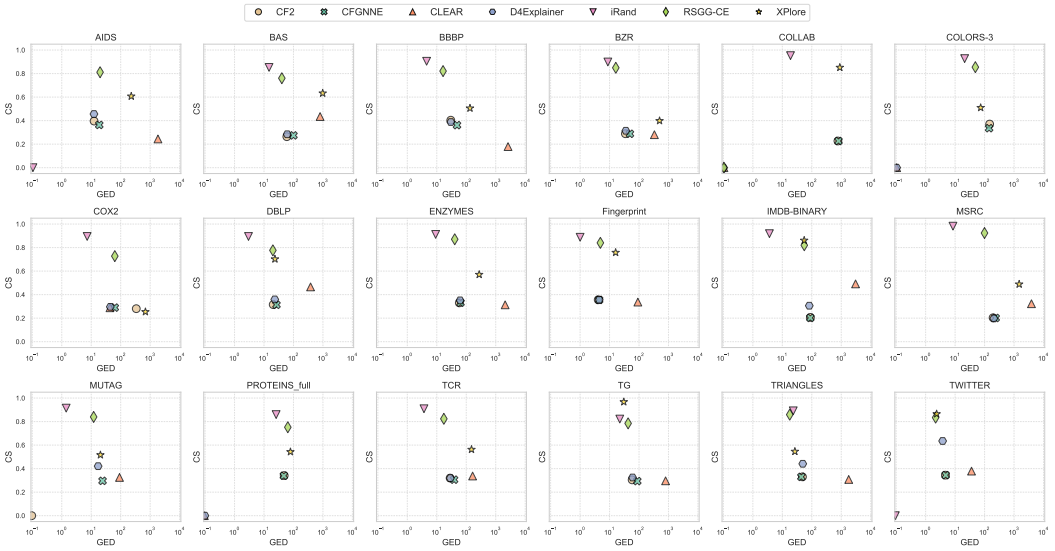


Figure 4.9. Relationship between graph edit distance and cosine similarity. **XPlore consistently achieves strong cosine similarity scores with respect to competitors along-side same graph edit distance values in logarithmic-scale**, indicating that counterfactuals retain meaningful semantic relations to original graphs, partially mitigating out-of-distribution effect.

Chen et al. (2023) observed that CF-GNNExplainer leverages the out-of-distribution (OOD) effect that influences the oracle's prediction, by deriving explanatory subgraphs while omitting additional potential edges.

Consequently, the extracted explanation lacks discriminative information for the counterfactual class but is still classified as counterfactual due to this out-of-distribution effect, ultimately misleading the oracle and compromising reliability. As shown in Figure 4.9, the proposed method reduce out-of-distribution effect exploitation by increasing cosine similarity: XPlore semantic similarity is higher with respect to competitors with similar graph edit distance scores and operating at the same perturbation level , thus relying less on out-of-distribution effect. Figure 4.11 shows that by identifying more CFs, XPlore also captures harder instances, yielding elongated or bimodal cosine similarity distributions, reflecting competitive performance on both easy and hard instances.

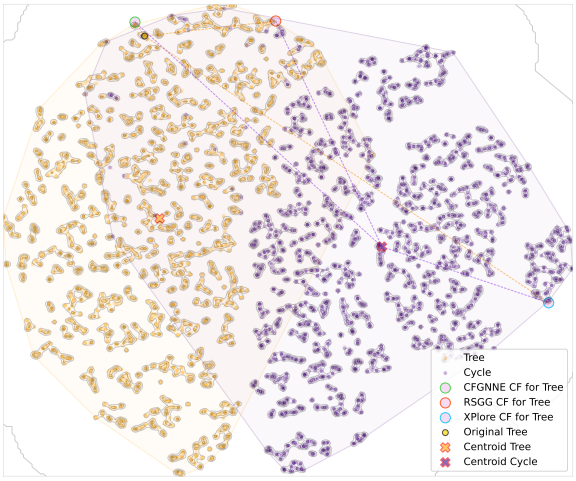
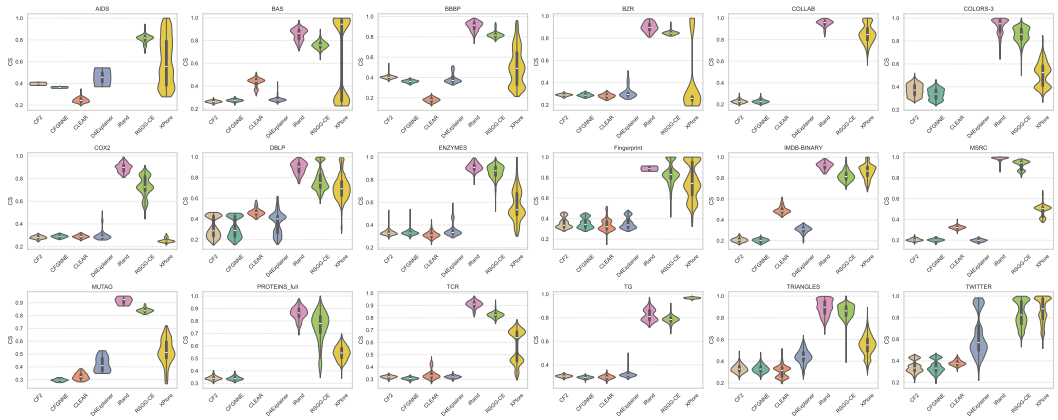
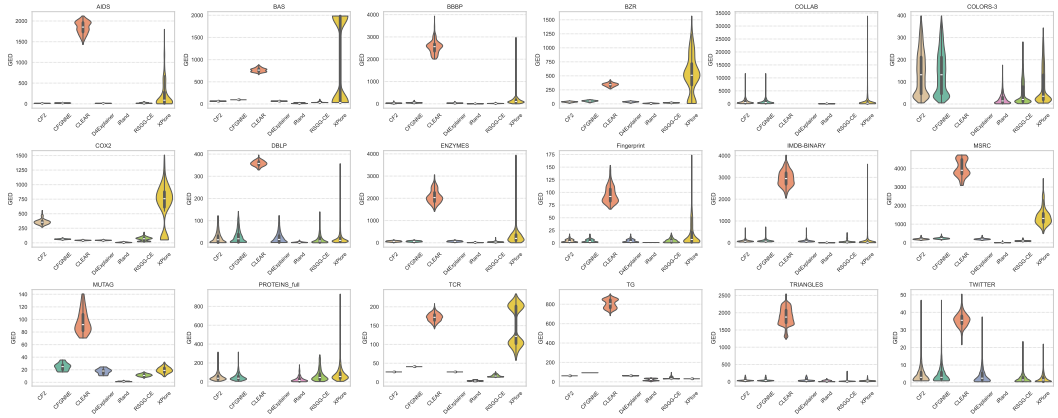


Figure 4.10. t-SNE projection of Wavelet Characteristic embeddings for tree-cycles rand dataset, comparing counterfactuals generated by CF-GNNExplainer, XPlore and RSGG for the Tree motif. CF-GNNExplainer and RSGG find a close CF but fail to land in the Cycle distribution, while XPlore achieves this correctly.

t-SNE projection of the Wavelet
Characteristic embedding space reveals that XPlore's counterfactual exhibits substantial semantic overlap with the target class, landing correctly in the Cycle distribution embedding space, which integrates both topological and node attribute information (Figure 4.10). Other explainers do not achieve this behaviour, remaining within the Tree distribution and exploiting out-of-distribution effect. Although XPlore's ability leads to generate more comprehensive and hence robust explanations, the out-of-distribution effect remains present (4.5.1), this highlights the oracle's limited expressive power and the fact that it was not trained to mitigate adversarial instances. It is possible to refer to Leemann et al. (2024) for non-adversarial counterfactual explanations.



(a) Cosine similarity distributions comparison.



(b) Graph edit distance distributions comparison.

Figure 4.11. XPlore identifies more CFs, thus capturing harder instances. Elongated and bimodal distributions reflect competitive performance and semantic meaning retention.

4.5.1 Out-of-distribution effect analysis

Comparison of XPlore and RSGG: t-SNE projection of Wavelet Characteristic embeddings (Tree-Cycle dataset). We can compare behaviours of XPlore against the second best explainer over the Tree-Cycle dataset. Depending on the original instance, we can see distinct behaviours of the two explainers taking place, let's analyze them.

XPlore unable to land on Tree distribution for a Cycle CF. XPlore sometimes is unable to land on Tree distribution for a Cycle CF (4.12a). 4.12b is a significative error by XPlore. RSGG is sometimes unable to correctly land on the Tree distribution as well (4.12c).

XPlore streghts, RSGG weaknesses. Looking at counterfactuals for the Tree class, the results show an **opposite trend**, XPlore is able to **correctly land CFs** on the Cycle distribution, while RSGG struggles to generate in-distribution CFs, not always landing on the Cycle distribution: (4.12d); yet sometimes, both succeed (4.12e).

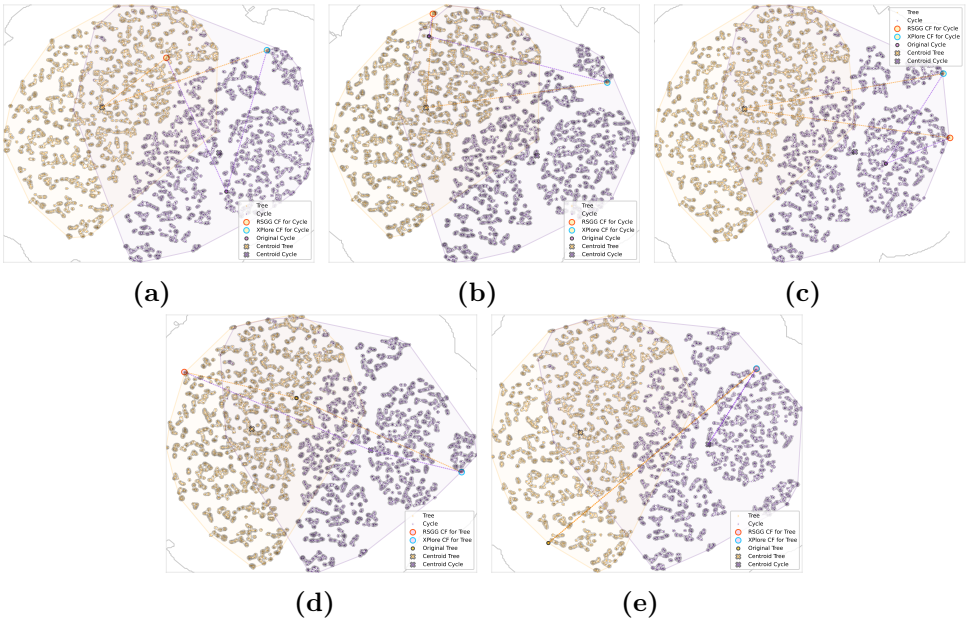


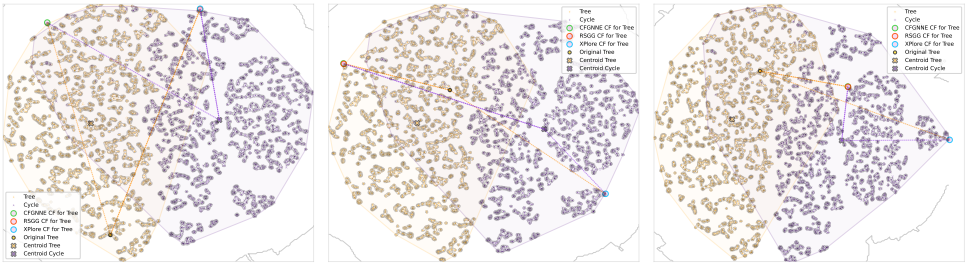
Figure 4.12. Comparison of XPlore and RSGG: t-SNE projection of Wavelet Characteristic embeddings (Tree-Cycle dataset)

Comparison of XPlore, CFGNNE and RSGG: t-SNE projection of Wavelet Characteristic embeddings (Tree-Cycle dataset). We now compare togheter CFs for XPlore, CFGNNE and RSGG. Here we show only CFs for the Tree class, as we plot only instances where all 3 explainers were able to find a counterfactual, and CFGNNE has success only in finding CFs for this class.

CFGNNE does not land on Cycle distribution CFGNNE does not land on Cycle distribution (4.13a), this is expected as **CFGNNE only removes edges**, hence it cannot produce Cycles. Note how **XPlore and RSGG behave similarly**. Sometimes, **XPlore is the only explainer finding a solution** (4.13b). Rarely, the embedder gets tricked (we know CFGNNE cannot produce cycles) (4.13c).

4.6 Interpretability

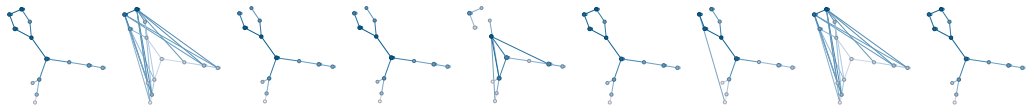
This section focuses on XPlore's qualitative interpretability by reporting representative counterfactual instances illustrations and discussing them.



(a) (b) (c)

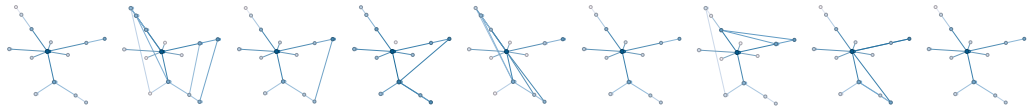
Figure 4.13. Comparison of XPlore, CFGNNE and RSGG: t-SNE projection of Wavelet Characteristic embeddings (Tree-Cycle dataset)

The main goal of an explainer is to produce human-interpretable explanations that can be meaningfully used to inspect the model's behaviour. The explanations may highlight some undesirable quality or trait of the instance or the model itself; the result may be that of updating the input instance to align it more closely to the oracle decision criteria or the retraining or correction of the model itself, this in case the model misclassifies inputs or produces wrong explanations (possibly exploiting the OOD-effect section 4.5).



(a) (b) (c) (d) (e) (f) (g) (h) (i)

Figure 4.14. Counterfactual interpretability illustration: Cycle to Tree. (a) Original Cycle instance. (b) GECN Tree CF. (c) GEAT Tree CF. (d) A-GIN Tree CF. (e) G-GIN Tree CF. (f) E-GRU Tree CF. (g) E-PNA Tree CF. (h) E-GPS Tree CF. (i) GECN DDPM Tree CF.



(a) (b) (c) (d) (e) (f) (g) (h) (i)

Figure 4.15. Counterfactual interpretability illustration: Tree to Cycle. (a) Original Tree instance. (b) GECN Cycle CF. (c) GEAT Cycle CF. (d) A-GIN Cycle CF. (e) G-GIN Cycle CF. (f) E-GRU Cycle CF. (g) E-PNA Cycle CF. (h) E-GPS Cycle CF. (i) GECN DDPM Cycle CF.

As already stated in section 3.2, by working with gradients perturbation and without adding artifacts and heuristics, or relying on another obscure black-box model, XPlore's explanations are able to reflect perfectly the oracle model internal state. Figures 4.14 and 4.15 illustrate counterfactuals found by XPlore for the different oracles models baselines. Note that for this synthetic datasets (TCR, section 4.2), the minimality of the explanation is strongly related to a better explanation; in real-world dataset, this may not be the case, cosine similarity metrics has been introduced for this reason (section 4.3.3). The original Cycle instance is reported in

section 4.6, it has three main branches and a cycle of five nodes. The CF found by the graph edge neural network, basically the entry-level model, reported in section 4.6, is a messy uninterpretable hyper-connected graph. The oracle, even if reaches strong accuracy results (table 4.6), has not high expressive capabilities, and XPlore 'exploits' it to find an OOD counterfactual that should still be classified as a Cycle and not as a Tree. Edge-GraphGPS counterfactual is similar (section 4.6). However, that does not happen for oracles that have a stronger inductive bias, as the edge-GAT **??** or A-GIN (section 4.6) oracles, for which XPlore is able to find similar, minimal, and correct interpretable explanations. Sometimes, in the process of removing one or more edges from the cycle, other edges may be incorrectly added. This behaviour can be seen for the edge-PNA (section 4.6) and gated-GIN (section 4.6) oracles, where one and five new edges are added, respectively. These spurious correlated updates may be seen as uninfluential to the classification by the oracle, thus (mis-)classifying the graph as a Tree, but are hindering, even slightly, the oracle's interpretability. Of course, it may be possible to argue that by eye-inspection of the counterfactual it would be easy to concentrate on the relevant part of the explanation and simply ignore the spurious edges, being able nonetheless to understand the real pattern underlying the explanation; however, this may not always be the case or be possible. Similar results for passing from the Tree class to the Cycle class can be seen in fig. 4.15.

From these figures it is clear that XPlore does not exploit the out-of-distribution effect to find counterfactuals that cheat the oracle but are not meaningful nor interpretable, but rather highlights the limited expressive power of the oracle in case its model is not able to really generalize the task and understand the true semantics relying behind the data. If the oracle has enough inductive power and generalization capabilities, then XPlore will effectively and efficiently find a minimal, interpretable, and therefore useful counterfactual explanation.

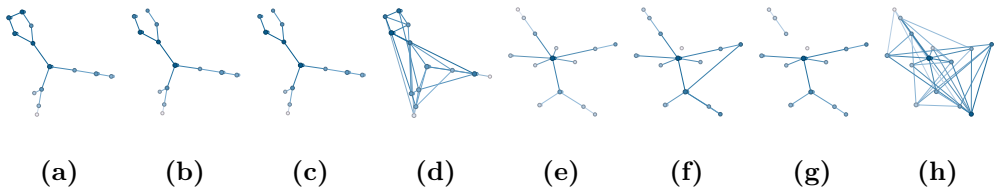


Figure 4.16. Counterfactual interpretability illustration: XPlore against CF-GNNExplainer and RSGG-CE. (a) Original Cycle instance. (b) XPlore Tree CF. (c) CF-GNNExplainer Tree CF. (d) RSGG-CE Tree CF. (e) Original Tree instance. (f) XPlore Cycle CF. (g) CF-GNNExplainer Tree (no CF found) – graph after 500 iterations. (h) RSGG-CE Cycle CF.

Figure 4.16 shows XPlore superior counterfactual explanations quality against CF-GNNExplainer and RSGG-CE. Despite good cosine similarity scores, RSGG-CE produces messy and unintelligible explanations. It can be noted here the CF-GNNExplainer impossibility to add edges to the counterfactual graph, thus missing completely explanations for the whole Cycle class (fig. 4.16 (g)).

4.7 Ablation Studies

Free node perturbation leads to better counterfactual validity. Since XPlore performs free node perturbations, we study the effect of this freedom by constraining it and assessing whether simpler mechanisms work in the same manner. Therefore, we test **(1)** XPlore w/ gating that performs edge additions and node feature manipulation by gating them, similarly to the original edge drop mechanism (Lucic et al., 2022), either retaining or discarding them; and **(2)** a variant that does not have the gating mechanism and cannot freely manipulate node features, called XPlore w/o freedom & gating. To be consistent with the previous nomenclature, we rename XPlore in XPlore w/ freedom. We show the validity of these variants in Table 4.7. Thanks to the better manipulation of graph elements, all three variants have higher validity w.r.t. the original CF-GNNExplainer. The additional ability to alter node features generally leads to even better results. The flexibility given to this modulation allows to tune XPlore to match the explainability criteria and find better counterfactuals.

Table 4.7. Validity (%) for different variants of XPlore ($\gamma = 0.01$). Bold values are best.

	w/o freedom + gating	w/ gating	w/ freedom	CF-GNNExpl
TCR	99.680	65.580	**100.00**	50.040
TG	99.660	**100.00**	**100.00**	49.860
BAS	94.340	**100.00**	**100.00**	44.180
MUTAG	45.745	60.106	**67.553**	10.106
BZR	98.519	**100.00**	**100.00**	19.753
COX2	**99.358**	97.859	98.073	22.056
AIDS	0.300	19.850	**30.300**	0.100
BBBP	38.892	38.794	**79.794**	22.315
ENZYMES	76.500	**99.667**	98.833	68.333
PROTEINS	17.071	64.960	**65.409**	16.352
Fingerprint	26.105	**98.232**	94.742	24.523
COLLAB	55.640	**100.00**	93.380	52.660
COLORS-3	67.943	**100.00**	94.476	52.067
TRIANGLES	39.873	90.436	**100.00**	37.127

In Table 4.8, we report metrics for $\gamma = 0$ and $\gamma = 0.01$ respectively, where γ is the value used to populate the Γ matrix entries that correspond to missing edges of A. Γ is added to P at initialization. Recall that an edge is present if and only if $\sigma(v_i, v_j) > 0.5$ s.t. v_i, v_j are nodes. A positive γ increases the likelihood that edges are present at initialization, since $\sigma(\gamma + \epsilon) > 0.5$. However, because of the added Gaussian noise, presence is not guaranteed. This behavior possibly results in a higher number of counterfactuals found, but at the cost of higher GED, higher sparsity, and lower CS. This functionality may be needed if some edge is required to be present in the CF explanation; this prior knowledge is injected in the form of edge probabilities in matrix Γ.

Table 4.8. Metrics comparison over TCR dataset for different values of hyperparameter γ. We emphasize in bold the variants of XPlore per metric; underlined is the second-best.

	Validity ↑			Fidelity ↑			Sparisty ↓			GED ↓			Oracle Calls ↓			CS ↑		
	$\gamma=0$	$\gamma=0.01$	$\gamma=0.1$	$\gamma=0$	$\gamma=0.01$	$\gamma=0.1$	$\gamma=0$	$\gamma=0.01$	$\gamma=0.1$	$\gamma=0$	$\gamma=0.01$	$\gamma=0.1$	$\gamma=0$	$\gamma=0.01$	$\gamma=0.1$	$\gamma=0$	$\gamma=0.01$	$\gamma=0.1$
w/o freedom + gating	50.04	<u>99.68</u>	70.86	0.500	<u>0.997</u>	0.709	0.745	3.578	3.996	<u>41.00</u>	201.6	224.3	46.00	24.09	16.16	0.309	0.273	0.406
w/ gating	65.58	65.58	49.18	0.656	0.656	0.492	**0.249**	4.931	5.491	**14.00**	278.5	308.5	12.36	3.706	**2.000**	<u>0.955</u>	0.426	0.293
w/ freedom	96.00	**100.0**	**100.0**	0.960	**1.000**	**1.000**	<u>0.251</u>	2.708	4.743	**14.00**	151.02	264.1	5.881	3.593	<u>3.570</u>	**0.956**	0.562	0.337

Chapter 5

Conclusion

This chapter outlines the future research direction, and summarizes the work presented into the manuscript and its contributions.

5.1 Future Work

Possible future research is outlined in this section.

Another avenue would be that of refining the loss function to better balance minimal perturbations with explanation quality and developing more precise control over feature modifications.

5.1.1 Interpretability

Mitigating out-of-distribution (OOD) effects and optimizing node feature perturbations remain key challenges. While the proposed method has shown promise in this area, the capacity to reduce these effects is contingent on how effectively the oracle captures the data distribution. Less robust oracles are more prone to OOD explanations, therefore by adopting more robust oracles, XPlore's OOD effects may be significantly reduced.

Therefore, studying XPlore interpretability against the other explainers baselines across the different oracle baselines, both qualitatively and quantitatively is a promising research direction. Pragmatically, it consists in expanding the work outlined in sections 4.4.2 and 4.6 to assess the semantic and interpretability quality of the counterfactual explanations of the other explainers baselines across the proposed oracle models baselines. Another important avenue would be that of refining the loss function to better balance minimal perturbations with explanation quality and developing more precise control over feature modifications.

5.1.2 Adversarial Training and Attacks

An interesting research avenue would be that of adversarial study. Because XPlore is a perturbation-based explainer, the perturbations it generates can be interpreted not only for explanation but also as structured search directions in the input space. This places XPlore close to the mechanisms underlying adversarial attacks, which also rely on identifying input directions that cause significant changes in model predictions. Although XPlore was designed for interpretability, its ability to localize

impactful perturbations naturally suggests applications in adversarial robustness research.

The perturbations discovered by XPlore can therefore be incorporated into adversarial training pipelines. By augmenting training data with samples that reflect the most influential localized perturbations leading to a misclassification, models can be encouraged to rely less on brittle input patterns. This aligns with the broader idea of explanation-guided training, where explanations identify fragile model behaviours that can then be mitigated through targeted regularization or augmentation.

Moreover, XPlore requires access to the oracle gradients to generate perturbations, which limits its use in black-box or remote adversarial settings where model gradients are unavailable. Therefore, creating explainability methods that are both interpretable and resistant to adversarial manipulation is valid and important research avenue.

5.2 Summary

This manuscript introduced XPlore, a novel counterfactual explainer for Graph Neural Networks (GNNs) that explores a more complete search space, allowing for both edge deletions/insertions and node feature perturbations. The proposed method is not a black box itself; it relies on a basic gradient-based optimization building block to perform an intuitive counterfactual search. Once an oracle is trained, our explainer is fast and lightweight, requiring no further training or intense computation. XPlore identifies minimal yet impactful modifications, ensuring high-quality counterfactual explanations while avoiding heuristic-driven biases. Through extensive benchmarking across various datasets, XPlore consistently outperformed existing state-of-the-art explainers in validity and fidelity. Our empirical evaluation highlights that XPlore achieves an average percentage gain of +17.3% in validity and +15.0% in fidelity over the second-best method across multiple datasets. The method relies on the cosine similarity of graph embeddings as a complementary metric, which demonstrates that XPlore captures structural modifications while maintaining semantic fidelity. This is further demonstrated by the qualitative interpretability of XPlore's counterfactual explanations. This confirms the effectiveness of incorporating a more flexible perturbation space and a comprehensive loss function.

Bibliography

Aamodt, A. and Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications*, 7(1):39–59.

Abrate, C. and Bonchi, F. (2021). Counterfactual graphs for explainable classification of brain networks. In *Proc. of the 27th ACM SIGKDD Conf. on Knowl. Disc. & Data Mining*, pages 2495–2504.

Almeida, L. B. (1990). A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Artificial neural networks: concept learning*, pages 102–111.

Alon, U. and Yahav, E. (2020). On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*.

Anand, N. and Achim, T. (2022). Protein structure and sequence generation with equivariant denoising diffusion probabilistic models. *arXiv preprint arXiv:2205.15019*.

Aragona, D., Podo, L., Prenkaj, B., and Velardi, P. (2021). Coronna: a deep sequential framework to predict epidemic spread. In *Proc. of the 36th Annual ACM Symposium on Applied Computing*, pages 10–17.

Artelt, A. and Hammer, B. (2019). On the computation of counterfactual explanations–a survey. *arXiv preprint arXiv:1911.07749*.

Artelt, A. and Hammer, B. (2020). Convex density constraints for computing plausible counterfactual explanations. In *International conference on artificial neural networks*, pages 353–365. Springer.

Artelt, A., Vaquet, V., Velioglu, R., Hinder, F., Brinkrolf, J., Schilling, M., and Hammer, B. (2021). Evaluating robustness of counterfactual explanations. In *2021 IEEE symposium series on computational intelligence (SSCI)*, pages 01–09. IEEE.

Atwood, J. and Towsley, D. (2016). Diffusion-convolutional neural networks. *Advances in neural information processing systems*, 29.

Azzolin, S., Longa, A., Barbiero, P., Liò, P., and Passerini, A. (2022). Global explainability of gnns via logic combination of learned concepts. *arXiv preprint arXiv:2210.07147*.

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bajaj, M., Chu, L., Xue, Z. Y., Pei, J., Wang, L., Lam, P. C.-H., and Zhang, Y. (2021). Robust counterfactual explanations on graph neural networks. *Advances in Neural Inf. Processing Systems*, 34:5644–5655.

Baldassarre, F. and Azizpour, H. (2019). Explainability techniques for graph convolutional networks. *arXiv preprint arXiv:1905.13686*.

Battaglia, P., Pascanu, R., Lai, M., Jimenez Rezende, D., et al. (2016). Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29.

Beaini, D., Passaro, S., Létourneau, V., Hamilton, W., Corso, G., and Liò, P. (2021). Directional graph networks. In *International Conference on Machine Learning*, pages 748–758. PMLR.

Bengio, Y., Lahlou, S., Deleu, T., Hu, E. J., Tiwari, M., and Bengio, E. (2023). Gflownet foundations. *Journal of Machine Learning Research*, 24(210):1–55.

Bengio, Y., Léonard, N., and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation.

Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., Shindyalov, I. N., and Bourne, P. E. (2000). The protein data bank. *Nucleic acids research*, 28(1):235–242.

Bertsekas, D. P. (1997). Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334.

Bodnar, C., Frasca, F., Otter, N., Wang, Y., Lio, P., Montufar, G. F., and Bronstein, M. (2021). Weisfeiler and lehman go cellular: Cw networks. *Advances in neural information processing systems*, 34:2625–2640.

Bodria, F., Giannotti, F., Guidotti, R., Naretto, F., Pedreschi, D., and Rinzivillo, S. (2023). Benchmarking and survey of explanation methods for black box models. *Data Mining and Knowledge Discovery*, 37(5):1719–1778.

Bojchevski, A., Shchur, O., Zügner, D., and Günnemann, S. (2018). Netgan: Generating graphs via random walks. In *International conference on machine learning*, pages 610–619. PMLR.

Booch, G. (1982). Object-oriented design. *ACM SIGAda Ada Letters*, 1(3):64–76.

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S., Smola, A. J., and Kriegel, H.-P. (2005). Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56.

Bouritsas, G., Frasca, F., Zafeiriou, S., and Bronstein, M. M. (2022). Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):657–668.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*.

Cai, C. and Wang, Y. (2022). A simple yet effective baseline for non-attributed graph classification.

Cai, H., Zheng, V. W., and Chang, K. C.-C. (2018). A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Trans. on Knowl. and Data Engineering*, 30(9):1616–1637.

Cai, J.-Y., Fürer, M., and Immerman, N. (1992). An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410.

Cai, T., Luo, S., Xu, K., He, D., Liu, T.-y., and Wang, L. (2021). Graphnorm: A principled approach to accelerating graph neural network training. In *International Conference on Machine Learning*, pages 1204–1215. PMLR.

Chen, H. and Koga, H. (2019). Gl2vec: Graph embedding enriched by line graphs with edge features. In *Neural Information Processing: 26th Int. Conf. ICONIP 2019*, pages 3–14. Springer.

Chen, J., Wu, S., Gupta, A., and Ying, R. (2023). D4explainer: In-distribution explanations of graph neural network via discrete denoising diffusion. *Adv. in Neural Inf. Processing Systems*, 36:78964–78986.

Chen, X., Han, X., Hu, J., Ruiz, F. J., and Liu, L. (2021). Order matters: Probabilistic modeling of node sequence for graph generation. *arXiv preprint arXiv:2106.06189*.

Chen, X., Li, Y., Zhang, A., and Liu, L.-p. (2022a). Nvdiff: Graph generation through the diffusion of node vectors. *arXiv preprint arXiv:2211.10794*.

Chen, Z., Silvestri, F., Wang, J., Zhang, Y., Huang, Z., Ahn, H., and Tolomei, G. (2022b). Grease: Generate factual and counterfactual explanations for gnn-based recommendations. *arXiv preprint arXiv:2208.04222*.

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Ciravegna, G., Barbiero, P., Giannini, F., Gori, M., Lió, P., Maggini, M., and Melacci, S. (2023). Logic explained networks. *Artificial Intelligence*, 314:103822.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. (2020). Principal neighbourhood aggregation for graph nets. *Advances in neural information processing systems*, 33:13260–13271.

Corso, G., Stark, H., Jegelka, S., Jaakkola, T., and Barzilay, R. (2024). Graph neural networks. *Nature Reviews Methods Primers*, 4(1):17.

Cui, H., Dai, W., Zhu, Y., Li, X., He, L., and Yang, C. (2022). Interpretable graph neural networks for connectome-based brain disorder analysis. In *International conference on medical image computing and computer-assisted intervention*, pages 375–385. Springer.

Cui, P., Wang, X., Pei, J., and Zhu, W. (2018). A survey on network embedding. *IEEE transactions on knowledge and data engineering*, 31(5):833–852.

Dai, E. and Wang, S. (2021). Towards self-explainable graph neural network. In *Proceedings of the 30th ACM international conference on information & knowledge management*, pages 302–311.

Dai, H., Kozareva, Z., Dai, B., Smola, A., and Song, L. (2018). Learning steady-states of iterative algorithms over graphs. In *International conference on machine learning*, pages 1106–1114. PMLR.

Dai, H., Nazi, A., Li, Y., Dai, B., and Schuurmans, D. (2020). Scalable deep generative modeling for sparse graphs. In *International conference on machine learning*, pages 2302–2312. PMLR.

De Cao, N. and Kipf, T. (2018). Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*.

de Lara, N. and Pineau, E. (2018). A simple baseline algorithm for graph classification.

Debnath, A. K., Lopez de Compadre, R. L., Debnath, G., Shusterman, A. J., and Hansch, C. (1991). Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797.

Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29.

Delaney, J. S. (2004). Esol: estimating aqueous solubility directly from molecular structure. *Journal of chemical information and computer sciences*, 44(3):1000–1005.

der Maaten, V. (2008). Visualizing data using t-sne. journal of machine learning research. *(No Title)*, 9.

Dhariwal, P. and Nichol, A. (2021). Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794.

Ding, M., Yang, K., Yeung, D.-Y., and Pong, T.-C. (2019). Effective feature learning with unsupervised learning for improving the predictive models in massive open online courses. In *Proc. of the 9th Int. Conf. on Learning Analytics & Knowl.*, pages 135–144.

Dobson, P. D. and Doig, A. J. (2003). Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783.

Donsker, M. D. and Varadhan, S. S. (1975). Asymptotic evaluation of certain markov process expectations for large time, i. *Communications on pure and applied mathematics*, 28(1):1–47.

Douglas, B. L. (2011). The weisfeiler-lehman method and graph isomorphism testing. *arXiv preprint arXiv:1101.5211*.

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

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*, 28.

Dwivedi, V. P., Luu, A. T., Laurent, T., Bengio, Y., and Bresson, X. (2021). Graph neural networks with learnable structural and positional representations. *arXiv preprint arXiv:2110.07875*.

d'Ascoli, S., Touvron, H., Leavitt, M. L., Morcos, A. S., Biroli, G., and Sagun, L. (2021). Convit: Improving vision transformers with soft convolutional inductive biases. In *International conference on machine learning*, pages 2286–2296. PMLR.

Erd, P. (1959). os and a. r enyi, on random graphs i. *Publ. Math. Debrecen*, 6:290–297.

Evdokimov, S. and Ponomarenko, I. (1999). Isomorphism of coloured graphs with slowly increasing multiplicity of jordan blocks. *Combinatorica*, 19(3):321–334.

Fan, S., Wang, X., Mo, Y., Shi, C., and Tang, J. (2022). Debiasing graph neural networks via learning disentangled causal substructure. *Advances in Neural Information Processing Systems*, 35:24934–24946.

Fan, Y., Yao, Y., and Joe-Wong, C. (2021). Gcn-se: Attention as explainability for node classification in dynamic graphs. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 1060–1065. IEEE.

Feng, A., You, C., Wang, S., and Tassiulas, L. (2022). Kergnns: Interpretable graph neural networks with graph kernels. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 6614–6622.

Feng, Q., Liu, N., Yang, F., Tang, R., Du, M., and Hu, X. (2023). Degree: Decomposition based explanation for graph neural networks. *arXiv preprint arXiv:2305.12895*.

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

Galland, A. and Lelarge, M. (2019). Invariant embedding for graph classification. In *ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data*, Long Beach, United States.

Gallicchio, C. and Micheli, A. (2010). Graph echo state networks. In *The 2010 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE.

Gao, F., Wolf, G., and Hirn, M. (2019). Geometric scattering for graph data analysis. In *Proc. of the 36th Int. Conf. on Machine Learning*, volume 97, pages 2122–2131. PMLR.

Getoor, L. (2005). Link-based classification. In *Advanced methods for knowledge discovery from complex data*, pages 189–207. Springer.

Ghorbani, A., Wexler, J., Zou, J. Y., and Kim, B. (2019). Towards automatic concept-based explanations. *Advances in neural information processing systems*, 32.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. Pmlr.

Giorgi, F., Silvestri, F., and Tolomei, G. (2025). Combinex: A unified counterfactual explainer for graph neural networks via node feature and structural perturbations. *arXiv preprint arXiv:2502.10111*.

Gori, M., Monfardini, G., and Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings. 2005 IEEE international joint conference on neural networks, 2005.*, volume 2, pages 729–734. IEEE.

Goyal, P. and Ferrara, E. (2018). Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94.

Guidotti, R. (2024). Counterfactual explanations and how to find them: literature review and benchmarking. *Data Mining and Knowledge Discovery*, 38(5):2770–2824.

Guidotti, R., Monreale, A., Matwin, S., and Pedreschi, D. (2019). Black box explanation by learning image exemplars in the latent feature space. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 189–205. Springer.

Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., and Pedreschi, D. (2018). A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):1–42.

Guidotti, R. and Ruggieri, S. (2019). On the stability of interpretable models. In *2019 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE.

Guo, J., Han, K., Wu, H., Tang, Y., Chen, X., Wang, Y., and Xu, C. (2022). Cmt: Convolutional neural networks meet vision transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12175–12185.

Guo, Z., Wu, Z., Xiao, T., Aggarwal, C., Liu, H., and Wang, S. (2025). Counterfactual learning on graphs: A survey. *Machine Intelligence Research*, 22(1):17–59.

Haefeli, K. K., Martinkus, K., Perraudin, N., and Wattenhofer, R. (2022). Diffusion models for graphs benefit from discrete state spaces. *arXiv preprint arXiv:2210.01549*.

Hamiache, G. and Navarro, F. (2020). Associated consistency, value and graphs. *International Journal of Game Theory*, 49(1):227–249.

Hamilton, W. L., Ying, R., and Leskovec, J. (2017). Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*.

Hammond, D. K., Vandergheynst, P., and Gribonval, R. (2011). Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150.

Han, X. and Zhao, Y. (2022). Interpretable graph reservoir computing with the temporal pattern attention. *IEEE Transactions on Neural Networks and Learning Systems*, 35(7):9198–9212.

He, H., Ji, Y., and Huang, H. H. (2022a). Illuminati: Towards explaining graph neural networks for cybersecurity analysis. In *2022 IEEE 7th European symposium on security and privacy (EuroS&P)*, pages 74–89. IEEE.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

He, W., Vu, M. N., Jiang, Z., and Thai, M. T. (2022b). An explainer for temporal graph neural networks. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 6384–6389. IEEE.

Henaff, M., Bruna, J., and LeCun, Y. (2015). Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*.

Henderson, R., Clevert, D.-A., and Montanari, F. (2021). Improving molecular graph neural network explainability with orthonormalization and induced sparsity. In *International Conference on Machine Learning*, pages 4203–4213. PMLR.

Herath, J. D., Wakodikar, P. P., Yang, P., and Yan, G. (2022). Cfgexplainer: Explaining graph neural network-based malware classification from control flow graphs. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 172–184. IEEE.

Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851.

Ho, J. and Salimans, T. (2022). Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*.

Hoogeboom, E., Satorras, V. G., Vignac, C., and Welling, M. (2022). Equivariant diffusion for molecule generation in 3d. In *International conference on machine learning*, pages 8867–8887. PMLR.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Huang, H., Sun, L., Du, B., Fu, Y., and Lv, W. (2022a). Graphgdp: Generative diffusion processes for permutation invariant graph generation. In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 201–210. IEEE.

Huang, Q., Yamada, M., Tian, Y., Singh, D., and Chang, Y. (2022b). Graphlime: Local interpretable model explanations for graph neural networks. *IEEE Trans. on Knowl. and Data Engineering*.

Huang, Z., Kosan, M., Medya, S., Ranu, S., and Singh, A. (2023). Global counterfactual explainer for graph neural networks. In *Proc. of the 16th ACM Int. Conf. on Web Search and Data Mining*, pages 141–149.

Igashov, I., Stärk, H., Vignac, C., Schneuing, A., Satorras, V. G., Frossard, P., Welling, M., Bronstein, M., and Correia, B. (2024). Equivariant 3d-conditional diffusion model for molecular linker design. *Nature Machine Intelligence*, 6(4):417–427.

Ingraham, J., Garg, V., Barzilay, R., and Jaakkola, T. (2019). Generative models for graph-based protein design. *Advances in neural information processing systems*, 32.

Ioffe, S. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Jaume, G., Pati, P., Bozorgtabar, B., Foncubierta, A., Anniciello, A. M., Feroce, F., Rau, T., Thiran, J.-P., Gabrani, M., and Goksel, O. (2021). Quantifying explainers of graph neural networks in computational pathology. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8106–8116.

Jiménez-Luna, J., Skalic, M., Weskamp, N., and Schneider, G. (2021). Coloring molecules with explainable artificial intelligence for preclinical relevance assessment. *Journal of Chemical Information and Modeling*, 61(3):1083–1094.

Jo, J., Lee, S., and Hwang, S. J. (2022). Score-based generative modeling of graphs via the system of stochastic differential equations. In *International conference on machine learning*, pages 10362–10383. PMLR.

Jørgensen, P. B., Schmidt, M. N., and Winther, O. (2018). Deep generative models for molecular science. *Molecular informatics*, 37(1-2):1700133.

Kakkad, J., Jannu, J., Sharma, K., Aggarwal, C., and Medya, S. (2023). A survey on explainability of graph neural networks. *arXiv preprint arXiv:2306.01958*.

Kearnes, S., McCloskey, K., Berndl, M., Pande, V., and Riley, P. (2016). Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608.

Kim, B., Khanna, R., and Koyejo, O. O. (2016). Examples are not enough, learn to criticize! criticism for interpretability. *Advances in neural information processing systems*, 29.

Kipf, T. N. and Welling, M. (2016). Variational graph auto-encoders. In *NeurIPS Workshop on Bayesian Deep Learning*.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *5th Int. Conf. on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conf. Track Proc.* OpenReview.net.

Knyazev, B., Taylor, G. W., and Amer, M. R. (2019). Understanding attention and generalization in graph neural networks.

Kommiya Mothilal, R., Mahajan, D., Tan, C., and Sharma, A. (2021). Towards unifying feature attribution and counterfactual explanations: Different means to the same end. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pages 652–663.

Kreuzer, D., Beaini, D., Hamilton, W., Létourneau, V., and Tossou, P. (2021). Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629.

Kriege, N. M., Johansson, F. D., and Morris, C. (2020). A survey on graph kernels. *Applied Network Science*, 5(1):6.

Kusner, M. J., Loftus, J. R., Russell, C., and Silva, R. (2018). Counterfactual fairness.

Lasater, C. G. (2006). *Design Patterns*. Jones & Bartlett Publishers, Burlington, MA, USA.

Laugel, T., Lesot, M.-J., Marsala, C., Renard, X., and Detyniecki, M. (2019). The dangers of post-hoc interpretability: Unjustified counterfactual explanations. *arXiv preprint arXiv:1907.09294*.

Leemann, T., Pawelczyk, M., Prenkaj, B., and Kasneci, G. (2024). Towards non-adversarial algorithmic recourse. In *World Conference on Explainable Artificial Intelligence*, pages 395–419. Springer.

Leman, A. and Weisfeiler, B. (1968). A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9):12–16.

Leskovec, J., Kleinberg, J., and Faloutsos, C. (2005). Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187.

Levie, R., Monti, F., Bresson, X., and Bronstein, M. M. (2018). Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109.

Li, W., Li, Y., Li, Z., Hao, J., and Pang, Y. (2023). Dag matters! gflownets enhanced explainer for graph neural networks. *arXiv preprint arXiv:2303.02448*.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. (2015). Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*.

Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. (2018). Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*.

Liang, F., Qian, C., Yu, W., Griffith, D., and Golmie, N. (2022). Survey of graph neural networks and applications. *Wireless Communications and Mobile Computing*, 2022(1):9261537.

Liao, R., Li, Y., Song, Y., Wang, S., Hamilton, W., Duvenaud, D. K., Urtasun, R., and Zemel, R. (2019). Efficient graph generation with graph recurrent attention networks. *Advances in neural information processing systems*, 32.

Lin, H., Huang, Y., Zhang, O., Ma, S., Liu, M., Li, X., Wu, L., Wang, J., Hou, T., and Li, S. Z. (2025). Diffbp: Generative diffusion of 3d molecules for target protein binding. *Chemical Science*, 16(3):1417–1431.

Lin, W., Lan, H., and Li, B. (2021). Generative causal explanations for graph neural networks. In *International conference on machine learning*, pages 6666–6679. PMLR.

Lin, W., Lan, H., Wang, H., and Li, B. (2022). Orphicx: A causality-inspired latent variable model for interpreting graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13729–13738.

Linardatos, P., Papastefanopoulos, V., and Kotsiantis, S. (2020). Explainable ai: A review of machine learning interpretability methods. *Entropy*, 23(1):18.

Liu, Q., Allamanis, M., Brockschmidt, M., and Gaunt, A. (2018). Constrained graph variational autoencoders for molecule design. *Advances in neural information processing systems*, 31.

Liu, Y., Chen, C., Liu, Y., Zhang, X., and Xie, S. (2021). Multi-objective explanations of gnn predictions. In *2021 IEEE Int. Conf. on Data Mining (ICDM)*, pages 409–418. IEEE.

Liu, Y., Zhang, X., and Xie, S. (2024). A differential geometric view and explainability of gnn on evolving graphs. *arXiv preprint arXiv:2403.06425*.

Livi, L. and Rizzi, A. (2013). The graph matching problem. *Pattern Analysis and Applications*, 16:253–283.

Lo, W. W., Kulatilleke, G., Sarhan, M., Layeghy, S., and Portmann, M. (2023). Xg-bot: An explainable deep graph neural network for botnet detection and forensics. *Internet of Things*, 22:100747.

Loyola-González, O. (2019). Black-box vs. white-box: Understanding their advantages and weaknesses from a practical point of view. *IEEE Access*, 7:154096–154113.

Lu, Y.-J. and Li, C.-T. (2020). Gcan: Graph-aware co-attention networks for explainable fake news detection on social media. *arXiv preprint arXiv:2004.11648*.

Lucic, A., Ter Hoeve, M., Tolomei, G., De Rijke, M., and Silvestri, F. (2022). Cf-gnnexplainer: Counterfactual explanations for graph neural networks. In *Int. Conf. on AI and Statistics*, pages 4499–4511. PMLR.

Luo, D., Cheng, W., Xu, D., Yu, W., Zong, B., Chen, H., and Zhang, X. (2020). Parameterized explainer for graph neural network. *Advances in neural information processing systems*, 33:19620–19631.

Luo, S., Su, Y., Peng, X., Wang, S., Peng, J., and Ma, J. (2022). Antigen-specific antibody design and optimization with diffusion-based generative models for protein structures. *Advances in Neural Information Processing Systems*, 35:9754–9767.

Luo, T., Mo, Z., and Pan, S. J. (2023). Fast graph generation via spectral diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(5):3496–3508.

Luong, K.-D., Kosan, M., Da Silva, A. L., and Singh, A. (2023). Robust ante-hoc graph explainer using bilevel optimization. *arXiv preprint arXiv:2305.15745*.

Ma, J., Guo, R., Mishra, S., Zhang, A., and Li, J. (2022). CLEAR: Generative counterfactual explanations on graphs. In *Adv. in Neural Inf. Processing Systems*.

Ma, J., Takigawa, I., and Yamamoto, A. (2025). C2explainer: Customizable mask-based counterfactual explanation for graph neural networks. In *Proceedings of the 2025 ACM Conference on Fairness, Accountability, and Transparency*, pages 137–149.

Ma, Z., Gu, H., and Liu, Z. (2021). Understanding drug abuse social network using weighted graph neural networks explainer. In *International Conference on Computational Science and Its Applications*, pages 52–61. Springer.

Maddison, C. J., Mnih, A., and Teh, Y. W. (2016). The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.

Magister, L. C., Kazhdan, D., Singh, V., and Liò, P. (2021). Gcexplainer: Human-in-the-loop concept-based explanations for graph neural networks. *arXiv preprint arXiv:2107.11889*.

Martins, I. F., Teixeira, A. L., Pinheiro, L., and Falcao, A. O. (2012). A bayesian approach to in silico blood-brain barrier penetration modeling. *Journal of chemical information and modeling*, 52(6):1686–1697.

Mauri, C., Cerri, S., Puonti, O., Mühlau, M., and Van Leemput, K. (2022). Accurate and explainable image-based prediction using a lightweight generative model. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 448–458. Springer.

Miao, S., Liu, M., and Li, P. (2022a). Interpretable and generalizable graph learning via stochastic attention mechanism. In *International conference on machine learning*, pages 15524–15543. PMLR.

Miao, S., Luo, Y., Liu, M., and Li, P. (2022b). Interpretable geometric deep learning via learnable randomness injection. *arXiv preprint arXiv:2210.16966*.

Micheli, A. (2009). Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511.

Molnar, C. (2020). *Interpretable machine learning*. Lulu. com.

Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. (2020). Tudataset: A collection of benchmark datasets for learning with graphs.

Mothilal, R. K., Sharma, A., and Tan, C. (2020). Explaining machine learning classifiers through diverse counterfactual explanations. In *Proceedings of the 2020 conference on fairness, accountability, and transparency*, pages 607–617.

Mu, J. and Andreas, J. (2020). Compositional explanations of neurons. *Advances in Neural Information Processing Systems*, 33:17153–17163.

Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., and Jaiswal, S. (2017). graph2vec: Learning distributed representations of graphs.

Navarin, N., Sperduti, A., et al. (2017). Approximated neighbours minhash graph node kernel. In *ESANN*, pages 281–286.

Neuhaus, M. and Bunke, H. (2005). A graph matching based approach to fingerprint classification using directional variance. In *International Conference on Audio-and Video-Based Biometric Person Authentication*, pages 191–200. Springer.

Neumann, M., Garnett, R., Bauckhage, C., and Kersting, K. (2016). Propagation kernels: efficient graph kernels from propagated information. *Machine learning*, 102(2):209–245.

Nguyen, T. M., Quinn, T. P., Nguyen, T., and Tran, T. (2022). Explaining black box drug target prediction through model agnostic counterfactual samples. *IEEE/ACM Trans. on Compt. Biology and Bioinformatics*.

Niepert, M., Ahmed, M., and Kutzkov, K. (2016). Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023. PMLR.

Nigam, A., Pollice, R., Krenn, M., dos Passos Gomes, G., and Aspuru-Guzik, A. (2021). Beyond generative models: superfast traversal, optimization, novelty, exploration and discovery (stoned) algorithm for molecules using selfies. *Chemical science*, 12(20):7079–7090.

Niu, C., Song, Y., Song, J., Zhao, S., Grover, A., and Ermon, S. (2020). Permutation invariant graph generation via score-based generative modeling. In *International conference on artificial intelligence and statistics*, pages 4474–4484. PMLR.

Numeroso, D. and Bacciu, D. (2021). Meg: Generating molecular counterfactual explanations for deep graph networks. In *2021 Int. Joint Conf. on Neural Networks*, pages 1–8. IEEE.

Pan, S., Wu, J., and Zhu, X. (2015). Cogboost: Boosting for fast cost-sensitive graph classification. *IEEE Transactions on Knowledge and Data Engineering*, 27(11):2933–2946.

Pan, S., Wu, J., Zhu, X., and Zhang, C. (2014). Graph ensemble boosting for imbalanced noisy graph stream classification. *IEEE transactions on cybernetics*, 45(5):954–968.

Pan, S., Wu, J., Zhu, X., Zhang, C., and Wang, Y. (2016). Tri-party deep network representation. In *25th International Joint Conference on Artificial Intelligence (IJCAI-16)*. AAAI Press/International Joint Conferences on Artificial Intelligence.

Pan, S., Zhu, X., Zhang, C., and Yu, P. S. (2013). Graph stream classification using labeled and unlabeled graphs. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 398–409. IEEE.

Pearl, J. (2009). Causal inference in statistics: An overview.

Pemantle, R. (1992). Vertex-reinforced random walk. *Probability Theory and Related Fields*, 92(1):117–136.

Pereira, T., Nascimento, E., Resck, L. E., Mesquita, D., and Souza, A. (2023). Distill n'explain: explaining graph neural networks using simple surrogates. In *International Conference on Artificial Intelligence and Statistics*, pages 6199–6214. PMLR.

Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710.

Petch, J., Di, S., and Nelson, W. (2021). Opening the black box: the promise and limitations of explainable machine learning in cardiology. *Canadian Journal of Cardiology*.

Pfeifer, B., Saranti, A., and Holzinger, A. (2022). Gnn-subnet: disease subnetwork detection with explainable graph neural networks. *Bioinformatics*, 38(Supplement_2):ii120–ii126.

Pineda, F. J. (1987). Generalization of back-propagation to recurrent neural networks. *Phys. Rev. Lett.*, 59:2229–2232.

Pope, P. E., Kolouri, S., Rostami, M., Martin, C. E., and Hoffmann, H. (2019). Explainability methods for graph convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10772–10781.

Prado-Romero, M. A., Prenkaj, B., and Stilo, G. (2023a). Are generative-based graph counterfactual explainers worth it? In *Joint European Conf. on Machine Learning and Knowl. Disc. in Databases*, pages 152–170. Springer.

Prado-Romero, M. A., Prenkaj, B., and Stilo, G. (2023b). Developing and evaluating graph counterfactual explanation with gretel. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*, WSDM '23, page 1180–1183, New York, NY, USA. Association for Computing Machinery.

Prado-Romero, M. A., Prenkaj, B., and Stilo, G. (2024). Robust stochastic graph generator for counterfactual explanations. In *Proc. of the AAAI Conf. on Artificial Intelligence*, volume 38, pages 21518–21526.

Prado-Romero, M. A., Prenkaj, B., Stilo, G., and Giannotti, F. (2023c). A survey on graph counterfactual explanations: Definitions, methods, evaluation, and research challenges. *ACM Comput. Surv.*

Prado-Romero, M. A., Prenkaj, B., Stilo, G., and Giannotti, F. (2023d). A survey on graph counterfactual explanations: Definitions, methods, evaluation, and research challenges. *ACM Comput. Surv.*

Prado-Romero, M. A. and Stilo, G. (2022). Gretel: Graph counterfactual explanation evaluation framework. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management*, CIKM '22, New York, NY, USA. Association for Computing Machinery.

Prenkaj, B., Villaizán-Vallelado, M., Leemann, T., and Kasneci, G. (2024). Unifying evolution, explanation, and discernment: A generative approach for dynamic graph counterfactuals. In *Proc. of the 30th ACM SIGKDD Conf. on Knowl. Disc. and Data Mining*, pages 2420–2431.

Qu, Z., Gomm, D., and Färber, M. (2024). Greedy and cody: Counterfactual explainers for dynamic graphs. *arXiv preprint arXiv:2403.16846*.

Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., and Beaini, D. (2022). Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515.

Rath, B., Morales, X., and Srivastava, J. (2021). Scarlet: explainable attention based graph neural network for fake news spreader prediction. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 714–727. Springer.

Riesen, K. and Bunke, H. (2008). Iam graph database repository for graph based pattern recognition and machine learning. In *Joint IAPR international workshops on statistical techniques in pattern recognition (SPR) and structural and syntactic pattern recognition (SSPR)*, pages 287–297. Springer.

Rozemberczki, B. and Sarkar, R. (2020). Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models.

Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead.

Satorras, V. G., Hoogeboom, E., and Welling, M. (2021). E (n) equivariant graph neural networks. In *International conference on machine learning*, pages 9323–9332. PMLR.

Scarselli, F., Gori, M., Tsoi, A., Hagenbuchner, M., and Monfardini, G. (2008). The graph neural network model. *IEEE Trans. on Neural Networks*, 20(1):61–80.

Schlichtkrull, M. S., De Cao, N., and Titov, I. (2020). Interpreting graph neural networks for nlp with differentiable edge masking. *arXiv preprint arXiv:2010.00577*.

Schmidt, R., Montani, S., Bellazzi, R., Portinale, L., and Gierl, L. (2001). Cased-based reasoning for medical knowledge-based systems. *International Journal of Medical Informatics*, 64(2-3):355–367.

Schnake, T., Eberle, O., Lederer, J., Nakajima, S., Schütt, K. T., Müller, K.-R., and Montavon, G. (2021). Higher-order explanations of graph neural networks via relevant walks. *IEEE transactions on pattern analysis and machine intelligence*, 44(11):7581–7596.

Schomburg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G., and Schomburg, D. (2004). Brenda, the enzyme database: updates and major new developments. *Nucleic acids research*, 32(suppl_1):D431–D433.

Schütt, K. T., Arbabzadah, F., Chmiela, S., Müller, K. R., and Tkatchenko, A. (2017). Quantum-chemical insights from deep tensor neural networks. *Nature communications*, 8(1):13890.

Shan, C., Shen, Y., Zhang, Y., Li, X., and Li, D. (2021). Reinforcement learning enhanced explainer for graph neural networks. *Advances in Neural Information Processing Systems*, 34:22523–22533.

Shapley, L. S. (1953). *17. A Value for n-Person Games*, pages 307–318. Princeton University Press, Princeton.

Shehzad, A., Xia, F., Abid, S., Peng, C., Yu, S., Zhang, D., and Verspoor, K. (2024). Graph transformers: A survey. *arXiv preprint arXiv:2407.09777*.

Shen, X., Pan, S., Liu, W., Ong, Y. S., and Sun, Q. S. (2018). Discrete network embedding. In *International Joint Conference on Artificial Intelligence 2018*, pages 3549–3555. Association for the Advancement of Artificial Intelligence (AAAI).

Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. (2011). Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9).

Shi, C., Luo, S., Xu, M., and Tang, J. (2021). Learning gradient fields for molecular conformation generation. In *International conference on machine learning*, pages 9558–9568. PMLR.

Shou, Y., Ai, W., Meng, T., and Li, K. (2026). Graph diffusion models: A comprehensive survey of methods and applications. *Computer Science Review*, 59:100854.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

Simonovsky, M. and Komodakis, N. (2018). Graphvae: Towards generation of small graphs using variational autoencoders. In *International conference on artificial neural networks*, pages 412–422. Springer.

Song, Y. and Ermon, S. (2020). Improved techniques for training score-based generative models. *Advances in neural information processing systems*, 33:12438–12448.

Song, Y., Garg, S., Shi, J., and Ermon, S. (2020a). Sliced score matching: A scalable approach to density and score estimation. In *Uncertainty in artificial intelligence*, pages 574–584. PMLR.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. (2020b). Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*.

Sorkun, M. C., Khetan, A., and Er, S. (2019). Aqsoldb, a curated reference set of aqueous solubility and 2d descriptors for a diverse set of compounds. *Scientific data*, 6(1):143.

Sperduti, A. and Starita, A. (1997). Supervised neural networks for the classification of structures. *IEEE transactions on neural networks*, 8(3):714–735.

Sui, Y., Wang, X., Wu, J., Lin, M., He, X., and Chua, T.-S. (2022). Causal attention for interpretable and generalizable graph classification. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pages 1696–1705.

Sundararajan, M., Taly, A., and Yan, Q. (2017). Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR.

Sutherland, J. J., O'Brien, L. A., and Weaver, D. F. (2003). Spline-fitting with a genetic algorithm: A method for developing classification structure-activity relationships. *Journal of Chemical Inf. and Computer Sciences*, 43(6):1906–1915.

Tan, J., Geng, S., Fu, Z., Ge, Y., Xu, S., Li, Y., and Zhang, Y. (2022). Learning and evaluating graph neural network explanations based on counterfactual and factual reasoning. In *Proc. of the ACM Web Conf. 2022*, WWW '22, page 1018–1027.

Tishby, N. and Zaslavsky, N. (2015). Deep learning and the information bottleneck principle. In *2015 ieee information theory workshop (itw)*, pages 1–5. Ieee.

Topping, J., Di Giovanni, F., Chamberlain, B. P., Dong, X., and Bronstein, M. M. (2021). Understanding over-squashing and bottlenecks on graphs via curvature. *arXiv preprint arXiv:2111.14522*.

Trippe, B. L., Yim, J., Tischer, D., Baker, D., Broderick, T., Barzilay, R., and Jaakkola, T. (2022). Diffusion probabilistic modeling of protein backbones in 3d for the motif-scaffolding problem. *arXiv preprint arXiv:2206.04119*.

Tsirtsis, S. and Gomez Rodriguez, M. (2020). Decisions, counterfactual explanations and strategic behavior. *Advances in Neural Information Processing Systems*, 33:16749–16760.

Tsitsulin, A., Mottin, D., Karras, P., Bronstein, A., and Müller, E. (2018). Netlsd: Hearing the shape of a graph. In *Proc. of the 24th ACM SIGKDD Int. Conf. on Knowl. Discovery & Data Mining*. ACM.

Ustun, B., Spangher, A., and Liu, Y. (2019). Actionable recourse in linear classification. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 10–19.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Vedaldi, A. and Soatto, S. (2008). Quick shift and kernel methods for mode seeking. In *European conference on computer vision*, pages 705–718. Springer.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y., et al. (2017). Graph attention networks. *stat*, 1050(20):10–48550.

Veličković, P., Ying, R., Padovano, M., Hadsell, R., and Blundell, C. (2019). Neural execution of graph algorithms. *arXiv preprint arXiv:1910.10593*.

Verenich, I., Dumas, M., La Rosa, M., and Nguyen, H. (2019). Predicting process performance: A white-box approach based on process models. *Journal of Software: Evolution and Process*, 31(6):e2170.

Verma, S., Armgaan, B., Medya, S., and Ranu, S. (2024a). Induce: Inductive counterfactual explanations for graph neural networks. *Transactions on Machine Learning Research*.

Verma, S., Boonsanong, V., Hoang, M., Hines, K., Dickerson, J., and Shah, C. (2024b). Counterfactual explanations and algorithmic recourses for machine learning: A review. *ACM Computing Surveys*, 56(12):1–42.

Verma, S. and Zhang, Z.-L. (2017). Hunt for the unique, stable, sparse and fast feature learning on graphs. In *Advances in Neural Information Processing Systems*, volume 30.

Vermeire, T., Brughmans, D., Goethals, S., de Oliveira, R. M. B., and Martens, D. (2022). Explainable image classification with evidence counterfactual. *Pattern Analysis and Applications*, 25(2):315–335.

Vignac, C., Krawczuk, I., Siraudin, A., Wang, B., Cevher, V., and Frossard, P. (2022). Digress: Discrete denoising diffusion for graph generation. *arXiv preprint arXiv:2209.14734*.

Vinyals, O., Bengio, S., and Kudlur, M. (2015). Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*.

Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R., and Borgwardt, K. M. (2010). Graph kernels. *The Journal of Machine Learning Research*, 11:1201–1242.

Von Kügelgen, J., Karimi, A.-H., Bhatt, U., Valera, I., Weller, A., and Schölkopf, B. (2022). On the fairness of causal algorithmic recourse. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 9584–9594.

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

Vu, M. N. and Thai, M. T. (2022). On the limit of explaining black-box temporal graph neural networks. *arXiv preprint arXiv:2212.00952*.

Wale, N., Watson, I. A., and Karypis, G. (2008). Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375.

Wang, L., Huang, C., Ma, W., Cao, X., and Vosoughi, S. (2021a). Graph embedding via diffusion-wavelets-based node feature distribution characterization. In *Proc. of the 30th ACM Int. Conf. on Information & Knowl. Management*, pages 3478–3482.

Wang, X. and Shen, H.-W. (2022). Gnninterpreter: A probabilistic generative model-level explanation for graph neural networks. *arXiv preprint arXiv:2209.07924*.

Wang, X., Wu, Y., Zhang, A., He, X., and Chua, T.-S. (2021b). Towards multi-grained explainability for graph neural networks. *Advances in neural information processing systems*, 34:18446–18458.

Wang, Z., Chen, J., and Chen, H. (2021c). Egat: Edge-featured graph attention network. In *International Conference on Artificial Neural Networks*, pages 253–264. Springer.

Wei, X., Liu, Y., Sun, J., Jiang, Y., Tang, Q., and Yuan, K. (2022). Dual subgraph-based graph neural network for friendship prediction in location-based social networks. *ACM Trans. on Knowl. Disc. from Data*.

Weininger, D. (1988). Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36.

Wellawatte, G. P., Seshadri, A., and White, A. D. (2022). Model agnostic generation of counterfactual explanations for molecules. *Chemical science*, 13(13):3697–3705.

Winn, J., Criminisi, A., and Minka, T. (2005). Object categorization by learned universal visual dictionary. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pages 1800–1807. IEEE.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. (2019). Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. Pmlr.

Wu, H., Chen, W., Xu, S., and Xu, B. (2021a). Counterfactual supporting facts extraction for explainable medical record based diagnosis with graph network. In *Proc. of the 2021 Conf. of the North American Chapter of the Assoc. for Compt. Linguistics: Human Language Technologies*, pages 1942–1955.

Wu, Y.-X., Wang, X., Zhang, A., He, X., and Chua, T.-S. (2022). Discovering invariant rationales for graph neural networks. *arXiv preprint arXiv:2201.12872*.

Wu, Z., Jain, P., Wright, M., Mirhoseini, A., Gonzalez, J. E., and Stoica, I. (2021b). Representing long-range context for graph neural networks with global attention. *Advances in neural information processing systems*, 34:13266–13279.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. (2020). A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24.

Wu, Z., Ramsundar, B., Feinberg, E. N., Gomes, J., Geniesse, C., Pappu, A. S., Leswing, K., and Pande, V. (2018). Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530.

Xia, W., Lai, M., Shan, C., Zhang, Y., Dai, X., Li, X., and Li, D. (2022). Explaining temporal graph models through an explorer-navigator framework. In *The Eleventh International Conference on Learning Representations*.

Xie, J., Liu, Y., and Shen, Y. (2022). Explaining dynamic graph neural networks via relevance back-propagation. *arXiv preprint arXiv:2207.11175*.

Xu, H., Sang, S., Yao, H., Herghelegiu, A. I., Lu, H., Yurkovich, J. T., and Yang, L. (2021). Aprile: exploring the molecular mechanisms of drug side effects with explainable graph neural networks. *BioRxiv*, pages 2021–07.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2018). How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Xu, M., Yu, L., Song, Y., Shi, C., Ermon, S., and Tang, J. (2022). Geodiff: A geometric diffusion model for molecular conformation generation. *arXiv preprint arXiv:2203.02923*.

Xuanyuan, H., Barbiero, P., Georgiev, D., Magister, L. C., and Liò, P. (2023). Global concept-based interpretability for graph neural networks via neuron analysis. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 10675–10683.

Yanardag, P. and Vishwanathan, S. (2015). Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1365–1374.

Yang, H., Pan, S., Zhang, P., Chen, L., Lian, D., and Zhang, C. (2018). Binarized attributed network embedding. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 1476–1481. IEEE.

Yang, L., Zhang, Z., Song, Y., Hong, S., Xu, R., Zhao, Y., Zhang, W., Cui, B., and Yang, M.-H. (2023a). Diffusion models: A comprehensive survey of methods and applications. *ACM computing surveys*, 56(4):1–39.

Yang, Q., Ma, C., Zhang, Q., Gao, X., Zhang, C., and Zhang, X. (2023b). Interpretable research interest shift detection with temporal heterogeneous graphs. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*, pages 321–329.

Yang, R., Srivastava, P., and Mandt, S. (2023c). Diffusion probabilistic modeling for video generation. *Entropy*, 25(10):1469.

Ying, Z., Bourgeois, D., You, J., Zitnik, M., and Leskovec, J. (2019). Gnnexplainer: Generating explanations for graph neural networks. *Adv. in Neural Inf. Processing Systems*, 32.

You, J., Ying, R., Ren, X., Hamilton, W., and Leskovec, J. (2018). Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*, pages 5708–5717. PMLR.

Yu, J., Cao, J., and He, R. (2022). Improving subgraph recognition with variational graph information bottleneck. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 19396–19405.

Yu, J., Xu, T., and He, R. (2021). Towards the explanation of graph neural networks in digital pathology with information flows. *arXiv preprint arXiv:2112.09895*.

Yu, J., Xu, T., Rong, Y., Bian, Y., Huang, J., and He, R. (2020). Graph information bottleneck for subgraph recognition. *arXiv preprint arXiv:2010.05563*.

Yuan, C., Zhao, K., Kuruoglu, E. E., Wang, L., Xu, T., Huang, W., Zhao, D., Cheng, H., and Rong, Y. (2025). A survey of graph transformers: Architectures, theories and applications. *arXiv preprint arXiv:2502.16533*.

Yuan, H., Tang, J., Hu, X., and Ji, S. (2020). Xgnn: Towards model-level explanations of graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 430–438.

Yuan, H., Yu, H., Gui, S., and Ji, S. (2022). Explainability in graph neural networks: A taxonomic survey. *IEEE Trans. on Pattern Analysis and Machine Intelligence*.

Yuan, H., Yu, H., Wang, J., Li, K., and Ji, S. (2021). On explainability of graph neural networks via subgraph explorations. In *Int. Conf. on Machine Learning*, pages 12241–12252. PMLR.

Zemni, M., Chen, M., Zablocki, E., Ben-Younes, H., Pérez, P., and Cord, M. (2023). Octet: Object-aware counterfactual explanations. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 15062–15071.

Zhang, D., Yin, J., Zhu, X., and Zhang, C. (2018). Network representation learning: A survey. *IEEE transactions on Big Data*, 6(1):3–28.

Zhang, M., Qamar, M., Kang, T., Jung, Y., Zhang, C., Bae, S.-H., and Zhang, C. (2023). A survey on graph diffusion models: Generative ai in science for molecule, protein and material. *arXiv preprint arXiv:2304.01565*.

Zhang, S., Liu, Y., Shah, N., and Sun, Y. (2022a). Gstarx: Explaining graph neural networks with structure-aware cooperative games. *Advances in neural information processing systems*, 35:19810–19823.

Zhang, Y., Defazio, D., and Ramesh, A. (2021). Relex: A model-agnostic relational model explainer. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pages 1042–1049.

Zhang, Z., Liu, Q., Wang, H., Lu, C., and Lee, C. (2022b). Protgnn: Towards self-explaining graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 9127–9135.

Zhao, L., Song, Y., Zhang, C., Liu, Y., Wang, P., Lin, T., Deng, M., and Li, H. (2019). T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE transactions on intelligent transportation systems*, 21(9):3848–3858.

Zhou, H., He, L., Zhang, Y., Shen, L., and Chen, B. (2022). Interpretable graph convolutional network of multi-modality brain imaging for alzheimer's disease diagnosis. In *2022 IEEE 19th International Symposium on Biomedical Imaging (ISBI)*, pages 1–5. IEEE.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI open*, 1:57–81.

Zhu, X., Zhang, Y., Zhang, Z., Guo, D., Li, Q., and Li, Z. (2022). Interpretability evaluation of botnet detection model based on graph neural network. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6. IEEE.