Journal of Statistical Theory and Applications (2025) 24:469–488
https://doi.org/10.1007/s44199-025-00118-x

RESEARCH ARTICLE

BetaExplainer: A Probabilistic Method to Explain Graph Neural Networks

Whitney Sloneker[1] · Shalin Patel[2] · Hung-Jen Wang[2,3] · Lorin Crawford[1,4] · Ritambhara Singh[1,2]

Received: 13 December 2024 / Accepted: 2 May 2025 / Published online: 3 June 2025

Abstract

Graph neural networks (GNNs) are powerful tools for conducting inference on graph data but are often seen as “black boxes" due to difficulty in extracting meaningful subnetworks driving predictive performance. Many interpretable GNN methods exist, but they cannot quantify uncertainty in edge weights and suffer in predictive accuracy when applied to challenging graph structures. In this work, we proposed BetaExplainer which addresses these issues by using a sparsity-inducing prior to mask unimportant edges during model training. To evaluate our approach, we examine various simulated data sets with diverse real-world characteristics. Not only does this implementation provide a notion of edge importance uncertainty, it also improves upon evaluation metrics for challenging datasets compared to state-of-the art explainer methods.

Keywords Deep learning · Graph neural networks · Probabilistic models · Explainability · Variational inference

✉ Whitney Sloneker
whitney_sloneker@brown.edu

✉ Ritambhara Singh
ritambhara_singh@brown.edu

[1] Center for Computational Molecular Biology, Brown University, Providence, Rhode Island, USA

[2] Department of Computer Science, Brown University, Brown University, Providence, Rhode Island, USA

[3] Department of Applied Mathematics, Brown University, Providence, Rhode Island, USA

[4] Microsoft Research, Cambridge, Massachusetts, USA

1 Introduction

Relational data occur in a variety of domains, such as social graphs [1], chemical structures [2], physical systems [1], gene-gene interactions [1], and epidemiological modeling [3]. These data are best represented by graphs that effectively model their relationships, such as chemical bonds in drug molecules that affect toxicity or treatment efficacy [1], or personal interactions in social networks indicating contact [2]. Although graph information represents these datasets more accurately by incorporating node features (i.e., chemical weight for molecules) and node interactions through edges (i.e., chemical bonds) [1], large-scale modeling to learn their patterns can be challenging if the graphs are complex [4, 5].

Embedding methods such as Graphlets [6] and DeepWalk [7] were developed to address these challenges. However, they may oversimplify complex graphical features by summarizing the graph and ignoring node features [8]. As a result, graph neural networks (GNNs) have been widely adopted in the machine learning community to model graph-based datasets because they incorporate edge structure and node features directly [9, 10].

GNN models have broad applications, such as capturing the complexity of traffic dynamics, approximating NP-hard graph combinatorial analysis, and learning real-world graphs such as molecule structures [1]. However, like other deep learning models, they can be difficult to explain [11]. It is challenging to extract important edges that the GNN learns to make accurate predictions [11]. Determining these important edges is needed for hypothesis development [11], such as answering "what chemical bonds might determine the prediction of toxicity in a molecule?", or "what features indicate email spam?" [12]. As a result, it is critical to develop GNN explanation methods to understand the model predictions that highlight the important edges of the graph.

Many methods have been created to explain GNNs, that is, highlight the important edges for predictions, but their performances vary widely depending on the underlying properties of the data or the GNN model. For example, gradient-based explainability methods struggle to produce accurate edge explanations when deep learning models experience gradient saturation [13]. Transformer models such as Graphormer [14] and GraphTrans [15] perform well, but must be specifically chosen as layers for GNN classification prior to training the model. Thus, they only explain models that have already incorporated transformer-based architectures. Data properties can also influence the explainer's performance. A recent benchmark study [16] found that many methods (Grad [17], GradCAM [18], GuidedBP [19], Integrated Gradients [20], GNNExplainer [21], PGExplainer [22], SubgraphX [23], and PGM-Explainer [24]) struggle against challenging data properties. For example, if the underlying graph of the data is heterophilic[1] or a low proportion of node features are critical for classification, the existing explainer methods struggle to produce accurate edge explanations from the GNN model.

[1] A highly heterophilic graph is one where edges tend to connect nodes of different classes. Its opposite, a homophilic graph, is one in which edges tend to connect nodes of the same class.

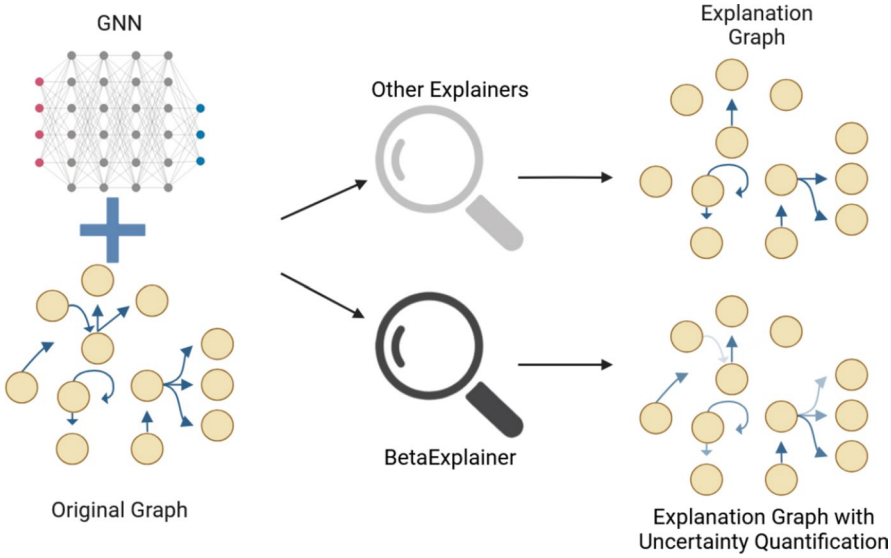

Fig. 1 BetaExplainer returns a mask for the important edges of a graph for a GNN classification. As it learns a probabilistic model to represent these important edges, the mask estimates a level of uncertainty in importance of each edge to the GNN

The study shows that out of all the methods examined, only GNNExplainer, PGExplainer, and SubgraphX function as effective edge explainers for GNN models [16]. However, PGExplainer often underperforms most methods in generating accurate edge explanations for the simulated datasets [16]. On the other hand, SubgraphX has a robust performance [16]. However, it cannot rank the edge importance as each highlighted edge is only denoted as 0 (as unimportant) or 1 (as important) [13]. This ranking is essential to hypothesis generation as it allows researchers to focus on the most highly ranked edges, easing downstream analysis. For instance, in the case of exploring edges representing gene-gene interactions for biological datasets, edges ranked based on their importance allow us to focus on experimentally confirming only the most likely interactions, saving the time and monetary cost required for wet lab experiments [25]. Finally, GNNExplainer performs relatively well on the proposed synthetic datasets [16] while also returning importance scores that can rank the relevant edges of the GNN model [13]. As a result, SubgraphX and GNNExplainer best represent state-of-the-art edge explainers for GNNs. Even so, their struggles to produce accurate edge explanations suggest a knowledge gap for explainer methods on the tested challenging datasets.

We explored another challenging setting for GNN explainer methods – explaining a GNN model for graph datasets constructed with sparse node features. A dataset with sparse node features is one where many node features are zeros. Sparse node feature datasets are common in various real-world domains, especially in the now-emerging single-cell gene expression data (or scRNA-seq). These datasets are notoriously sparse, and any gene-gene correlation or interaction graphs created from them will result in graphs with low informative node features [26] [25].

We hypothesize that existing explainers will also struggle to accurately return edge explanations for the sparse node feature dataset.

We propose a new method, BetaExplainer, that uses a probabilistic distribution to determine the important edges from a GNN model. BetaExplainer learns a probabilistic edge mask to maximize the similarity of the output of the trained GNN on a masked graph to its original output through statistical inference to approximate which edges are most important (Fig. 1). This probabilistic approach allows us to produce edge importance scores with uncertainty quantification and rank edges by the order of score confidence. We have evaluated BetaExplainer on seven simulated datasets with various challenging underlying data properties that explainers struggle to adapt to, including a heterophilic graph and a sparse node feature dataset. The results demonstrate that BetaExplainer significantly outperforms existing methods in being faithful to the underlying graph importance on five out of seven simulated datasets and improves accuracy compared to state-of-the-art methods on graph datasets with sparse features. Finally, it can achieve a faithful explanation of a real-world dataset on a small set of edges. It also conveys a notion of uncertainty in edge importance for its explanations, guiding downstream analysis.

2 Methods

2.1 BetaExplainer Algorithmic Framework

Given a trained GNN model f, graph input $G = (V, E)$ with the set of nodes or vertices defined as V and edges E with edge e_{ij} connecting vertices v_i and v_j, node features X, and model output on the input graph and node features $f(X, G)$, we define a Beta distribution prior $P(M)$ on the edge mask M of the input graph. BetaExplainer learns the posterior Beta distribution $P(M \mid f(X, E))$ of the edge mask by comparing the results of the masked-out graph G_s and a full set of node features or $f(X, G_s)$ to the original output on the unmasked graph G and all node features or $f(X, G)$ (Fig. 2) where the likelihood over the GNN or $p(f(X, E))$ is a Bernoulli distribution described by

$$P(f(X,E) \mid M) = \begin{cases} p & \text{if } M \geq 0.5, \\ 1-p & \text{if } M < 0.5. \end{cases} \tag{1}$$

Based on the Kullback–Leibler (KL) divergence between the new and original outputs, BetaExplainer updates the edge mask probabilities to increase or decrease each edge importance value in the edge mask as applicable. We optimize evidence lower bound (ELBO) to learn the final edge mask. This edge mask conveys the importance of each edge as a probabilistic importance score.

BetaExplainer has two major benefits: (1) using a probabilistic framing allows us to convey edge importance for easy interpretation and edge rankings while also conveying uncertainty in edge importance, and (2) users may choose distributional parameters most relevant to the underlying data to improve performance by better representing the underlying distribution of edge importance. For this work, we choose the Beta distribution, which is described by

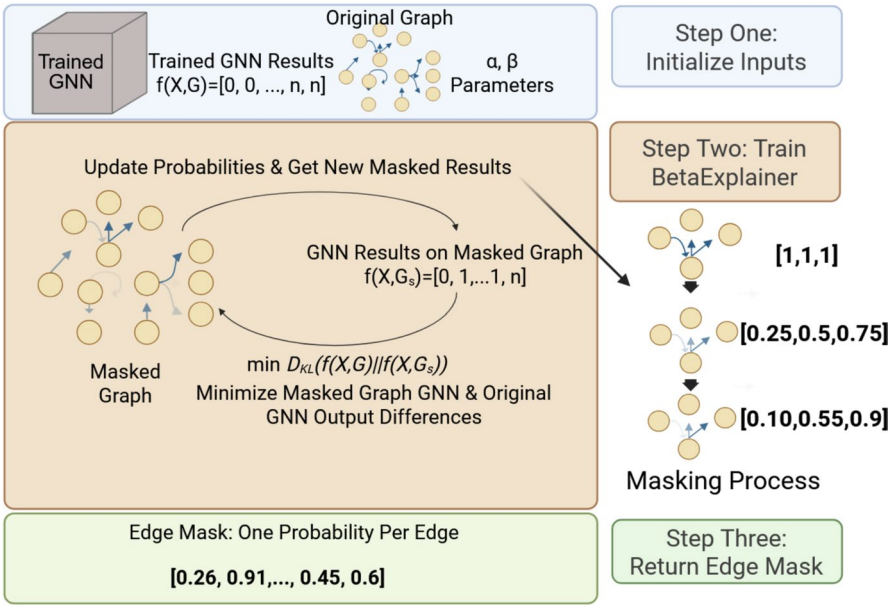

Fig. 2 Given a trained GNN, the original graph, and Beta distribution parameters α and β, BetaExplainer is trained by learning the masked-out graph minimizing the KL-Divergence Loss between the model output on the masked-out graph and original graph. It will return the learned edge mask representing a probabilistic importance score for each edge when complete

$$P(M_{ij} \mid \alpha, \beta) = \frac{M_{ij}^{\alpha-1}(1 - M_{ij})^{\beta-1}\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \tag{2}$$

where α and β denote are real valued shape parameters. Here, $P(M_{ij} \mid \alpha, \beta)$ is the probability that the mask importance for edge e_{ij} is value M_{ij}. As edges of a graph can mostly be described by a Bernoulli distribution measuring binary outcomes - indicating importance or unimportance - the Beta distribution functions as the conjugate prior[2] of the Bernoulli making it a reasonable choice to describe edge importance uncertainty. Equivalently, if the prior edge importance described by the edge mask $P(M)$ is a Beta distribution and the distribution $P(A \mid M)$, indicating whether each edge is important given the Bernoulli distribution, denotes the mask, then the posterior distribution of edge importance $P(M \mid A)$ will also be a Beta distribution.

We assume that the learned Beta distributions for each edge $e \in E$ are independent of each other. Thus, the learned distribution of edge importance for any edge $e_{ij} \in E$ is equivalent to its learned importance for the full mask M through mean field variational inference. Therefore, we may learn the importance of all edges simultaneously. Thus, the ELBO may be calculated as follows with respect to the original output:

[2] When the prior and posterior distributions come from the same family, it is called a conjugate prior.

$$\mathbb{E}[\log(P(G_s, f(X,G)|M) - \log(P(f(X,G)))] \tag{3}$$

The ELBO is the lower bound for $P(G_s, f(X,G)|M)$, or

$$P(G_s, f(X,G)|M) \geq ELBO \tag{4}$$

This expression indicates that the difference between the two expressions can be no less than zero and by maximizing the ELBO, the following holds:

$$\log(P(G_s, f(X,G)|M) - ELBO = KL(\log(P(f(X,G))) \parallel P(G_s, f(X,G)|M)) \tag{5}$$

Maximizing the ELBO indirectly minimizes the KL divergence between the model output on the masked edges and the original output, learning the optimal mask. While we could calculate the closed form of the Bayes theorem directly in this case, we chose to approximate the true distribution with a variational family instead to increase BetaExplainer's applicability to large-scale real-world datasets. Thus, BetaExplainer's algorithm is formulated as follows:

Algorithm 1 BetaExplainer Model Set Up

Require: $X, G = (V, E), e_{ij} \in E, \alpha > 0, \beta > 0$ {Require positive Beta distribution parameters and vertices in the graph}
Require: $c \sim f(X,G)$ {Return model outputs given the input data.}
$Z \leftarrow z$ {Define number of training epochs Z as positive integer z}
$T \leftarrow 1$ {Initialize epoch tracker T for training}
while $T \leq Z$ **do**
$\alpha \leftarrow \hat{\alpha}$, $\beta \leftarrow \hat{\beta}$ {Update parameters α and β on the original graph based on the results of the prior iteration.}
$M_{ij} \sim Beta(\alpha, \beta) \forall e_{ij} \in E$ {Determine edge weights from Beta distribution}
$y \leftarrow f(X, G, M)$ {Return the model output on input dataset with generated weights for edges}
$c \sim y$ {Compare outputs on weighted dataset to original with KL divergence loss to update parameters}
$T \leftarrow T + 1$ {Update epoch tracker.}
end while
return $M_{ij} \forall e_{ij}$ {Return weights representing edge probabilities over all edges}

BetaExplainer uses the `Pyro` framework for variational inference to develop the edge importance model, and the `pytorch_geometric` framework to train all of the GNN models used.

2.2 Baselines

Due to methodological similarities, the first baseline we compare BetaExplainer to is the state-of-the-art method GNNExplainer [21]. GNNExplainer performs well on various node and graph classification datasets, suggesting it is a strong baseline [13]. While BetaExplainer and GNNExplainer have similar optimization goals, the different training algorithms ensure that BetaExplainer has certain beneficial properties. GNNExplainer randomly initializes the edge mask and then directly optimizes the Bernoulli distribution of edge importance, requiring a re-parameterization trick.

In contrast, BetaExplainer directly shrinks edges by using the black-box variational inference. Furthermore, GNNExplainer does not directly drop out parameters by forcing them to be zero; based on the distribution learned, BetaExplainer does this as needed.

The next baseline we compare BetaExplainer against is SubgraphX [23], because it is a well-performing edge explainer like GNNExplainer [21]. SubgraphX uses Markov Chain Tree Search to determine the subgraph that achieves a Shapley value, suggesting that the model results on the subgraph are similar to the original results. Although similar to BetaExplainer - both learn the subgraph that returns similar model results to the initial - it does not allow for a notion of uncertainty[13, 23].

Finally, BetaExplainer gives us access to properties, such as incorporating prior information through hyperparameters. These priors allow the proposed method to easily adapt to challenging data properties such as a heterophilic graph or highly sparse node features. Furthermore, as BetaExplainer directly learns a distribution, it can convey a notion of uncertainty in edge importance. We hypothesize these features will benefit BetaExplainer on challenging datasets.

2.3 Experimental Setup

2.3.1 Datasets

Argawal et al. [16] propose that standardized methods to evaluate GNN explainability lack key characteristics. Limitations include few datasets with a notion of ground truth needed to measure explainer performance and under-represented real-world properties. To address these challenges, they developed the ShapeGGen simulator, which generates a variety of datasets with real-world properties and associated known ground truth. This simulator returns a diverse set of graph datasets given defined parameters. A house-shaped motif makes up the ground truth (i.e., all important edges) and generates 1200 subgraphs. Once generated, these subgraphs are connected so that each node has one or two ground truth motifs in its 1-hop neighborhood.[3] The node's class of two is determined by the number of motifs in this neighborhood (zero if there is one motif and one if there are two). This structure makes up the first dataset with no challenging properties, SG-BASELINE, and all remaining graphs are some modification of this baseline.

The graphs with challenging properties are created by modifying one property of SG-BASELINE at once. Where the baseline graph is homophilic, in this case, indicating that nodes sharing an edge tended to contain the same number of motifs in their neighborhood and thus are of the same class, a graph can also be heterophilic. For the ShapeGGen simulator datasets, a heterophilic graph would be one where it is more likely that nodes with one motif in their 1-hop neighborhood will connect to nodes with two motifs in their 1-hop neighborhood. The second potentially

[3] A node's l-hop neighborhood is the set of nodes whose shortest path to the original node contains no more than l edges.

challenging property is based on how correlated node class is to protected node features and how likely these features will be flipped. If not altered, it is assumed there is no correlation, and there is a 50% likelihood the simulator would flip them. This setting ensures that the graph is entirely fair[4] since there is no added node feature bias which would affect the model but not actual node class [16]. Finally, the proportion of informative node features - or features correlated with the node class - to the total number of features may change where the baseline proportion of important features to total features is 4:11. The simulator may increase or decrease this ratio.

The resulting set of datasets examined is as follows, where each dataset alters only one property in relation to the baseline:

- SG-BASE: A baseline dataset that is a large and homophilic graph with house ground-truth motifs, which is adapted to form the remaining datasets.
- SG-HETEROPHILIC: A modified version of the baseline dataset with a heterophilic graph.
- SG-UNFAIR: A modified version of the baseline dataset with a strongly unfair ground truth wherein protected node features are negatively correlated with the node class, and there is a 75% chance node features will be flipped
- SG-MOREINFORM: A modified version of the baseline dataset with a high proportion of important to total features, 8:11.
- SG-LESSINFORM: A modified version of the baseline dataset with a low proportion of importance to total features, 4:21.

While these datasets explore a variety of critical properties, the simulator does not return a sparse node feature dataset or dataset with a high proportion of zero node features with respect to the total number with known ground truth. As a result, there are no benchmarking results on how the explainer methods respond to this potentially challenging feature. Real-world datasets, particularly for biological applications, may have highly sparse features such as single-cell gene expression datasets [26]. This makes analyzing gene-gene interactions computationally, needed to ease wet lab analysis, more challenging [25]. We hypothesize that sparse node features will influence explainer performance, which inspired us to include a second set of datasets: a baseline dataset at two different levels of sparsity.

To examine the effects of sparsity on BetaExplainer, we needed a dataset with some ground truth gene-gene interaction graph. Otherwise, it would be unclear whether the method correctly selects important edges. We chose the SERGIO gene expression simulator [27] as no ShapeGGen parameters allow us to control for this dataset explicitly. The SERGIO gene expression simulator is a widely used tool in the field of gene-gene interaction inference, capable of simulating the gene expression of a set of single cells using the chemical Langevin equation. This method allows the simulator to capture how regulator expression changes affect regulated gene expression, resulting in a gene regulatory network (GRN). Once SERGIO runs a set of simulations long enough to achieve steady-state, it samples gene expression

[4] Fairness measures the similarity of the model results on the data components deemed necessary to the model results on the original data/graph.

from the graph to simulate a single-cell dataset. Specifically for our purposes, we use two cell types or classes, 100 genes per cell, and 1000 cells per cell type. Once sampled, we applied 25% and 50% random sparsity to the original dataset. This means we have two datasets with sparsity and known important edges - or the GRN that governs the cell expression (Fig. 8). Unlike the first set of datasets, these will both be graph classification problems: given a graph governing gene interactions and gene expression, a GNN predicts cell type.

However, in the real-world, the full groundtruth GRN may be unknown, suggesting simulating methods of approximating this underlying GRN are needed. We choose to create a graph based on genes that have a correlation of at least 0.35, indicating that they are often expressed together, for each dataset. Due to limitations such as sparsity and regulation through intermediary genes, this graph contains a mix of true and false edges but is unable to capture all true gene-gene interactions while maintaining a computationally tractable graph. This allows us to approximate explainer accuracy on what it is given, though all explainers are unable to determine the importance of unseen edges.

Finally, we applied BetaExplainer to the Texas [28] dataset to examine its potential on real-world datasets, specifically one that has a heterophilic graph [29]. This dataset uses nodes to represent websites associated with the Computer Science Department of a Texas University, divided into five classes denoting website affiliation (student, course, project, staff, or faculty) [28]. Edges represent links between websites, and node features represent the sites through a bag-of-words method [28].

2.3.2 Implementation

All trained GNN models used the Adam optimizer and cross-entropy loss, but model architecture (Fig. 7) and parameters (Table 3, in supplementary material) vary to optimize the train and test accuracy for each dataset.

We initialize BetaExplainer with the node features and edge index of a given dataset, the α and β parameters needed to initialize the Beta distribution, and the original model trained to classify outputs on the input data with epochs, learning rates, α, and β hyperparameters chosen based on the most balanced results across metrics (Table 4, in supplementary material). Similarly, we chose the parameters resulting in the best performance for the GNNExplainer and the baselines (Tables 4 and 5, in supplementary material).

2.3.3 Metrics

We judge whether BetaExplainer returned more important edges than GNNExplainer and SubgraphX through accuracy metrics (specifically accuracy and F1 Scores to determine how well BetaExplainer returns important edges while ignoring unimportant edges) and unfaithfulness (to capture the similarity of model output on edges the explainer deems important to model output on all edges). For the accuracy metrics, calculation details varied depending on the dataset used. For the ShapeGGen [16] datasets, we focus on the best-performing subgraph since this was the method chosen for the simulator [16] and the whole graph for the SERGIO [27]

datasets for all analysis - qualitative (such as explanation graphs) and quantitative (the edge mask probability distributions and metrics). Letting P denote precision and R represent recall, we consider the F1 score which is computed as the following

$$\frac{2PR}{P+R}. \tag{6}$$

Accuracy was another area in which datasets differ. Given that TP represents the number of true positives, TN the true negatives, FP the false positives, and FN the false negatives, we used the traditional accuracy calculation for the SERGIO datasets:

$$\frac{TP+TN}{TP+TN+FP+FN}. \tag{7}$$

We used two calculations for the number of false negatives: the first, containing all false negatives on the ground-truth elements in the input graph plus the missing ground-truth elements in the said graph, and the second, with just false negatives on the ground-truth elements for these datasets. We chose this method to capture explainer limitations as explainers may only analyze given data while ensuring the resulting metrics make sense. For the remaining datasets, to better mimic the ShapeGGen simulator [16], we used the Jaccard Index:

$$\frac{TP}{TP+FP+FN+1e-9} \tag{8}$$

While the $1e-9$ term is not strictly part of the Jaccard Index, the simulator incorporated it to avoid division by zero errors while maintaining rounding-accurate results, so we chose to include it as well [16]. The final metric calculated is unfaithfulness [16], or

$$1-\exp(-KL(f(X,G)||f(X,G_s))) \tag{9}$$

KL represents the KL divergence between the original GNN output given the full dataset versus the GNN output on the subgraph the explainer deems essential.

We calculated all metrics across ten random seeds, reporting the mean and standard error and ensuring randomness played less of a role in our results. We observed BetaExplainer's edge mask probability distribution of true and false positives over multiple runs with the same seed to best mimic the simulation structure and ensure randomness did not affect runs strongly. Visualizing the best-performing subgraph or graph over these runs for each explainer and the BetaExplainer edge mask probability distributions for true positives and true negatives also provided means to evaluate the model. To demonstrate the uncertainty quantification that BetaExplainer provides, we also displayed a graph weighing each edge based on probability modified through a variation of min-max scaling (supplementary) to clarify the range of probabilities taken.

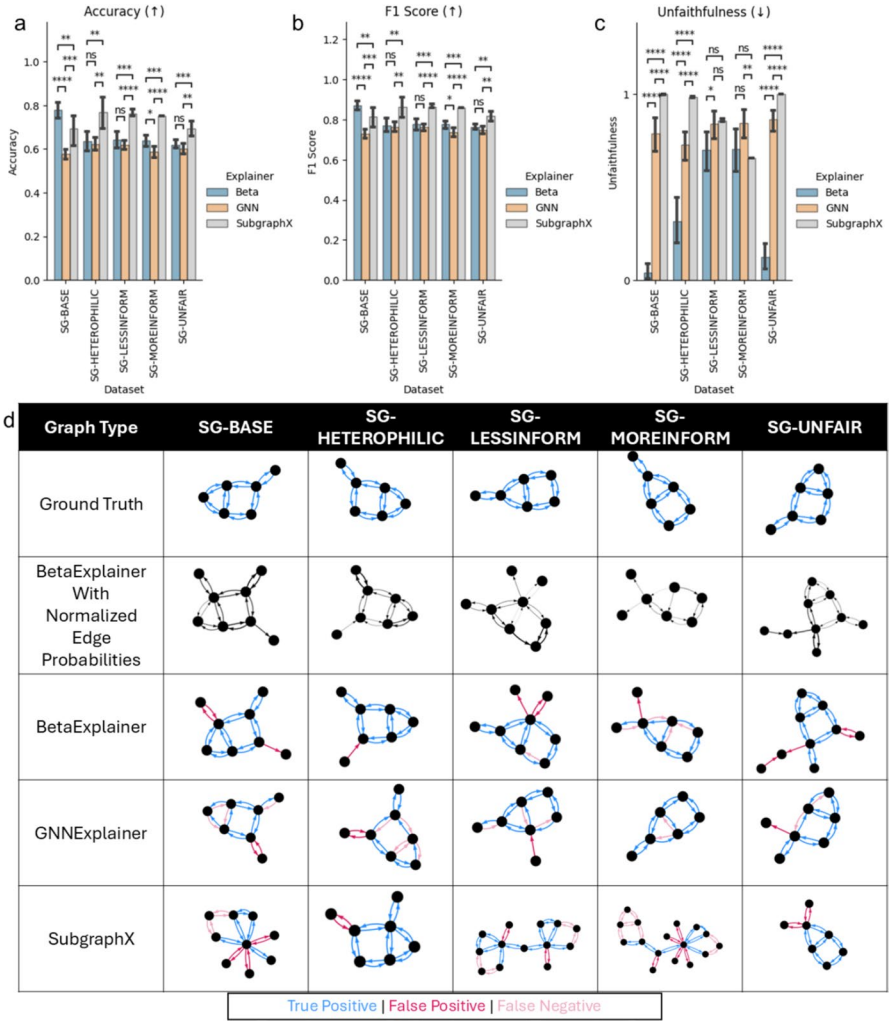

Fig. 3 We calculate the mean and standard errors of the **a** Accuracy, **b** F1 Score, and **c** unfaithfulness results and whether explainer differences are significant (ns: $0.05 < p \leq 1$, *: $0.01 < p \leq 0.05$, **: $0.001 < p \leq 0.01$, ***: $0.0001 < p \leq 0.001$, and ****: $p \leq 0.0001$). We graph the best subgraphs for the datasets for each explainer versus the groundtruth **d**, denoting true positive (blue), false positive (red), and false negative (pink) edges and weighting BetaExplainer edges by probabilities

3 Results

3.1 BetaExplainer Performs Well on Simulated Datasets With Challenging Real-World Properties

We show that BetaExplainer achieves a better Jaccard Index, F1 Score, and unfaithfulness score than GNNExplainer on the five simulated datasets and better

unfaithfulness than SubgraphX on four of the five datasets from [16], with significant improvements for many of these comparisons. We used the Mann–Whitney U test to calculate the p value (Fig. 3 a-c) to test the significance of performance improvement. Explainers, particularly when faced with challenging properties such as heterophilic graphs, tend to generate unfaithful explanation graphs [16], suggesting that these improvements are relevant. BetaExplainer minimizes KL divergence between the masked and original GNN outputs, aligning closely with the unfaithfulness metric. This likely explains the decreased unfaithfulness score for BetaExplainer on most datasets and justifies the choice of our formulation. While BetaExplainer does not achieve better unfaithfulness than SubgraphX on SG-MOREINFORM, this dataset has more informative features than SG-BASELINE. These features may negate the need for informative priors provided by BetaExplainer, particularly as a hyperparameter sweep on SG-BASE suggests well-chosen alpha and beta parameters increase accuracy metrics while decreasing unfaithfulness (Supplementary Fig. 10a-e and h-l). Furthermore, SG-MOREINFORM may not fully represent the challenges of real-world datasets, which are more likely to grapple with measurement errors.

Next, we comprehensively investigate BetaExplainer's performance compared to GNNExplainer and SubgraphX. We visualize the ground-truth sub-graphs and the explanation output for all the methods in Fig. 3d. We see that BetaExplainer generally returns more edges as important than GNNExplainer but balances the precision-recall trade-off well while maintaining a higher true positive rate than this baseline. SubgraphX returns a similar number of edges as BetaExplainer but lacks the ranked scores for relevant edges. Since BetaExplainer is a probabilistic model, unlike SubgraphX, we can obtain the probability distribution of the edge mask scores. Figures. 4a-e plot the empirical cumulative distribution

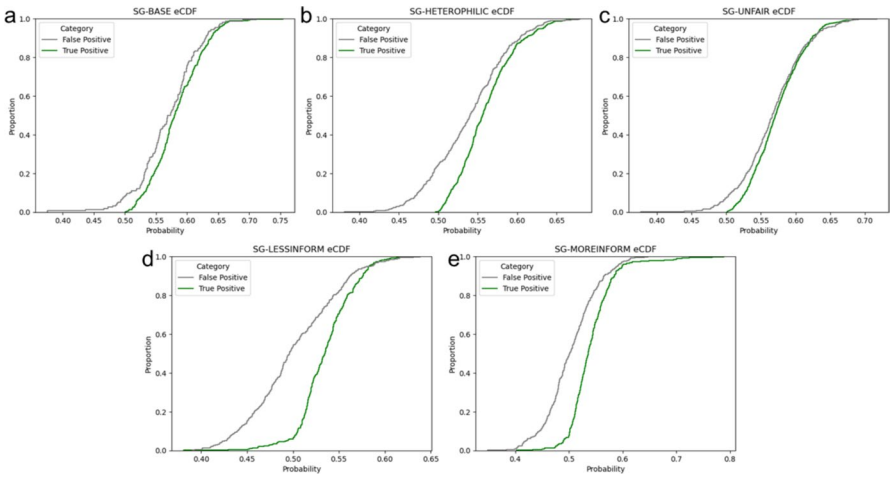

Fig. 4 We plot the empirical cumulative distribution (eCDF) of BetaExplainer probabilities for true and false positives with respect to the groundtruth, noting that the true positives tend to be associated with higher probabilities than false positives (a, b, c, d, e)

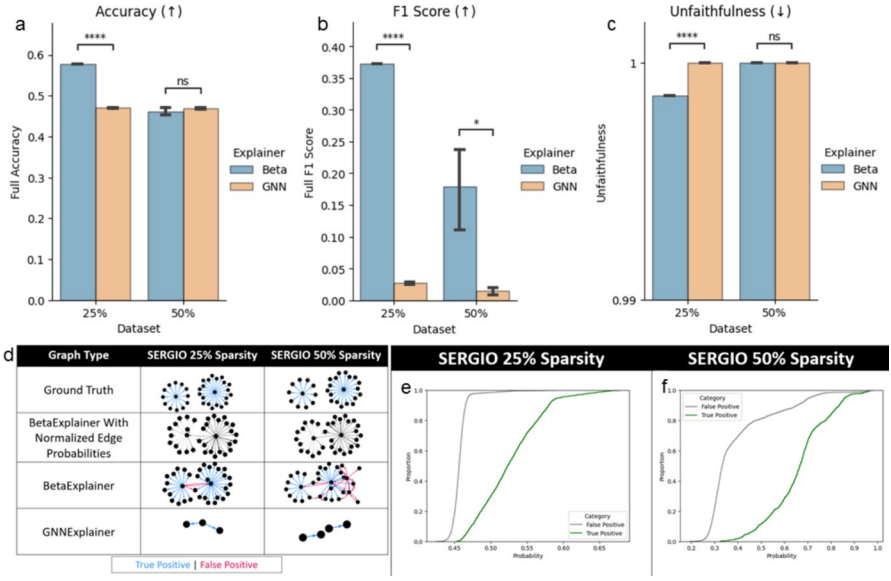

Fig. 5 We calculate the mean and standard errors of the accuracy **a**, F1 score **b**, and unfaithfulness **c** and the significant differences between explainer results (ns: $0.05 < p \leq 1$, *: $0.01 < p \leq 0.05$, **: $0.001 < p \leq 0.01$, ***: $0.0001 < p \leq 0.001$, and ****: $p \leq 0.0001$). We graph best results per explainer for datasets (if true positives *are* returned) **d**, denoting true positive (blue), false positive (red), and false negative (pink) edges and weighting BetaExplainer edges by probabilities and the eCDF of the BetaExplainer probabilities for true and false positives with respect to the groundtruth (e, f)

functions (eCDFs) of these probabilities for the true and false positive edges when compared to the ground truth. On average, we see that the probabilities assigned by BetaExplainer for true edges are higher than the ones for false edges. These probabilistic edge scores allow the user to select the most probable edges for the explanation based on a threshold of their choice. Uncertainty quantification is not possible using other existing GNN interpretation methods (including SubgraphX), highlighting the need for probabilistic explanation models like BetaExplainer.

Furthermore, higher average explanation certainty appears to improve performance metrics. Higher certainty associated with edges on average is associated with lower (or better) unfaithfulness over a hyperparameter sweep (Supplementary Fig. 11a-g). One potential concern is the unfaithfulness and sparsity trade-off: choosing too many certain edges may affect experimental prioritization. However, there are some edge masks with average certainty in the 0.5 to 0.6 range that achieve similar unfaithfulness to the most certain edge masks (Supplementary Fig. 11a-g), mitigating these concerns. The strong possibility of improvement indicates BetaExplainer will also perform well on other challenging datasets.

Considering that BetaExplainer improves upon GNNExplainer and SubgraphX in the unfaithfulness dimension, we will test it on an additional graph simulation with sparse node features. This is relevant because many real-world datasets are sparse, such as scRNA-seq datasets, due to technical limitations [26].

3.2 BetaExplainer Performs Well on Graph Datasets With Highly Sparse Node Features

BetaExplainer achieves similar accuracy as GNNExplainer and SubgraphX on the sparse SERGIO datasets and significantly better F1 Scores for both 25% and 50% sparse node feature datasets (Fig. 5a-b). These results again highlight the better precision-recall trade-off of BetaExplainer than the baseline methods. We expect overall low scores as the GNN input graph calculated using correlation is sparse, as is standard in the field [30]. We test this hypothesis by excluding the false negatives representing the true edges absent from the correlation graph. This experiment confirms our assumption: we see a massive improvement in accuracy and F1 Scores (Supplementary Fig. 9a-b). Furthermore, this calculation maintains the same pattern as the original – BetaExplainer outperforms GNNExplainer and SubgraphX for the F1 Score metrics (Supplementary Fig. 9b).

Following the nuance of this dataset described above, the unfaithfulness metric (Fig. 5c) for this task is less reliable than the F1 score metric. The previous simulation datasets [16] contained all ground truth edges through both the model training and explanation evaluation. However, using a graph with a different structure as input seems to affect GNN training. This skews the unfaithfulness metric results for the explained graph since important model edges may not be in the ground truth graph, suggesting a trade-off between accuracy and unfaithfulness metrics.

A qualitative analysis of the best-performing graphs for each explainer over each dataset confirms the primary driver of F1 Scores in Fig. 5d. While GNNExplainer may have higher precision, it comes at a significant cost to the recall: it returns only two edges. SubgraphX returned no edges, indicating comparable accuracy due to accurate negative instances. BetaExplainer may have lower precision, but its recall is much higher. For real-world testing, researchers will need fewer false negative gene-gene interaction edges, suggesting in this case, BetaExplainer has a better precision-recall trade-off balance.

Next, we examine the eCDFs of the true and false positive edges for BetaExplainer in Fig. 5e-f. Most true positives have a probability of 0.5 or greater, while most false negatives are less than or equal to the 0.5 bound. Users may prioritize a small set of the most essential edges in a real-world scenario from the high probabilistic scores obtained by BetaExplainer. One can be confident of this selection as this set contains few false positives. This property is compelling as BetaExplainer can approximate the edge mask probabilistic distribution, prioritizing the actual important edges.

Finally, BetaExplainer likely performs better than GNNExplainer due to the ability to capture the underlying distribution of edge importance by choosing the best α and β parameters and thus is the better option for sparse datasets. Even if there is no improvement for non-sparse datasets, indicating either explainer is efficacious, improving upon challenging datasets is necessary. BetaExplainer improves upon critical metrics to the baselines across all challenging datasets tested, making it a helpful explanation method for the community.

3.3 Example of BetaExplainer's Real-World Application

BetaExplainer performs well on the heterophilic Texas dataset [29][28], demonstrating its applicability to real-world datasets. BetaExplainer achieves an unfaithfulness of 0.5629 on 48.31% of the original edges. While SubgraphX and GNNExplainer deem a smaller fraction of edges important (0.308% and 33.85%, respectively), they produce worse unfaithfulness results (1.0 and 0.9988, respectively). BetaExplainer captures a better sparsity-performance trade-off, suggesting BetaExplainer's improved performance on the synthetic datasets holds on real-world data.

Next, we turn to qualitative analysis of BetaExplainer on the Texas dataset, to fully compare it to the baselines (Fig. 6a). BetaExplainer returns a masked graph that contains more of the large-scale original structures, which likely accounts for its improved performance (Fig. 6a). Furthermore, BetaExplainer appears to prioritize edges between nodes of a similar class or edges connecting these nodes within a two-hop neighborhood which may also increase model performance (Fig. 6a). GNNExplainer and SubgraphX do not appear to retain these overall structure to the same degree, confirming our quantitative analysis (Fig. 6a).

Finally, we graph the cumulative distribution function of BetaExplainer's edge mask (Fig. 6b). We are able to generate a set of probabilities for each edge, providing an edge ranking for analysis. This is particularly important for real-world datasets, wherein a large set of potential hypothesis would not be viable for experimental confirmation.

3.4 Computational Complexity & Scalability

Finally, we examine the runtime and memory usage of BetaExplainer and compare it to GNNExplainer and SubgraphX. Incorporating variational inference in our method is bound to increase the runtime per epoch as compared to other methods as our experimental analysis confirms (Table 1). Using batch input methods for the graph classification datasets appears to drastically improve this average runtime in the case of graph datasets, though not enough to outperform SubgraphX or GNNExplainer.

Similarly, the memory required for BetaExplainer is higher than the comparison methods (Table 2). The best-performing method is SubgraphX, which lacks the notion of uncertainty BetaExplainer provides. As a result, the complexity-benefit tradeoff may rule in favor of BetaExplainer. In addition, while this trade-off is

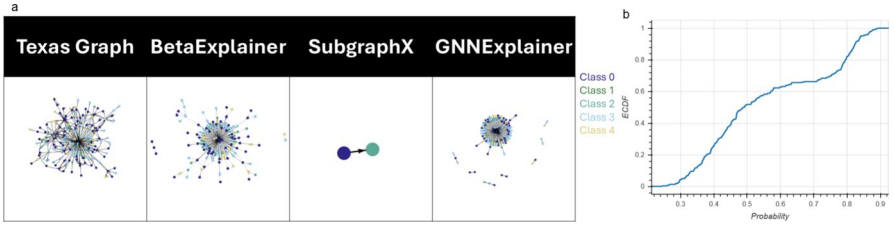

Fig. 6 We graph all edges with a probability of at least 0.5 and specify thickness based on the weighted probability **a**. The nodes connected to these edges are denoted in colors associated with the node classes. We also graph the eCDF denoting the spread of probabilities of the associated edge mask **b**

Table 1 The average runtime (in seconds) over 50 epochs was calculated runs, and the average average runtime per epoch was calculated for BetaExplainer and GNNExplainer. As SubgraphX does not have a notion of training epochs, the average runtime for SubgraphX is denoted as a *. Lower (better) times are bolded

Dataset	Explainer	Full Runtime (s)	Average Epoch Runtime (s)
SG-BASE	GNN	**2.09**	**0.0419**
SG-BASE	SubgraphX	20.1	*
SG-BASE	Beta	11.2	2.24e−1
SG-HETEROPHILIC	GNN	3.35	**0.0669**
SG-HETEROPHILIC	SubgraphX	**0.464**	*
SG-HETEROPHILIC	Beta	15.4	3.08e−1
SG-LESSINFORM	GNN	2.96	**0.0593**
SG-LESSINFORM	SubgraphX	**2.21**	*
SG-LESSINFORM	Beta	11.4	2.29e−1
SG-MOREINFORM	GNN	2.07	**0.0414**
SG-MOREINFORM	SubgraphX	**0.605**	*
SG-MOREINFORM	Beta	14.1	2.82e−1
SG-UNFAIR	GNN	2.86	**0.0572**
SG-UNFAIR	SubgraphX	**2.71**	*
SG-UNFAIR	Beta	21.8	4.35e−1
SERGIO 25% Sparsity	GNN	0.127	**0.00510**
SERGIO 25% Sparsity	SubgraphX	**0.0932**	*
SERGIO 25% Sparsity	Beta	515	10.3
SERGIO 25% Sparsity With Batching	Beta	57.2	1.14
SERGIO 50% Sparsity	GNN	**3.19e −2**	**1.27e −3**
SERGIO 50% Sparsity	SubgraphX	0.208	*
SERGIO 50% Sparsity	Beta	415	8.3
SERGIO 50% Sparsity With Batching	Beta	57.2	1.14

expected, BetaExplainer's memory usage is still within feasible constraints. As a result, we feel the added complexity of variational inference is reasonable from an application perspective.

4 Discussion

BetaExplainer learns a probabilistic importance score for each edge by learning a Beta distribution. This is achieved by minimizing the KL divergence between the model output on the masked graph and the original output. By learning an importance score, users have a notion of uncertainty in edge importance. Furthermore, learning a probability distribution allows users to incorporate priors, which

Table 2 The average current and peak memories were calculated over 25 runs for each dataset and explainer combination. Lower (better) memories are bolded

Dataset	Explainer	Memory Post Run (MB)	Peak Memory Usage (MB)
SG-BASE	GNN	3.5e−3	0.11
SG-BASE	SubgraphX	**0.00026**	**0.0024**
SG-BASE	Beta	0.025	0.055
SG-HETEROPHILIC	GNN	0.0032	0.11
SG-HETEROPHILIC	SubgraphX	**0.00026**	**2.4e −3**
SG-HETEROPHILIC	Beta	0.026	0.056
SG-LESSINFORM	GNN	0.0036	0.11
SG-LESSINFORM	SubgraphX	**0.00026**	**2.4e −3**
SG-LESSINFORM	Beta	0.029	0.059
SG-MOREINFORM	GNN	0.00036	0.11
SG-MOREINFORM	SubgraphX	**0.00026**	**2.4e −3**
SG-MOREINFORM	Beta	0.028	0.057
SG-UNFAIR	GNN	0.0033	0.11
SG-UNFAIR	SubgraphX	**0.00026**	**2.4e −3**
SG-UNFAIR	Beta	0.027	0.057
SERGIO 25% Sparsity	GNN	0.012	0.017
SERGIO 25% Sparsity	SubgraphX	**0.00027**	**0.0024**
SERGIO 25% Sparsity	Beta	0.062	2.05
SERGIO 50% Sparsity	GNN	0.012	0.018
SERGIO 50% Sparsity	SubgraphX	**0.000274**	**0.0024**
SERGIO 50% Sparsity	Beta	0.063	2.05

provides the method with more information to adapt to datasets with challenging properties.

BetaExplainer achieves similar performance across accuracy and F1 scores to current state-of-the-art method GNNExplainer [21] and SubgraphX [23] for associated datasets and particularly achieves significantly better F1 Scores for the sparse node feature datasets. It also has similar if not better unfaithfulness results for almost all datasets, which are often significantly better, for the first five datasets [16]. Finally, BetaExplainer provides a measure of uncertainty, allowing users to focus on the most certain edges.

BetaExplainer has a few potential areas of improvement. It is sensitive to the number of GNN convolution layers due to the GNN oversmoothing issue. Addressing runtime is also a potential area to improve on, as BetaExplainer can take approximately 8.5 to 58 more seconds than GNNExplainer to run and from about 9.19 to 57.1 s longer than SubgraphX, and potentially more if batching is not used for graph datasets (Table 1). Similar improvements can be made for reducing the memory usage as a future direction (Table 2). Finally, the method does not directly provide a node explanation. Thus, further extension should add a node explainer element while decreasing complexity. As BetaExplainer results

are at minimum comparable to other baselines, it appears both the complexity and performance tradeoff and the node explanation extension are viable.

The properties of the BetaExplainer model probably explain the improvements seen across metrics. We use various α and β parameters based on the datasets, which likely perform best as they well-capture the underlying dataset properties. Exploration of these best-performing parameters for the dataset ensures a strong prior on important edges. Furthermore, BetaExplainer likely improves upon unfaithfulness results by optimizing KL divergence between the original model output on the full graph and model output on the masked graph.

In addition, results suggest users may apply BetaExplainer to a wide variety of datasets including an example from the real-word setting. However, we plan to perform more exploration of real-world applications for the method. Much like the SERGIO [27] datasets, which represent gene expression data, many real-world expression datasets have sparse node features due to experimental limitations [26]. Since computational methods may clarify gene-gene interactions without expensive laboratory resources, BetaExplainer's ability to adapt to sparse datasets is critical [25]. Graph classification models are another area of exploration as this may prove to be a factor in the SERGIO results as they simulate a graph classification problem [27]. Follow-up will determine whether sparsity constitutes the major difference in performance across models. In addition, examining whether the improved performance holds for real-world heterophilic graph problems such as protein structure model prediction [31]. Exploration of these real-world datasets and other important applications will be critical to understanding BetaExplainer's potential applications and determining important elements of real-world graphs.

We anticipate anomaly detection as a particular application focus. Khan et al. (2024) [32] propose both KL divergence regularization and probabilistic models [33] for anomaly detection, suggesting that BetaExplainer may also perform well in this domain. BetaExplainer's ability to provide prior information through well-chosen parameters, in particular, may be beneficial in discovering anomalies.

Adding other model loss or architecture components to BetaExplainer may also prove advantageous. Incorporating Tversky Loss, for instance, into our original loss term will ensure that the model remains robust on a dataset with few important features [34]. BetaExplainer may also benefit Transformer methods by indirect masking of attention weights through edge masking [15][14]. Masking out these edges improves self-supervised learning performance by providing guidance towards the most important graph structures given the prior information provided by BetaExplainer [15][14]. This will provide these attention mechanisms [15][14] with domain-relevant distributional prior information. Thus, these extensions will be of primary interest for future work.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s44199-025-00118-x.

Acknowledgements We are grateful to Ghulam Murtaza for helping us understand challenges with the SERGIO datasets and for aid in revising, Michal Golanvesky for providing seed resources, and Alexandra

Miller for aid in proofreading. This research was conducted using computational resources and services provided by the Center for Computation and Visualization at Brown University.

Author Contributions W. S. wrote the main manuscript text, prepared all figures, and ran the code to get the main results. S. P. developed the model. M. W. re-ran all models to ensure reproducibility. R. S. and L. C. supervised the project and reviewed the manuscript.

Funding This research was also supported in part by a David & Lucile Packard Fellowship for Science and Engineering awarded to LC. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any of the funders.

Data Availability Datasets are accessible at the associated GitHub or through simulation code as described at the GitHub: https://github.com/wsloneker/BetaExplainerDemo.

Code Availability All code is available under the open-source MIT license at https://github.com/wsloneker/BetaExplainerDemo.

Declarations

Conflict of interest LC is an employee of Microsoft and owns equity in the company. All other authors have declared that they have no Conflict of interest.

References

1. Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Su, M.: Graph neural networks: A review of methods and applications. AI Open, pages 57–81 (2020)
2. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.: A comprehensive survey on graph neural networks. IEEE Trans. Neural Netw. Learn. Syst. **32**(1), 4–24 (2021)
3. Liu, Z., Wan, G., Prakash, B., Lau, M., Jin, W.: A review of graph neural networks in epidemic modeling. ACM Digital Library, Pages 6577 – 6587 (2024)
4. Ju, W., Yi, S., Wang, Y., Xiao, Z., Mao, Z., Li, H., Gu, Y., Qin, Y., Yin, N., Wang, S., Liu, X., Luo, X., Yu, P., Zhang, M.: A survey of graph neural networks in real world: Imbalance, noise, privacy and ood challenges. arxiv (2024)
5. Zhang, H., Wu, B., Yuan, X., Pan, S., Tong, H., Pei, J.: Trustworthy graph neural networks: Aspects, methods, and trends. Proc. IEEE **112**(2), 97–139 (2024)
6. Pržulj, N., Corneil, D., Jurisica, I.: Modeling interactome: scale-free or geometric? Bioinformatics **20**(18), 3508–3515 (2004)
7. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 701–710 (2014)
8. Xenos, A.: Simplifying complex machine learning by linearly separable network embedding spaces. arxiv (2024)
9. Zhou, Y., Zheng, H., Hao, S., Li, D., Zhao, J.: Graph neural networks: Taxonomy, advances, and trends. ACM Trans. Int. Syst. Technol. **13**(15), 1–54 (2022)

10. Waikhom, L., Patgiri, R.: A survey of graph neural networks in various learning paradigms: methods, applications, and challenges. Artif. Intell. Rev. **56**, 6295–6364 (2023)
11. Zhang, H., Wu, B., Yuan, X., Pan, S., Tong, H., Pei, J.: Trustworthy graph neural networks: Aspects, methods, and trends. IEEE. Proc (2024)
12. Khan, W., Haroon, M.: A pilot study and survey on methods for anomaly detection in online social networks. Human-Centric Smart Comput. **316**, (2022)
13. Yuan, H., Yu, H., Gui, S., Ji, S.: Explainability in graph neural networks: A taxonomic survey. IEEE (2022)
14. Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., Liu, T.-Y.: Do transformers really perform bad for graph representation? Conference on Neural Information Processing Systems (2021)
15. Wu, Z., Jain, P., Wright, M.A., Mirhosein, A., Gonzalez, J.E., Stoica, I.: Representing long-range context for graph neural networks with global attention. Conference on Neural Information Processing Systems (2021)
16. Agarwal, C., Queen, O., Lakkaraju, H., Zitnik, M.: Evaluating explainability for graph neural networks. Sci Data **10**(144), (2023)
17. Simonyan, K., Vedaldi, A., Zisserman, A.: Deep inside convolutional networks: Visualising image classification models and saliency maps. ICLR (2024)
18. Pope, P., Kolouri, S., Rostami, M., Martin, C., Hoffmann, H.: Explainability methods for graph convolutional neural networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 10772–10781 (2019)
19. Baldassarre, F., Azizpour, H.: Explainability techniques for graph convolutional networks. International Conference on Machine Learning (ICML) Workshops (2019)
20. Sundararajan, M., Taly, A., Yan, Q.: Axiomatic attribution for deep networks. ICML, page 3319-3328 (2017)
21. Ying, R., Bourgeois, D., You, J., Zitnik, M., Leskovec, J.: Gnnexplainer: Generating explanations for graph neural networks. Adv. Neural Inf. Process. Syst. (2019)
22. Luo, D., Cheng, W., Xu, D., Yu, W., Zong, B., Chen, H., Zhang, X.: Parameterized explainer for graph neural network. Advances in Neural Inf. Process. Syst. Pages 19620–19631 (2020)
23. Yuan, H., Yu, H., Wang, J., Li, K., Ji, S.: On explainability of graph neural networks via subgraph explorations. Proceedings of The 38th International Conference on Machine Learning, pages 1241–12252 (2021)
24. Vu, M., Thai, M.: Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. Advances in Neural Information Processing Systems, pages 12225–12235 (2020)
25. Bigness, J., Loinaz, X., Patel, S., Larschan, E., Singh, R.: Integrating long-range regulatory interactions to predict gene expression using graph convolutional networks. J. Comput. Biol. **29**(5), 409–422 (2022)
26. Bouland, G.A., Mahfouz, A., Reinders, M.J.T.: Consequences and opportunities arising due to sparser single-cell rna-seq datasets. Genome Biol. **24**(86), (2023)
27. Dibaeinia, P., Sinha, S.: Sergio: A single-cell expression simulator guided by gene regulatory networks. Sci. Data **11**(3), (2020)
28. Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., Yang, B.: Geom-gcn: Geometric graph convolutional networks. In International Conference on Learning Representations (2020)
29. Ye, Y., Ji, S.: Sparse graph attention networks. IEEE Trans. Knowl. Data Eng. **35**(1), 905–916 (2023)
30. Kim, D., Tran, A., Kim, H., Lin, Y., Yang, J., Yang, P.: Gene regulatory network reconstruction: harnessing the power of single-cell multi-omic data. NPJ Syst. Biolo. Appl. **9**(51), (2023)
31. Zheng, X., Wang, Y., Liu, Y., Li, M., Zhang, M., Jin, D., Yu, P., Pan, S.: Graph neural networks for graphs with heterophily: A survey. arXiv:2202.07082**14**(8), (2024)
32. Khan, W., Ishrat, M., Neyaz Khan, A., Arif, M., Ahamed Shaikh, A., Khubrani, M.M., Alam, S., Shuaib, M., John, R.: Detecting anomalies in attributed networks through sparse canonical correlation analysis combined with random masking and padding. IEEE Access **12**, 65555–65569 (2024)
33. Khan, W., Haroon, M.: An efficient framework for anomaly detection in attributed social networks. Int J Inf Technol. Page 3069–3076 (2022)
34. Khan, W., Ebrahim, N.: Anogat-sparse-tl: A hybrid framework combining sparsification and graph attention for anomaly detection in attributed networks using the optimized loss function incorporating the twersky loss for improved robustness. Knowl-Based Syst **311**, 113144 (2025)