Towards Inductive and Efficient Explanations for Graph Neural Networks

Dongsheng Luo, *Member, IEEE*, Tianxiang Zhao, Wei Cheng, Dongkuan Xu, Feng Han, Wenchao Yu, Xiao Liu, Haifeng Chen, and Xiang Zhang

***Abstract*—Despite recent progress in Graph Neural Networks (GNNs), explaining predictions made by GNNs remains a challenging and nascent problem. The leading method mainly considers the local explanations, i.e., important subgraph structure and node features, to interpret why a GNN model makes the prediction for a single instance, e.g. a node or a graph. As a result, the explanation generated is painstakingly customized at the instance level. The unique explanation interpreting each instance independently is not sufficient to provide a global understanding of the learned GNN model, leading to the lack of generalizability and hindering it from being used in the inductive setting. Besides, training the explanation model explaining for each instance is time-consuming for large-scale real-life datasets. In this study, we address these key challenges and propose PGExplainer, a parameterized explainer for GNNs. PGExplainer adopts a deep neural network to parameterize the generation process of explanations, which renders PGExplainer a natural approach to multi-instance explanations. Compared to the existing work, PGExplainer has better generalization ability and can be utilized in an inductive setting without training the model for new instances. Thus, PGExplainer is much more efficient than the leading method with significant speed-up. In addition, the explanation networks can also be utilized as a regularizer to improve the generalization power of existing GNNs when jointly trained with downstream tasks. Experiments on both synthetic and real-life datasets show highly competitive performance with up to 24.7% relative improvement in AUC on explaining graph classification over the leading baseline.**

***Index Terms*—Deep learning, graph neural networks, interpretability.**

I. Introduction

GRAPH Neural Networks (GNNs) are powerful tools for representation learning of graph-structured data, such as social networks [54], document citation graphs [44], and microbiological graphs [52]. GNNs routinely adopt a message passing scheme to learn node representations by aggregating representation vectors of their neighbors [17], [55]. This scheme enables GNN to capture both node features and graph topology. GNN-based methods have achieved state-of-the-art performance in node classification, graph classification, and link prediction [23], [53], [63].

Despite their remarkable effectiveness, the rationales of predictions made by GNNs are not easy for humans to understand. Since GNNs aggregate both node features and graph topology to make predictions, to understand predictions made by GNNs, important subgraphs and/or a set of features, which are also known as explanations, need to be uncovered. In the literature, although a variety of efforts have been undertaken to interpret general deep neural networks, existing approaches [6], [14], [21], [22], [33], [37], [50] in this line fall short in their ability to explain graph structures, which is essential for GNNs. Explaining predictions made by GNNs remains a challenging problem, on which few methods have been proposed [48], [58], [60], [61], [62]. The combinatorial nature of explaining graph structures makes it difficult to design models that are both robust and efficient.

Recently, the first general model-agnostic approach for GNNs, GNNExplainer [58], was proposed to address the problem. It takes a trained GNN and its predictions as inputs to provide interpretable explanations for a given instance, e.g. a node or a graph. The explanation includes a compact subgraph structure and a small subset of node features that are crucial in GNN's prediction for the target instance. Nevertheless, there are several limitations to the existing approach. First, GNNExplainer largely focuses on providing the local interpretability by generating a painstakingly customized explanation for a single instance individually and independently. The explanation provided by GNNExplainer is limited to a single instance, making GNNExplainer difficult to be applied in the inductive setting because the explanations are hard to generalize to other unexplained nodes. As pointed out in previous studies, models interpreting each instance independently are not sufficient to provide a global understanding of the trained model [21]. Furthermore, GNNExplainer has to be retrained for every single explanation. As a result, in real-life scenarios where plenty of nodes need to be interpreted, GNNExplainer would be time-consuming and impractical. Moreover, as GNNExplainer was developed for interpreting individual instances, the authors further provided additional ad-hoc post-analysis, such as identifying the

Manuscript received 19 May 2021; revised 28 August 2023; accepted 31 January 2024. Date of publication 6 February 2024; date of current version 2 July 2024. This work was supported in part by NSF under Grant IIS-1707548, Grant CBET-1638320 and Grant IIS-2331908. Recommended for acceptance by K. M. Lee. *(Corresponding author: Dongsheng Luo.)*

Dongsheng Luo is with Florida International University, Miami, FL 33199 USA (e-mail: dluo@fiu.edu).

Tianxiang Zhao, Xiao Liu, and Xiang Zhang are with Pennsylvania State University, State College, PA 16802 USA (e-mail: tkz5084@psu.edu; xxl213@psu.edu; xzz89@psu.edu).

Wei Cheng, Wenchao Yu, and Haifeng Chen are with NEC Lab America, Inc., San Jose, CA 95110 USA (e-mail: weicheng@nec-labs.com; wyu@nec-labs.com; haifeng@nec-labs.com).

Dongkuan Xu is with North Carolina State University, Raleigh, NC 27695 USA (e-mail: dxu27@ncsu.edu).

Feng Han is with the University of California, Berkeley, CA 94720 USA (e-mail: feng.han@berkeley.edu).

Digital Object Identifier 10.1109/TPAMI.2024.3362584

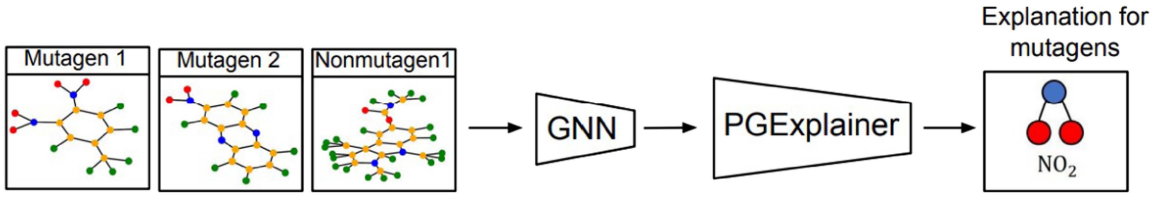


Fig. 1. PGExplainer provides human-understandable explanations for predictions made by GNNs. The left part shows the process of applying GNNs for graph classification on the MUTAG dataset. A GNN based model is trained to predict their mutagenic effects. As a post-hoc method, PGExplainer takes the trained GNN model as input and provides consistent explanations for predictions made by the GNN model. For the mutagen molecule graphs in the example, the explanation is the NO_2 group.

representative instance and graph alignment, to extract the shared motifs to explain a set of instances (e.g., graphs of a given class in graph classification task) [58]. Since the explanatory motifs are not learned end-to-end, the model however may suffer from suboptimal generalization performance. How to explain predictions of GNNs on a set of instances collectively and easily generalize the learned explainer model to other instances in the inductive setting remains largely unexplored in the literature.

To provide a global understanding of predictions made by GNNs, in this study, we emphasize the *collective* and *inductive* nature of this problem that an explanation model should be able to explain a set of instances and infer new instances without re-training. We present an inductive and effective explanation model to interpret graph neural networks, denoted by PGExplainer (Fig. 1). PGExplainer is a general explainer that applies to any GNN based models for various tasks, including node classification, graph classification, and link prediction. Specifically, a generative probabilistic model for graph data is utilized in PGExplainer. Generative models have shown the power to learn succinct underlying structures from the observed graph data [27]. PGExplainer uncovers these underlying structures as the explanations, which is believed to make the most contribution to GNNs' predictions [41]. We model the underlying structure as edge distributions, where the explanatory graph is sampled from. To collectively explain predictions of multiple instances, the generation process in PGExplainer is parameterized with a deep neural network. Since the neural network parameters are shared across the population of explained instances, PGExplainer is naturally applicable to provide global explanations of GNNs. Furthermore, PGExplainer has better generalization power because a trained PGExplainer model can be utilized in an inductive setting to infer explanations of unexplained nodes without retraining the explanation model. This also makes PGExplainer much faster than the existing work.

Another application of PGExplainer is to learn graph structures for GNN models, which are error-prone to noisy topological structures. GNNs learn node embeddings by recursively aggregating messages from their neighborhoods. Such message passing mechanism is associated with cascading effects that small noise may propagate to neighboring nodes and leads to sub-optimal representations of many others. To improve the robustness of GNN models, graph structure learning methods first learn a denoised graph structure for the downstream task [67]. However, the original input graph is insufficiently analyzed in existing state-of-the-art methods since they will be dropped once the subgraphs are extracted [39], [66]. Based upon PGExplainer, we propose a more flexible method to learn compact subgraph structures that maximizes the mutual information between the subgraph and labels in the downstream task. When the downstream task objective is jointed optimized with a size constraint regularizer, our method can improve the accuracy performance and robustness of GNNs.

Experimental results on both synthetic and real-life datasets demonstrate that PGExplainer can achieve consistent and accurate explanations, bringing up to 24.7% improvement in AUC over the SOTA method on explaining graph classification with significant speed-up.

II. Related Work

Graph neural networks: Graph Neural networks (GNNs) have achieved remarkable success in various tasks, including node classification [23], [31], [53], graph classification [12], and link prediction [63]. The study of GNNs was initiated in [19], and then extended in [47]. These methods iteratively aggregate neighbor information to learn node representations until reaching a static state. Inspired by the success of convolutional neural networks (CNNs) in computer vision, attempts of applying convolutional operations to graphs were derived based on graph spectral theory [4] and graph Fourier transformation [49]. In recent work, GNNs broadly encode node features as messages and adopt the message passing mechanism to propagate and aggregate them along edges to learn node/graph representations, which are then utilized for downstream tasks [12], [31], [36], [40], [47], [53]. For efficiency consideration, localized filters were proposed to reduce computation cost [23]. The self-attention mechanism was introduced to GNNs in GAT to differentiate the importance of neighbors [53]. Xu. et al. analyzed the relationship between GNNs and Weisfeiler-Lehman graph isomorphism test, and showed the express power of GNNs [55].

Explaining GNNs: Interpretability and feature selection have been extensively addressed in neural networks. Methods demystifying complicated deep learning models can be grouped into two main families, whitebox and blackbox [21], [22]. Whitebox mechanisms mainly focus on yielding explanations for individual predictions. Forward and backward propagation based methods are used routinely in whitebox mechanisms. Forward propagation based methods broadly perturb the input and/or hidden representations and check the corresponding updating results in the downstream task [10], [15], [35]. The underlying

intuition is that the outputs of the downstream task are likely to significantly change if important features are occluded. Backward propagation based methods, in general, infer important features from the gradients of the deep neural networks. They compute weights of features by propagating the gradients from the output back to the input. Blackbox methods generate explanations by locally learning interpretable models, such as linear models, and additive models to approximate the predictions [5], [38], [64], [65].

Following the line of forward propagation methods, GNNExplainer initiates the research on explaining predictions on graph-structured data [58]. It excludes certain edges and node features to observe the changes in node/graph classification. Explanations (subgraphs/important features) are extracted by maximizing the mutual information between the distribution of possible subgraphs and the GNN's prediction. However, similar to other forward propagation methods, GNNExplainer generates customized explanations for single instance prediction independently, making it insufficient to provide a global understanding of the trained GNN model. Besides, it is naturally difficult to be applied in the inductive setting and provide explanations for multiple instances.

Graph generation: PGExplainer learns a probabilistic graph generative model to provide explanations for GNNs. The first model generating random graph is the Erdős-Rényi model [13], [16]. In the random graph proposed by Gilbert, each potential edge is independently chosen from a Bernoulli distribution. Some works generate graphs with certain properties reflected, such as pairwise distances betweenness [7], node degree distribution [34], and spectral properties [2], [26]. In recent years, deep learning models have shown great potential to generate graphs with complex properties preserved [20], [41], [59]. However, these methods mainly aim to generate graphs that reflect certain properties in the training graphs.

III. Background

We first describe notations, and then provide some background on graph neural networks.

Notations: Let $G = (\mathcal{V}, \mathcal{E})$ represent the graph with $\mathcal{V} = \{v_1, v_2 \ldots v_N\}$ denoting the node set and $\mathcal{E} \in \mathcal{V} \times \mathcal{V}$ as the edge set. The numbers of nodes and edges are denoted by N and M, respectively. A graph can be described by an adjacency matrix $\mathbf{A} \in \{0,1\}^{N \times N}$, with $a_{ij} = 1$ if there is an edge connecting node i and j, and $a_{ij} = 0$ otherwise. Nodes in $\mathcal{V}$ are associated with the d-dimensional features, denoted by $\mathbf{X} \in \mathbb{R}^{N \times d}$.

Graph neural networks: GNNs adopt the message-passing mechanism to propagate and aggregate information along edges in the input graph to learn node representations [17], [23], [31], [53]. Each GNN layer includes three essential steps. First, at the propagation step of the i-th GNN layer, for each edge (i, j), GNN computes a message $\mathbf{m}_{ij}^l = \text{Message}(\mathbf{h}_i^{l-1}, \mathbf{h}_j^{l-1})$, where $\mathbf{h}_i^{l-1}$ and $\mathbf{h}_j^{l-1}$ are representations of v_i and v_j in previous layer, respectively. Second, at the aggregation step, for each node v_i, GNN aggregates messages received from its neighbor nodes, denoted by $\mathcal{N}_i$, with an aggregation function $\mathbf{m}_{i\cdot}^l = \text{aggregation}(\{\mathbf{m}_{ij}^l | j \in \mathcal{N}_i\})$. Last, at the updating step, GNN updates the vector representation for each node v_i via $\mathbf{h}_i^l = \text{update}(\mathbf{m}_{i\cdot}^l, \mathbf{h}_i^{l-1})$, a function taking the aggregated message and the representation of itself as inputs. Hidden representation of the last GNN layer serve as the final node representation: $\mathbf{z}_i = \mathbf{h}_i^L$, which is then used for downstream tasks, such as node/graph classification, and link prediction.

IV. The PGExplainer

In this section, we introduce PGExplainer. Different from GNNExplainer which provides explanations on both structure and features, PGExplainer focuses on explanation on graph structures because feature explanation in GNNs is similar to that in non-graph neural networks, which has been extensively studied in the literature [1], [10], [15], [21], [35], [38], [45]. PGExplainer is flexible and applicable to interpreting all kinds of GNNs. We start with the learning objective of PGExplainer (Section IV-A) and then present the reparameterization strategy for efficient optimization (Section IV-B). In Section IV-C, we specify particular instantiations to understand GNNs on node and graph classifications.

A. The Learning Objective

The literature has shown that real-life graphs are with underlying structures [41], [43]. To explain predictions made by a GNN model, we divide the original input graph G_o into two subgraphs: $G_o = G_s + \Delta G$, where G_s presents the underlying subgraph that makes important contributions to GNN's predictions, which is the expected *explanatory graph*, and ΔG consists of the remaining task-irrelevant edges for predictions made by the GNN. Following [58], PGExplainer finds G_s by maximizing the mutual information between the GNN's predictions and the underlying structure G_s:

$$\max_{G_s} \text{MI}(Y_o, G_s) = H(Y_o) - H(Y_o | G = G_s), \tag{1}$$

where Y_o is the prediction of the GNN model with G_o as the input. The mutual information quantifies the probability of prediction Y_o when the input graph to the GNN model is limited to the explanatory graph G_s. The intuition behind this comes from the traditional forward propagation-based methods for the white box explanation [10]. For example, if removing an edge (i, j) dramatically changes the prediction in the GNN, then this edge is important and should be included in G_s. Otherwise, it can be considered as irrelevant edge for the GNN model to make the prediction. Since $H(Y_o)$ is only related to the GNN model whose parameters are fixed in the explanation stage, the objective is equivalent to minimizing the conditional entropy $H(Y_o | G = G_s)$.

However, the direct optimization of the above objective function is intractable as there are 2^M candidates for G_s. Thus, we consider a relaxation by assuming that the explanatory graph is a Gilbert random graph [16], where selections of edges from the original input graph G_o are conditionally independent to each other. Let $e_{ij} \in \mathcal{V} \times \mathcal{V}$ be the binary variable indicating whether the edge is selected, with $e_{ij} = 1$ if the edge (i, j) is selected, and 0 otherwise. Let G be the random graph variable. Based

on the above assumption, the probability of a graph G can be factorized as:

$$P(G) = \Pi_{(i,j)\in\mathcal{E}} P(e_{ij}). \tag{2}$$

A straightforward instantiation of $P(e_{ij})$ is the Bernoulli distribution $e_{ij} \sim Bern(\theta_{ij})$. $P(e_{ij} = 1) = \theta_{ij}$ is the probability that edge (i, j) exists in G. With this relaxation, we thus can rewrite the objective as:

$$\begin{aligned}\min_{G_s} H(Y_o|G = G_s) &= \min_{G_s} \mathbb{E}_{G_s}[H(Y_o|G = G_s)] \\ &\approx \min_{\Theta} \mathbb{E}_{G_s\sim q(\Theta)}[H(Y_o|G = G_s)] \end{aligned} \tag{3}$$

where $q(\Theta)$ is the distribution of the explanatory graph parameterized by θ's.

B. The Reparameterization Trick

Due to the discrete nature of G_s, we relax edge weights from binary variables to continuous variables in the range (0,1) and adopt the reparameterization trick to efficiently optimize the objective function with gradient-based methods [29]. We approximate the sampling process $G_s \sim q(\Theta)$ with a determinant function of parameters Ω, temperature τ, and an independent random variable ϵ: $G_s \approx \hat{G}_s = f_\Omega(G_o, \tau, \epsilon)$. The temperature τ is used to control the approximation. Here we utilize the binary concrete distribution as the instantiation [42]. Specifically, the weight $\hat{e}_{ij} \in (0, 1)$ of edge (i, j) in $\hat{G}_s$ is calculated by:

$$\epsilon \sim \text{Uniform}(0, 1), \quad \hat{e}_{ij} = \sigma((\log\epsilon - \log(1-\epsilon) + \omega_{ij})/\tau), \tag{4}$$

where $\sigma(\cdot)$ is the Sigmoid function, and $\omega_{ij} \in \mathbb{R}$ is the parameter. When $\tau \to 0$, the weight $\hat{e}_{ij}$ is binarized with $\lim_{\tau\to 0} P(\hat{e}_{ij} = 1) = \frac{\exp(\omega_{ij})}{1+\exp(\omega_{ij})}$. Since $P(e_{ij} = 1) = \theta_{ij}$, by choosing $\omega_{ij} = \log\frac{\theta_{ij}}{1-\theta_{ij}}$, we have $\lim_{\tau\to 0}\hat{G}_s = G_s$. This demonstrates the rationality of using binary concrete distribution to approximate the Bernoulli distribution. Moreover, with temperature $\tau > 0$, the objective function is smoothed with a well-defined gradient $\frac{\partial\hat{e}_{ij}}{\partial\omega_{ij}}$. Thus, with reparameterization, the objective in (3) becomes:

$$\min_{\Omega} \mathbb{E}_{\epsilon\sim\text{Uniform}(0,1)} H(Y_o|G = \hat{G}_s) \tag{5}$$

Considering efficient optimization, we follow [58] to modify the conditional entropy with cross-entropy $H(Y_o, \hat{Y}_s)$, where $\hat{Y}_s$ is the prediction of the GNN model with $\hat{G}_s$ as the input. With the above relaxations, we adopt Monte Carlo to approximately optimize the objective function:

$$\begin{aligned}&\min_{\Omega} \mathbb{E}_{\epsilon\sim\text{Uniform}(0,1)} H(Y_o, \hat{Y}_s) \\ &\approx \min_{\Omega} -\frac{1}{K}\sum_{k=1}^{K}\sum_{c=1}^{C} P(Y_o = c)\log P(\hat{Y}_s = c) \\ &= \min_{\Omega} -\frac{1}{K}\sum_{k=1}^{K}\sum_{c=1}^{C} P_\Phi(Y = c|G = G_o)\log P_\Phi(Y = c|G \\ &= \hat{G}_s^{(k)}),\end{aligned} \tag{6}$$

where Φ denotes the parameters in the trained GNN, K is the total number of sampled graph, C is the number of labels, $\hat{G}_s^{(k)}$ is the k-th sampled graph with (4), parameterized by Ω.

C. Explanation of Graph Neural Networks With a Global View

Although explanations provided by the leading method GNNExplainer [58] preserve the local fidelity, they do not help to understand the general picture of the model across a population [56]. Furthermore, various GNN based models have been applied to analyze graph data with millions of instances [57], the cost of applying local explanations one-by-one can be prohibitive with such large datasets in practice. On the other hand, explanations with a global view of the model ascertain users' trust [45]. Furthermore, these models can generalize explanations to new instances without retraining, making it more efficient to explain large scale datasets.

To have a global view of a GNN model, our method collectively explains predictions made by a trained model on multiple instances. Instead of treating Ω in (6) as independent variables, we utilize a parameterized network to learn to generate explanations from the trained GNN model. After training, the parameterized network can be used in the inductive setting to provide explanations for unexplained instances. In general, GNN based models first learn node representations and then feed the vector representations to downstream tasks [23], [31], [53]. We denote these two functions by $\text{GNNE}_{\Phi_0}(\cdot)$ and $\text{GNNC}_{\Phi_1}(\cdot)$, respectively. For GNNs without explicit classification layers, we use the last layer instead. As a result, we can represent a GNN model with:

$$\mathbf{Z} = \text{GNNE}_{\Phi_0}(G_o, \mathbf{X}), \;\; Y = \text{GNNC}_{\Phi_1}(\mathbf{Z}). \tag{7}$$

$\mathbf{Z}$ is the matrix of node representations encoding both features and structure of the input graph, which is used as an input in the explanation network to calculate the parameter Ω:

$$\Omega = g_\Psi(G_o, \mathbf{Z}). \tag{8}$$

Ψ denotes parameters in the explanation network, which is shared by all edges among the population. Therefore, PGExplainer can be utilized to collectively provide explanations for multiple instances. Specifically, in the collective setting with instance set $\mathcal{I}$, the objective of PGExplainer is:

$$\begin{aligned}\min_{\Psi} &-\sum_{i\in\mathcal{I}}\sum_{k=1}^{K}\sum_{c=1}^{C} P_\Phi(Y = c|G = G_o^{(i)})\log P_\Phi(Y = c|G \\ &= \hat{G}_s^{(i,k)}),\end{aligned} \tag{9}$$

where $G^{(i)}$ is the input graph and $\hat{G}_s^{(i,k)}$ is the k-th sampled graph with (4, 8) for i-th instance. We consider both graph and node classifications and specify an instantiation for each task. Solutions for other tasks, such as link prediction, are similar and thus omitted.

Explanation network for node classification: Considering that explanations for nodes in a graph may appear diverse structures [58], especially for nodes with different labels. For example, an edge (i, j) is important for the prediction of node u, but not for another node v. Based on this motivation, we

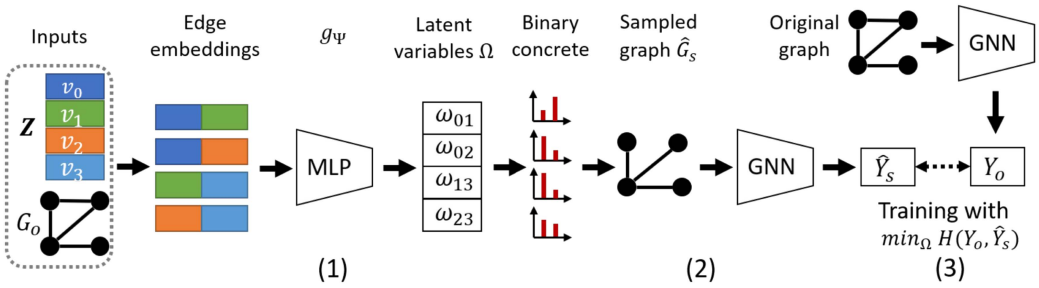


Fig. 2. Illustration of PGExplainer for explaining GNNs on graph classification. (1) The left part demonstrates the explanation network. It takes node representations $\mathbf{Z}$ as well as the original graph G_o as inputs to compute Ω, the latent variables in edge distributions. Edge distributions are severed as the explanation. In case that an explanatory subgraph is wanted, we select top-ranked edges according to latent variables Ω. (2) A random graph $\hat{G}_s$ is sampled from edge distributions and then feed to the trained GNN model to get the prediction $\hat{Y}_s$. (3) Parameter Ψ in the explanation network is optimized with cross-entropy between the original prediction Y_o and the updated prediction $\hat{Y}_s$.

Algorithm 1: Training Algorithm for Explaining Node Classification.

1: **Input**: The input graph $G_o = (\mathcal{V}, \mathcal{E})$, node features $\mathbf{X}$, node labels Y, the set of instances to be explained $\mathcal{I}$, a trained GNN model: $\text{GNNE}_{\Phi_0}(\cdot)$ and $\text{GNNC}_{\Phi_1}(\cdot)$, and a parameterized explainer MLP_Ψ.
2: **for** each node $i \in \mathcal{I}$ **do**
3: $G_o^{(i)} \leftarrow$ extract the computation graph for node i.
4: $\mathbf{Z}^{(i)} \leftarrow \text{GNNE}_{\Phi_0}(G_o^{(i)}, \mathbf{X})$.
5: $Y_o^{(i)} \leftarrow \text{GNNC}_{\Phi_1}(\mathbf{Z}^{(i)})$.
6: **end for**
7: **for** each epoch **do**
8: **for** each node $i \in \mathcal{I}$ **do**
9: $\Omega \leftarrow$ latent variables calculated with (10)
10: **for** $k \leftarrow 1$ **to** K **do**
11: $\hat{G}_s^{(i,k)} \leftarrow$ sampled from (4).
12: $\hat{Y}_s^{(i,k)} \leftarrow \text{GNNC}_{\Phi_1}(\text{GNNE}_{\Phi_0}(\hat{G}_s^{(i,k)}, \mathbf{X}))$
13: **end for**
14: **end for**
15: Compute loss with (9).
16: Update parameters Ψ with backpropagation.
17: **end for**

Algorithm 2: Training Algorithm for Explaining Graph Classification.

1: **Input:** A set of input graphs with i-th graph represented by $G_o^{(i)}$, node features $\mathbf{X}^{(i)}$, and a label $Y^{(i)}$, a trained GNN model: $\text{GNNE}_{\Phi_0}(\cdot)$ and $\text{GNNC}_{\Phi_1}(\cdot)$, and a paramterized explainer MLP_Ψ.
2: **for** each graph $G_o^{(i)}$ **do**
3: $\mathbf{Z}^{(i)} \leftarrow \text{GNNE}_{\Phi_0}(G_o^{(i)}, \mathbf{X}^{(i)})$.
4: $Y_o^{(i)} \leftarrow \text{GNNC}_{\Phi_1}(\mathbf{Z}^{(i)})$.
5: **end for**
6: **for** each epoch **do**
7: **for** each graph $G_o^{(i)}$ **do**
8: $\Omega \leftarrow$ latent variables calculated with (11)
9: **for** $k \leftarrow 1$ **to** K **do**
10: $\hat{G}_s^{(i,k)} \leftarrow$ sampled from (4).
11: $\hat{Y}_s^{(i,k)} \leftarrow \text{GNNC}_{\Phi_1}(\text{GNNE}_{\Phi_0}(\hat{G}_s^{(i,k)}, \mathbf{X}^{(i)}))$
12: **end for**
13: **end for**
14: Compute loss with (9).
15: Update parameters Ψ with backpropagation.
16: **end for**

implement the $\Omega = g_\Psi(G_o, \mathbf{Z})$ to explain the prediction of node v with:

$$\omega_{ij} = \text{MLP}_\Psi([\mathbf{z}_i; \mathbf{z}_j; \mathbf{z}_v]). \tag{10}$$

MLP_Ψ is a multi-layer neural network parameterized with Ψ and $[\cdot;\cdot]$ is the concatenation operation.

The algorithms of PGExplainer for node and graph classification are shown in Algorithm 1. In GNNs with message passing mechanisms, the prediction at a node v is fully determined by its local computation graph, which is defined by its L-hop neighborhoods [58]. L is the number of GNN layers. Thus, for each node i in the instance set $\mathcal{I}$ to be explained, we first extract a local computation graph $G_o^{(i)}$ (line 3). With $G_o^{(i)}$ as the input graph, the trained GNN model generates the label of node i, denoted by $Y_o^{(i)}$ (line 4-5). To train the explanation network, each time we select a node i and compute parameters Ω in edge distributions with (10) (line 9). After that, we sample K graphs as input graphs for GNN to get updated predictions for node i, with the k-th prediction denoted by $\hat{Y}_s^{(i,k)}$ (line 11-13). We compute the loss and update parameters Ψ in the explanation network in line 15-16.

Explanation network for graph classification: For graph level tasks, each graph is considered as an instance. The explanation of the prediction of a graph is not conditional to a specific node. Therefore, we specify the $\Omega = g_\Psi(G_o, \mathbf{Z})$ for graph classification as:

$$\omega_{ij} = \text{MLP}_\Psi([\mathbf{z}_i; \mathbf{z}_j]). \tag{11}$$

With the graph classification as an example, the architecture of PGExplainer is shown in Fig. 2.

The training algorithm for explaining graph classification is shown in Algorithm 2. The algorithm is similar to the one explaining node classifications, except that computation graphs

are not used, because, for graph classification, each graph is treated as an instance. Given a set of graphs $\{G_o^{(i)}\}_{i\in\mathcal{I}}$, we first compute the node embeddings $\mathbf{Z}^{(i)}$ and graph labels $Y_o^{(i)}$ with the trained GNN model (line 2-4). In each epoch, for each i-th graph, we compute the parameters Ω in its edge distributions with (11) (line 8). We then sample K subgraphs and get the updated predictions. We compute the loss with (9) and update parameters Ψ with backpropagation.

Regularization: The framework of PGExplainer is flexible with various regularization terms to preserve desired properties on the explanation. We now discuss the regularization terms used in our experiments as well as their principles. Following [58], to obtain compact and succinct explanations, we can impose a constraint on the explanation size by adding $||\Omega||_1$, the l_1 norm on latent variables Ω, as a regularization term. Besides, element-wise entropy can also be applied to further achieve discrete edge weights [58].

Next, we provide more regularization terms that are compatible with PGExplainer. Note that for a fair comparison, the following regularization terms are not utilized in experimental studies in Section VI.

Budget constraint: To obtain a compact explanation, the l_1 norm on latent variables Ω was introduced, which penalizes all edge weights to sparsify the explanatory graph. In cases that a predefined budget B is available, for example, $|G_s| \leq B$, we could modify the size constraint to budget constraint:

$$R_b = \text{ReLU}\left(\sum_{(i,j)\in\mathcal{E}} \hat{e}_{ij} - B\right). \tag{12}$$

When the size of the explanatory graph is smaller than the budget B, the budget regularization $R_b = 0$. On the other hand, it works similarly to the size constraint when out of budget.

Connectivity constraint: In many real-life scenarios, determinant motifs are expected to be connected. Although it is claimed that GNNExplainer empirically tends to detect a small connected subgraph, the explicit constraints are not provided [58]. We propose to implement the connected constraint with the cross-entropy of adjacent edges, which connect to the same node. For instance, (i, j) and (i, k) both connected to the node i. The motivation is that is (i, j) is selected in the the explanatory graph, then its adjacent edge (i, k) should also be included. Formally, we design the connectivity constraint as:

$$H(\hat{e}_{ij}, \hat{e}_{ik}) = -[(1-\theta_{ij})\log(1-\theta_{ik}) + \theta_{ij}\log\theta_{ik}], \tag{13}$$

where $\theta_{ij} = \lim_{\tau\to 0} P(\hat{e}_{ij} = 1) = \frac{\exp(\omega_{ij})}{1+\exp(\omega_{ij})}$.

Computational complexity: PGExplainer is more efficient than GNNExplainer for two reasons. First, PGExplainer learns a latent variable for each edge in the original input graph with a neural network parameterized by Ψ, which is shared by all edges in the population of input graphs. Different from GNNExplainer, whose parameter size is linear to the number of edges, the number of parameters in PGExplainer is irrelevant to the size of the input graph, which makes PGExplainer applicable to large scale datasets. Further, since the explanation is shared among a population, a trained PGExplainer can be utilized in the inductive setting to explain new instances without retraining the explainer. To explain a new instance with $|\mathcal{E}|$ edges in the input graph, the time complexity of PGExplainer is $O(|\mathcal{E}|)$. As a comparison, GNNExplainer has to retrain for the new instance, leading to the time complexity of $O(T|\mathcal{E}|)$, where T is the number of epochs for retraining.

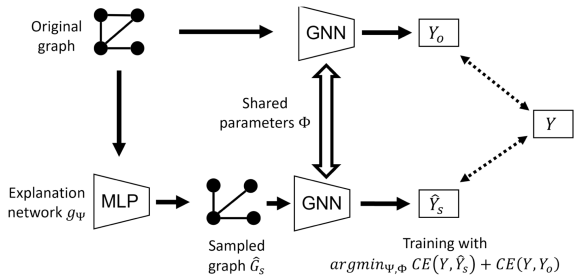


Fig. 3. Framework of self-explainable graph neural network via joint training.

D. Selection of Subset of Features

In this paper, we focus on globally understanding predictions made by GNNs by providing topological explanations. To explain node features, in GNNExplainer, the authors propose to use a feature mask to select features that are important to preserve original predictions. Feature selection has been extensively studied in non-graph neural networks and can be applied directly to explain GNN, such as the concrete autoencoder [1]. Besides, since selected features are shared among instances across the population, feature explanation is naturally global and applicable to new instances in the inductive setting.

V. Self-Explainable Graph Neural Network Via Joint Training

Graph data generated from real-life scenarios usually have complex and topological noise in the local neighborhood. Despite the success of existing GNN models on learning node/graph representations, task-irrelevant information is aggregated into node representations and propagated by stacked layers, leading to sub-optimal performances [66]. In addition, the quality of post-hoc explanations may be limited to the accuracy performance of the to-be-explained GNN model.

To address this problem, we propose to extend the PGExplainer to a self-explainable framework that can improve the accuracy performance and provide explanations simultaneously. Specifically, we consider the explanation network as a graph structure learning module, which is jointly trained with the downstream task. By explicitly extracting the informative subgraph, the proposed self-explainable framework can boost the generalization power of existing GNN models [39], [66].

With graph classification as the downstream task, The algorithm is shown in Algorithm 3, and the framework is shown in Fig. 3. The framework can be extended to other downstream tasks with minor modifications.

Given a set of graphs $\{G_o^{(i)}\}_{i=1}^N$, each graph is associated with a graph label $Y^{(i)}$, and a matrix storing node feature $\mathbf{X}^{(i)}$. A

Algorithm 3: Self-Explainable Graph Learning Framework for Graph Classification.

Input: A set of input graphs with i-th graph represented by $G_o^{(i)}$, node features $\mathbf{X}^{(i)}$, and a label $Y^{(i)}$, a GNN model: $\text{GNNE}_{\Phi_0}(\cdot)$ and $\text{GNNC}_{\Phi_1}(\cdot)$.
pre-train GNN model by minimizing (14).
for each epoch **do**
  for each graph $G_o^{(i)}$ **do**
    $\mathbf{Z}^{(i)} \leftarrow \text{GNNE}_{\Phi_0}(G_o^{(i)}, \mathbf{X}^{(i)})$.
    $Y_o^{(i)} \leftarrow \text{GNNC}_{\Phi_1}(\mathbf{Z}^{(i)})$.
    $\Omega \leftarrow$ latent variables calculated with (11)
    for $k \leftarrow 1$ **to** K **do**
      $\hat{G}_s^{(i,k)} \leftarrow$ sampled from (4).
      $\hat{Y}_s^{(i,k)} \leftarrow \text{GNNC}_{\Phi_1}(\text{GNNE}_{\Phi_0}(\hat{G}_s^{(i,k)}, \mathbf{X}^{(i)}))$
    end for
  end for
  Compute loss with (16).
  Update parameters Φ and Ψ with backpropagation.
end for

GNN model has two components: an encoder network GNNE_{Φ_0} and a classifier network GNNC_{Φ_1}. $\Phi = \{\Phi_0, \Phi_1\}$ denotes the parameters in the GNN model. Since the explanation network takes node embeddings as the input, to avoid optimizing Φ and Ψ from scratch simultaneously, we first pre-train the GNN model with the cross-entropy loss:

$$\sum_i \text{CE}(Y^{(i)}, Y_o^{(i)}), \tag{14}$$

where $Y_o^{(i)} = \text{GNNC}_{\Phi_1}(\text{GNNE}_{\Phi_0}(G_o^{(i)}, \mathbf{X}^{(i)}))$ is the output of GNN model with graph $G_o^{(i)}$ as the input.

In the second phase, parameters in both the explanation network and GNN model, Ψ and Ω, are jointly optimized. Besides, the loss in (14), for a graph $G_o^{(i)}$, we also consider the cross-entropy between $Y^{(i)}$ and $\hat{Y}_s^{(i,k)}$:

$$\frac{1}{K}\sum_{k=1}^{K} \text{CE}(Y^{(i)}, \hat{Y}_s^{(i,k)}), \tag{15}$$

where $Y_s^{(i,k)} = \text{GNNC}_{\Phi_1}(\text{GNNE}_{\Phi_0}(G_s^{(i,k)}, \mathbf{X}^{(i)}))$ is the prediction of GNN with sampled subgraph $G_s^{(i,k)}$ as the input and K is the number of Monte Carlo sampling. The explanation network is optimized following (9) to identify sparse yet discriminative sub-graphs. Putting everything together, we have the following loss function:

$$\sum_i [\text{CE}(Y^{(i)}, Y_o^{(i)}) + \frac{1}{K}\sum_{k=1}^{K} (\text{CE}(Y^{(i)}, \hat{Y}_s^{(i,k)})]. \tag{16}$$

VI. Experimental Study

In this section, we evaluate our PGExplainer with a number of experiments. We first describe synthetic and real-world datasets, baseline methods, and experimental setup. Then, we present the experimental results on explanations of both node and graph classification. With qualitative and quantitative evaluations, we demonstrate that our PGExplainer can improve the SOTA method up to 24.7% in AUC on explaining graph classification. At the same time, with a trained explanation network, our PGExplainer is significantly faster than the baseline when explaining unexplained instances.

A. Datasets

We follow the setting in GNNExplainer and construct four kinds of node classification datasets, BA-Shapes, BA-Community, Tree-Cycles, and Tree-Grids [58]. Furthermore, we also construct a graph classification dataset, BA-2motifs. (1) BA-Shapes is a single graph consisting of a base Barabasi-Albert (BA) graph with 300 nodes and 80 "house"-structured motifs. These motifs are attached to randomly selected nodes from the BA graph. After that, random edges are added to perturb the graph. Nodes features are not assigned in BA-Shapes. Nodes in the base graph are labeled with 0; the ones locating at the top/middle/bottom of the "house" are labeled with 1,2,3, respectively. (2) BA-Community dataset consists of two BA-Shapes graphs. Two Gaussian distributions are utilized to sample node features, one for each BA-Shapes graph. Nodes are labeled based on their structural roles and community memberships, leading to 8 classes in total. (3) In the Tree-Cycles dataset, an 8-level balanced binary tree is adopted as the base graph. A set of 80 six-node cycle motifs are attached to randomly selected nodes from the base graph. (4) Tree-Grid is constructed in the same way as TREE-CYCLES, except that 3-by-3 grid motifs are used to replace the cycle motifs. (5) For graph classification, we build the BA-2motifs dataset of 1000 graphs. We adopt the BA graphs as base graphs. Half graphs are attached with "house" motifs and the rest are attached with five-node cycle motifs. Graphs are assigned to one of 2 classes according to the type of attached motifs. The first four datasets are also used in [58] for node classification. For datasets without node features, we use a 10-dimensional vector with all elements set to 1 [58].

We also include a real-life dataset, MUTAG, for graph classification, which is also used in previous work [58]. It consists of 4,337 molecule graphs. Each graph is assigned to one of 2 classes based on its mutagenic effect [46], [58]. As discussed in [11], [58], carbon rings with chemical groups NH_2 or NO_2 are known to be mutagenic. We observe that carbon rings exist in both mutagen and nonmutagenic graphs, which are not discriminative. Thus, we can treat carbon rings as the shared base graphs and NH_2, NO_2 as motifs for the mutagen graphs. There are no explicit motifs for nonmutagenic ones.

Table I shows the statistics of synthetic and real-life datasets.

B. Baselines and Experimental Setup

Baselines: We compare with the SOTA method, GNNExplainer as well as other baselines in [58], i.e., a gradient-based method (GRAD), and graph attention network (ATT). (1) GNNExplainer is a post-hoc method providing explanations for every single instance. (2) GRAD learns weights of edges by computing gradients of GNN's objective function w.r.t. the adjacency

TABLE I
DATASET STATISTICS

	Node Classification				Graph Classification	
	BA-Shapes	BA-Community	Tree-Cycles	Tree-Grid	BA-2motifs	MUTAG
#graphs	1	1	1	1	1,000	4,337
#nodes	700	1,400	871	1,231	25,000	131,488
#edges	4,110	8,920	1,950	3,410	51,392	266,894
#labels	4	8	2	2	2	2

TABLE II
ACCURACY PERFORMANCE OF GNN MODELS

	Node Classification				Graph Classification	
Accuracy	BA-Shapes	BA-Community	Tree-Cycles	Tree-Grid	BA-2motifs	MUTAG
Training	0.98	0.99	0.99	0.92	1.00	0.87
Validation	1.00	0.88	1.00	0.94	1.00	0.89
Testing	0.97	0.93	0.99	0.94	1.00	0.87

matrix. (3) ATT utilizes self-attention layers to distinguish edge attention weights in the input graph. Each edge's importance is obtained by averaging its attention weights across all attention layers.

Experimental setup: All experiments are conducted on a Linux machine with an Nvidia GeForce RTX 2070 SUPER GPU with 8 GB memory. CUDA version is 10.2 and Driver Version is 440.64.00. We follow the experimental settings in GNNExplainer [58]. Specifically, for post-hoc methods including ATT, GNNExplainer, and PGExplainer, we first train a three-layer GNN and then apply these methods to explain predictions made by the GNN. For a fair comparison, we exactly use the specially designed GNN model in GNNExplainer [58]. Each GNN layer is represented by $f(H^{(l)}, A) = \sigma(W^{(l)}AH^{(l)})$, where $H^{(l)}$ is the hidden representations of nodes in the l-th layer, A is the normalized Laplacian matrix, and $W^{(l)}$ is the weight matrix. We adopt the Adam optimizer with an initial learning rate of 1.0×10^{-3}. All variables are initialized with Xavier. We follow GNNExplainer to split train/validation/test with 80/10/10% for all datasets. Each model is trained for 1000 epochs. The accuracy performances of GNN models are shown in Table II. The results show that the designed GNN models are powerful enough for node/graph classifications on both synthetic and real-life datasets. Since weights in attention layers are jointly optimized with the GNN model in ATT, we thus train another GNN model with self-attention layers. We follow [1] to tune temperature τ.

The network structure of explanation networks in PGExplainer is MLPs with one hidden layer. To train PGExplainer, we also adopt the Adam optimizer with the initial learning rate of 3.0×10^{-3}. The coefficient of size regularization is set to 0.05 and entropy regularization is 1.0. The epoch T is set to 30 for all datasets. The temperature τ in (4) is set with an annealing schedule [1]: $\tau^{(t)} = \tau_0(\tau_T/\tau_0)^t$, where τ_0 and τ_T are the initial and final temperatures. A small temperature tends to generate more discrete graphs which may hinder the explanation network from being optimized with backpropagation. In this task, we find that relatively high temperatures work well in practice.

C. Comparison With Baselines

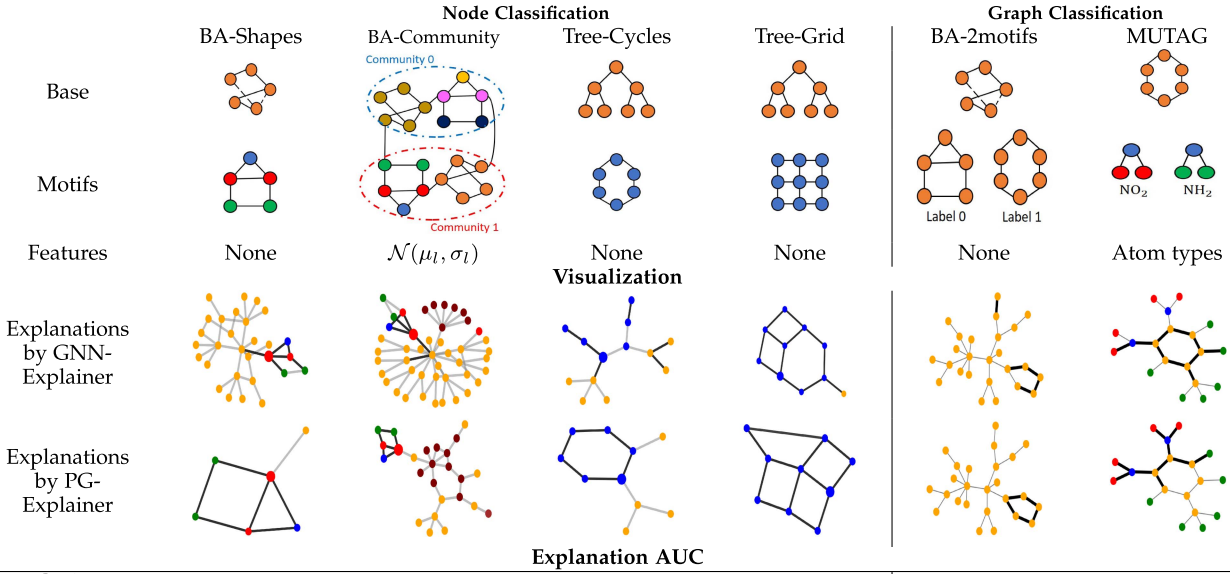
The results of comparative evaluation experiments on both synthetic and real-life datasets are summarized in Table III . In these datasets, node/graph labels are determined by the motifs, which are treated as ground truth explanations. These motifs are utilized to calculate explanation accuracy for PGExplainer as well as other baselines.

Qualitative evaluation: We choose an instance for each dataset and visualize its explanations given by GNNExplainer and PGExplainer in Table III. In these explanations, bold black edges indicate top-K edges ranked by their importance weights, where K is set to the number of edges inside motifs for synthetic datasets and 10 for MUTAG [58]. As demonstrated in these figures, the whole motifs, such as "house" in BA-Shapes and BA-Community, cycles in Tree-Cycles and BA-2motifs, grids in Tree-Grid, and NO_2 groups in MUTAG are correctly identified by PGExplainer. On the other hand, some important edges are missing in the explanations given by GNNExplainer. For example, the explanation provided by GNNExplainer for the instance in MUTAG contains the carbon rings and part of a NO_2 group. However, the carbon rings appear frequently in both mutagen and nonmutagenic graphs, which are not discriminative. Conversely, PGExplainer correctly identifies both NO_2 groups.

Quantitative evaluation: We follow the experimental settings in GNNExplainer [58] and formalize the explanation problem as a binary classification of edges. We treat edges inside motifs as positive edges, and negative otherwise. Importance weights provided by explanation methods are considered as prediction scores. A good explanation method assigns high weights to edges in the ground truth motifs than the ones outside. AUC is adopted as the metric for quantitative evaluation. Especially, for the MUTAG dataset, we only consider the mutagen graphs because no explicit motifs exist in nonmutagenic ones. For PGExplainer, we repeat each experiment 10 times and report the average AUC scores and standard deviations here.

From the table, we have the following observations. PGExplainer achieves SOTA performances in all scenarios and the accuracy gains are up to 13.0% in node classification and

TABLE III
ILLUSTRATION OF DIFFERENT DATASETS TOGETHER WITH PERFORMANCE EVALUATION OF PGEXPLAINER AND OTHER BASELINES



	BA-Shapes	BA-Community	Tree-Cycles	Tree-Grid	BA-2motifs	MUTAG
Explanation AUC						
GRAD	0.882	0.750	0.905	0.667	0.717	0.783
ATT	0.815	0.739	0.824	0.612	0.674	0.765
Gradient	-	-	-	-	0.773	0.653
GNNExplainer	0.925	0.836	0.948	0.875	0.742 ± 0.001	0.727 ± 0.014
PGExplainer	0.963±0.011	0.945±0.019	0.987±0.007	0.907±0.014	0.926±0.021	0.873±0.013
Improve	4.1%	13.0%	4.1%	3.7%	24.7%	11.5%
AVG. Time (ms)						
GNNExplainer	650.60	696.61	690.13	713.40	934.72	409.98
PGExplainer (w/ Training)	4.62e3	2.54e4	6.09e3	1.85e4	9.87e3	4.26e4
PGExplainer (Inference)	10.92	24.07	6.36	6.72	80.13	9.68
Speed-up	59x	29x	108x	106x	12x	42x

BA-Shapes, BA-Community, Tree-Cycles, Tree-Grid are datasets for node classification [58]. Node labels are represented by their colors. BA-2motifs and MUTAG datasets are used for graph classification. Graphs with "house" motifs are labeled with 0 and the ones with cycles are with 1 in BA-2motifs dataset. NH2, NO2 are treated as motifs of the mutagen graphs in MUTAG. Explanations extracted by GNNExplainer and PGExplainer are also shown as case studies.

24.7% in graph classification. Compared to GNNExplainer, which tackles instances independently thus can only achieve suboptimal explanations, PGExplainer utilizes a parameterized explanation network based upon graph generative model to collectively provide explanations for multiple instances. As a result, PGExplainer can provide a global understanding of the GNNs. That answers why PGExplainer can outperform GNNExplainer by relatively large margins.

Fidelity evaluation: We adopt the experimental framework outlined in [61] for assessing the Fidelity of the generated explanations. The Fidelity metric evaluates the degree to which the explanations genuinely reflect the model's predictions by sequentially remove edges based on their weights in explanation and testing the classification performance afterwards. The removal of more important edges would degrade the classification performance more significantly. Thus, a faster performance drop would denote stronger fidelity and better explanations.

The results on two node classification datasets, BA-Shapes and Tree-Grid, are reported in Fig. 4, where we plot the curves of model accuracy with respect to the number of removed edges. The removal of edges is ordered based on their weights in explanation. As shown in Fig. 4, PGExplainer outperforms

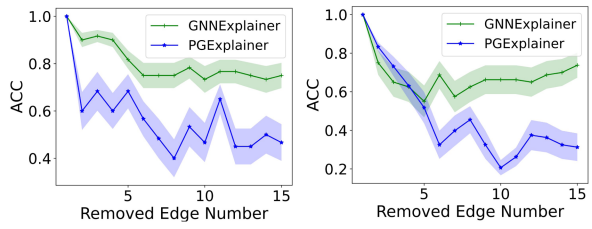


Fig. 4. Fidelity analysis by removing identified explanatory edges.

GNNExplainer by a significant margin and show an improved fidelity, which further verifies the effectiveness of the PGExplainer.

Efficiency evaluation: Explanation network in PGExplainer is shared across the population of instances. Thus, a trained PGExplainer can be utilized to explain new instances in the inductive setting. We denote the time to explain a new instance with a trained explanation method by inference time. Since GNNExplainer has to retrain the model, we also count the training time here. The running time comparison in Table III shows that

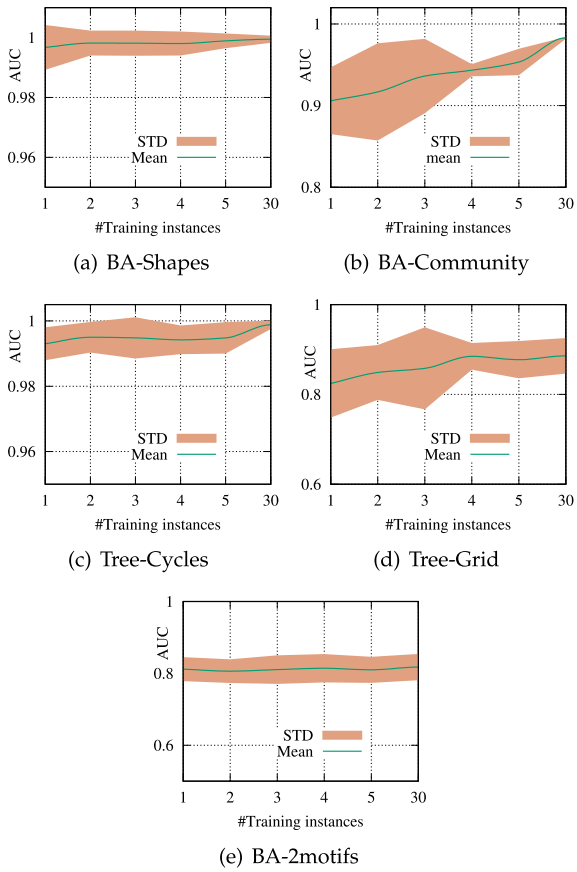


Fig. 5. Evaluation of PGExplainer in the inductive setting.

PGExplainer can speed up the computation significantly, up to 108 times faster than GNNExplainer, which makes PGExplainer more practical for large-scale datasets.

D. Inductive Performance

As we discussed in Section IV-C, the explanation network is shared across the population. Thus, with a trained PGExplainer, we can directly infer the explanation without retraining the explanation network. As a result, our PGExplainer has better generalization power than the leading baseline GNNExplainer. Besides, our PGExplainer is more efficient in the inductive setting. In this section, we empirically demonstrate the effectiveness of PGExplainer in the inductive setting. In the inductive setting, we select α instances for training, $(N-\alpha)/2$ for validation, and the rest for testing. α is ranged from $[1, 2, 3, 4, 5, 30]$. Note that, with $\alpha = 1$, our method degenerates to the single-instance explanation method. Recall that to explain a set of instances, GNNExplainer first detects a reference node and then computes the explanation for the reference node. The explanation is then generalized to other nodes with graph alignment [58]. We claim that it may lead to sub-optimal explanations because reference node selection and graph alignment are not jointly optimized with the explanation in an end-to-end fashion. The AUC scores of PGExplainer are shown in Fig. 5. We have the following observations. 1) The testing AUC increase as more instances are trained, verifying the effectiveness of PGExplainer. Some results are higher than the reported ones in Section VI because here we adopt validation datasets to fine-tune the hyper-parameters. 2) More training instances lead to smaller standard deviation, PGExplainer tends to globally detect shared motifs with higher robustness. 3) PGExplainer can achieve relatively good performance with a small number of trained instances, which makes PGExplainer more practical in large datasets. The results also explain why we dismiss the training time of PGExplainer and only count the inference time in Section VI.

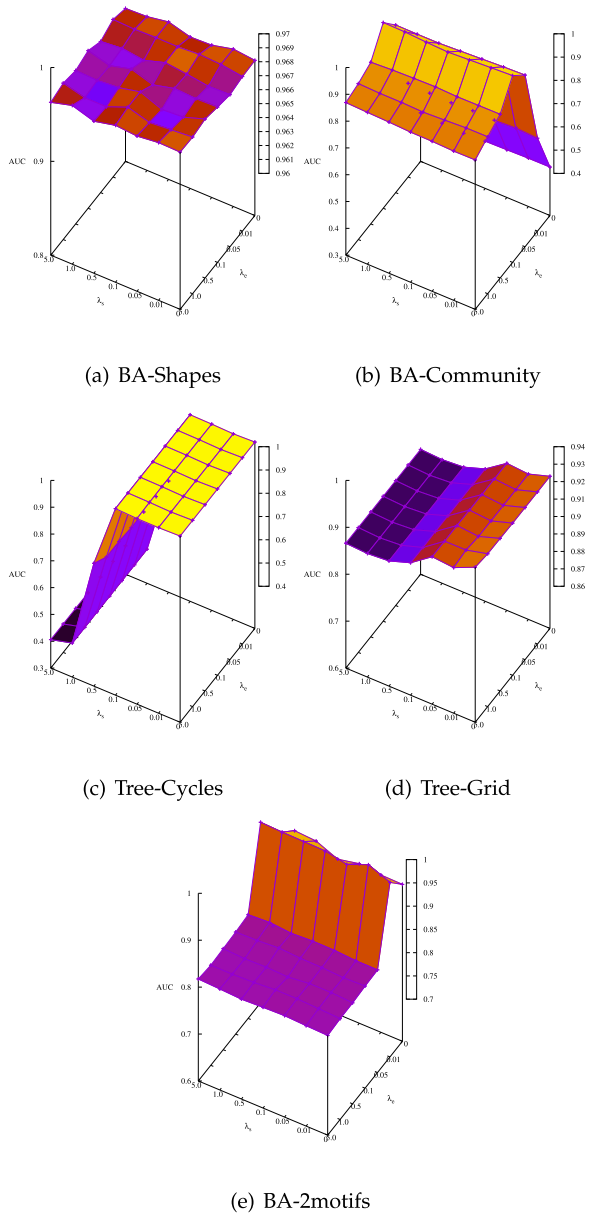
E. Model Analysis

1) Effects of Regularization Terms: In this part, we analyze the effects of regularization terms. In addition to the size and entropy regularizers introduced in GNNExplainer, we also have discussed regularization terms on budgets and connectivity constraints. Since the first two regularizers are used in the quantitative evaluation, we first conduct parameter studies. Visualization results on synthetic datasets show that the explanatory graph extract by PGExplainer tends to be small and compact. To verify the effectiveness of the proposed regularizer for connectivity constraint, we synthesize a noisy BA-Shapes dataset.

Effects of size and entropy constraint: We select synthetic datasets for parameter study. The coefficients of size and entropy regularizers are denoted by λ_s and λ_e, respectively. AUC scores w.r.t coefficients are shown in Fig. 6. We observe that PGExplainer achieves competitive performances even without any regularization terms in all datasets except the BA-Community, which verifies the effectiveness of the model itself. For the BA-Community dataset, the entropy constraint plays an important role.

Effects of connectivity constraint: To show the effect of the connectivity constraint on the explanatory graph, we build a noisy BA-Shapes dataset with $0.2\,N$ noisy edges and vary the coefficient of the connectivity regularization term λ_c from 0 to 10 and apply PGExplainer to explain a single instance. The visualization results with regard to different choices of coefficients are shown in Fig. 7. An analysis of the number of connected components is also conducted for Tree-Cycles and Tree-Grid, with the result presented in Fig. 8. Note that in the ideal scenario, the number of connected components should be 1. Therefore, the number of connected components dropping from 1.62 to 1.50 denotes an improvement of 20%. These figures demonstrate that without explicit constraint, PGExplainer may detect several connected edges in the noisy input graph, although these edges are also inside motifs. With the connectivity constraint, we observe that PGExplainer tends to reduce the number of connected components and enhance the connectivity of the explanation subgraph.

2) Effects of Stochastic Sampling: To show the effects of the stochastic subgraph sampling, we vary the distribution range of the uniform variable ϵ in (6). In the original PGExplainer, the independent variable ϵ is sampled from the distribution $\epsilon \sim \text{Uniform}(0, 1)$. The stochastic magnitude can be adjusted by including another hyper-parameter s, that $\epsilon \sim \text{Uniform}(s, 1-s)$, where $s \in [0, 0.5]$. A larger value of s indicates a smaller

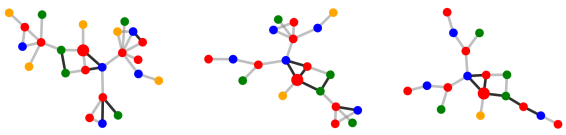


(a) BA-Shapes (b) BA-Community

(c) Tree-Cycles (d) Tree-Grid

(e) BA-2motifs

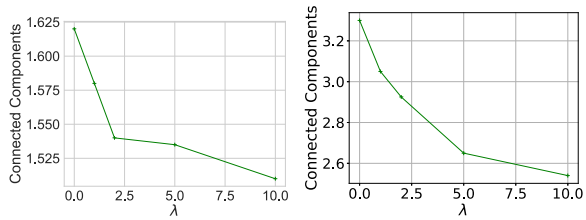
Fig. 6. Effects of size and entropy constraint.



(a) λ_c=0 (b) λ_c=5 (c) λ_c=10

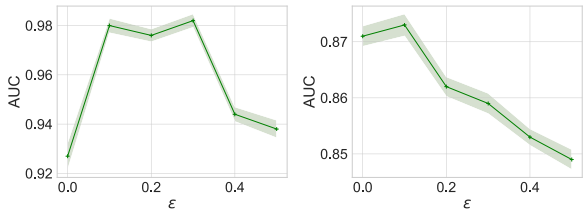
Fig. 7. Effects of connectivity constraint.

stochastic magnitude. And $s = 0.5$ is the deterministic version of the PGExplainer. We adopt the Tree-Cycles and MUTAG datasets in this section and results are shown in Fig. 9, From the figure, We observe that stochastic sampling significantly and consistently improves the explanation performances in the



(a) Tree_Grid (b) Tree_Cycles

Fig. 8. Quantitative evaluation of of connectivity constraint.



(a) Tree_Cycles (b) Mutag

Fig. 9. Influence of stochastic sampling variable ϵ.

MUTAG datasets. With $\epsilon \in [0.1, 0.3]$, PGExplainer outperforms its deterministic variant in the Tree-Cycles.

3) GCN as the Backbone: To show the robustness of the PGExplainer, in this section, we compare to GNNExplainer with a representative and commonly used GNN model, GCN [31]. Other settings are kept the same with Section VI-B. We adopt the implementations in [28]. The accuracy performances of GCN and explanation performances of explainers are shown in Table IV.

In general, GCN works worse than the specifically designed GNN in [58]. As a result, the explanation performances of both GNNExplainer and PGExplainer drop. However, the relative improvement of PGExplainer over GNNExplainer remains with GCN as the backbone. PGExplainer outperforms GNNExplainer on 5 out of 6 datasets, except the Tree-Grid datasets. Specifically, AUC gains of PGExplainer over GNNExplainer are up to 40.7% on node classification and 43.6% on graph classification, respectively, showing the robustness of the proposed PGExplainer over the leading baseline. In addition, PGExplainer is 6 to 103 times faster than GNNExplainer.

4) More Choices of Explanation Networks: In PGExplainer, we adopt a two-layer MLP as the explanation network. To show the robustness of the PGExplainer and investigate the effects of other types of neural networks as the explainer, in this section, we include linear (Linear), MLP with 3 layers (MLP3), and GCN (GCN) to replace the vanilla MLP in PGExplainer. Results on 4 dataset are shown in Fig. 10.

From the figure, we observe that our PGExplainer is robust to the choice of explainer architectures. More sophisticated GNN methods, such as GCN fails to further improve the explanation performances. The reason is the node embeddings used in parameterized explainer are obtained by aggregating both node features and topological structures. Linear models and simple MLPs are sufficient to extract informative subgraphs from them.

TABLE IV
COMPARISON TO GNNEXPLAINER WITH GCN AS THE BACKBONE

	Node Classification				Graph Classification	
	BA-Shapes	BA-Community	Tree-Cycles	Tree-Grid	BA-2motifs	MUTAG
Accuracy performance						
Training	0.97	0.75	0.94	0.91	0.96	0.82
Validation	1.00	0.67	0.98	0.94	1.00	0.87
Testing	1.00	0.71	0.94	0.93	0.95	0.80
Explanation AUC						
GNNExplainer	0.742±0.007	0.708±0.004	0.540±0.017	0.714 ± 0.003	0.498 ± 0.004	0.587 ± 0.002
PGExplainer	0.999±0.000	0.825±0.040	0.762±0.014	0.683±0.008	0.566±0.042	0.843±0.084
Improve	34.6%	16.5%	40.7%	-4.3%	13.6%	43.6%
Inference Time (ms)						
GNNExplainer	412.0	575.5	360.9	428.3	218.6	91.7
PGExplainer	28.1	41.7	3.5	5.6	7.5	15.9
Speed-up	15x	14x	103x	76x	29x	6x

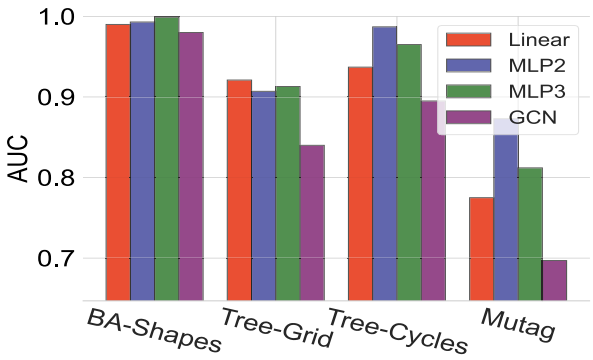


Fig. 10. Comparison of different explainer structures.

The conclusion is aligned with the previous graph generation methods, where simple decoders are utilized [30].

F. Application to Brain Network Analysis

Analyzing the relationship between human behavioral measures and their functional brain connectivity, as measured by resting-state functional magnetic resonance imaging (fMRI) correlations, is a fundamental task in the Neuroscience field [8], [9]. A set of 460 subjects from the Human Connectome Project (HCP) are utilized in this case study [52], denoted by $\mathcal{S}$. For each subject $s \in \mathcal{S}$, we generated the functional connectivity of brain networks using resting-state fMRI data, where nodes denote brain regions and edges describe their functional connectivity. Each brain connectivity network is a fully connected network with 76 nodes. Each node is associated with the cortical myelination level, which was measured by the ratio of T1 and T2 ration [18], and cortical thickness of the corresponding region. In addition, each subject is associated with 151 behavioral measures, including Pittsburgh Sleep Quality Index (PSQI) score, NIH Toolbox Sadness Survey: Unadjusted Scale Score, et. al. A canonical-correlation analysis (CCA) score has been calculated for each subject to summarize his/her position along a direction that maximally linked these behavioral measures to the neuroimaging data [24]. Based on their CCA scores, we split the 460 subjects into two groups, "pos" and "neg". A subject is assigned to the "pos" set, denoted by $\mathcal{S}_{pos}$ if the CCA score is positive. Otherwise, s/he is assigned to the "neg" set $\mathcal{S}_{neg}$.

A three-layer GNN model is utilized to predict a subject's group label based on its brain network. For subject i, we adopt PGExplainer to learn a weight matrix $\Omega_i \in \mathbb{R}^{76\times 76}$, which measures the strength of the relationship between functional connectivity and the subject's behavior measurements. Specifically, the diagonal elements in Ω_i reflect the importance of each brain region. To analyze the relationship between human behaviors and brain network connectivity at a group level, we consider three groups. $\Omega_{pos} = \frac{1}{|\mathcal{S}_{pos}|}\sum_{i\in\mathcal{S}_{pos}} \Omega_i$ is the average weight matrices of subjects in the "pos" set $\mathcal{S}_{pos}$; $\Omega_{neg} = \frac{1}{|\mathcal{S}_{neg}|}\sum_{i\in\mathcal{S}_{neg}} \Omega_i$ is for the "neg" set; $\Omega_{all} = \frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}} \Omega_i$ is for all these 460 subjects. For each group, we extracted the diagonal elements from the corresponding matrix and displayed the results on a brain surface. The results for these three groups are shown in Fig. 11(a), (b), and (c), respectively. Regions with yellow color are associated with higher weights, indicating critical impacts on the behavior measurements.

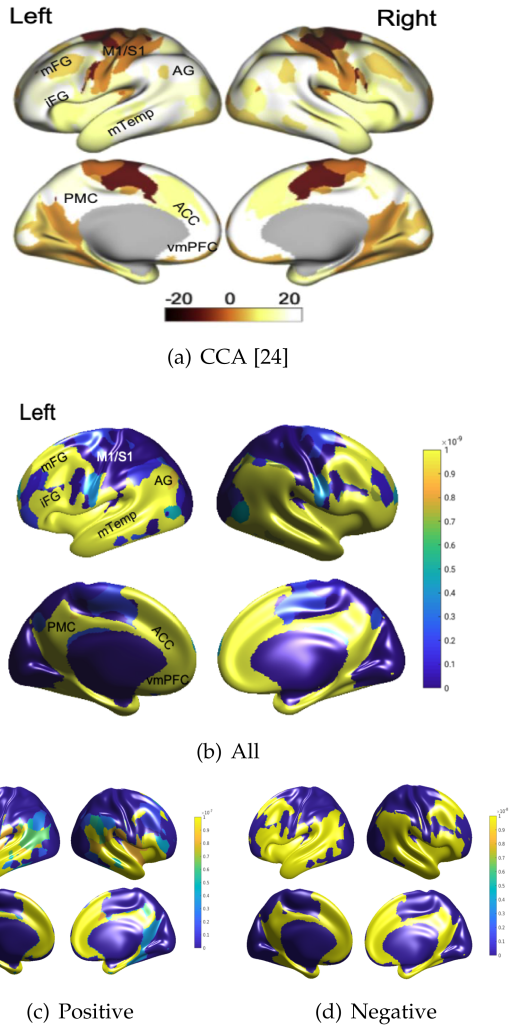
Based on these figures, we have the following observations. First, the overall spatial patterns of the "pos", "neg", and "all" groups are consistent with the patterns found by traditional canonical-correlation analysis (Fig. 11(a)), especially the contrast between the higher-order/hierarchy association regions (middle frontal gyrus [mFG], inferior frontal gyrus [iFG], angular gyrus [AG], middle temporal cortex [mTemp], anterior cingulate cortex [ACC], posteromedial cortex [PMC], ventromedial prefrontal cotex [vmPFC]) and lower-order primary sensory-motor networks (primary motor cortex [M1], primary somatosensory cortex [S1]), are close to each other. Of note, this cross-hierarchy pattern was the core feature emphasized in [24]. On the other hand, we also observe the difference between spatial patterns of "pos" and "neg" groups. To measure the consistency quantitatively, we further calculate the Spearman's rank correlation coefficient between the patterns discovered by us and those by traditional analysis. Across 76 cortical regions (Fig. 11(a)), we get the correlation coefficient as 0.3346 and the p-value as 0.0031, which validate the significance of consistency between two panels.

G. Self-Explainable Graph Learning

In this section, we evaluate the effectiveness of the self-explainable graph learning framework. First, we compare the

TABLE V
DATASET STATISTICS

Dataset	PROTEINS	BZR_MD	Cuneiform	PTC_FR	PTC_MR
#graphs	1,113	306	267	351	344
Avg. # nodes	39.06	21.30	21.27	14.56	14.29
#edges	72.82	225.06	44.80	15.00	14.69
#labels	2	2	30	2	2



(a) CCA [24]

(b) All

(c) Positive (d) Negative

Fig. 11. Case study on brain network analysis.

TABLE VI
PERFORMANCE ANALYSIS OF OUR SELF-EXPLAINABLE LEARNING FRAMEWORK

		BA-Shapes	Tree-Cycles	Tree-Grid
AUC	GNNExpl	0.74	0.54	0.71
	PGExpl	0.99	0.76	0.68
	SelfExpl	0.99	0.78	0.82
ACC	Backbone	1.00	0.94	0.93
	SelfExpl	1.00	0.98	0.92

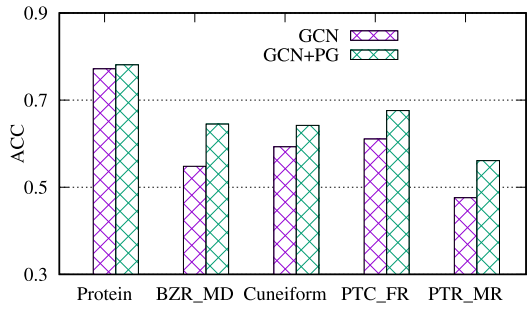


Fig. 12. Effects of utilizing explanation networks with size constraint as the sparsification constraint.

explanations achieved by the self-explainable method with post-hoc counterparts. We include GNNExplainer and PGExplainer as baselines and report the results w.r.t both the explanation quality (in terms of AUC on edges) and classification performance (in terms of ACCuracy), with the result in Table VI. It can be observed that our self-explainable framework can achieve comparable or better explanation quality with PGExplainer and bring improvements to downstream classification in most cases.

Second, we verify the accuracy performance of the self-explain GNN method with 5 datasets from different fields,[1] including PROTEINS [3], BZR_MD [51], Cuneiform [32], PTC_FR [25] and PTC_MR [25]. Dataset statistics are shown in Table V.

Three-layer GCNs are adopted as the backbone. Readout operation is then applied to get the graph embeddings from node embeddings. We apply explanation networks with size constraints to the GCN, denoted by GCN+PG, and compare it to the basic one. The comparison results are shown in Fig. 12. Our GCN+PG consistently outperforms GCN in all these five datasets with up to 17.80% achievement in classification accuracy, verifying the effectiveness of our method in the performance of graph classification.

VII. CONCLUSION

We present PGExplainer, a parameterized method to provide a global understanding of any GNN models on arbitrary machine learning tasks by collectively explaining multiple instances. We show that PGExplainer can leverage the representations produced by GNNs to learn the underlying subgraphs that are important to the predictions made by them. Furthermore, PGExplainer is more efficient due to its capacity to explain GNNs in inductive settings, which makes PGExplainer more practical in real-life applications. In addition, we show that the explanation network with size constraint can also extract informative structures in the input graph to improve the performance of GNNs.

[1]https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets

REFERENCES

[1] A. Abid, M. F. Balin, and J. Zou, "Concrete autoencoders for differentiable feature selection and reconstruction," 2019, *arXiv:1901.09346*.

[2] R. Arora and J. Upadhyay, "On differentially private graph sparsification and applications," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 13378–13389.

[3] K. M. Borgwardt et al., "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. suppl_1, pp. i47–i56, 2005.

[4] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," 2013, *arXiv:1312.6203*.

[5] R. Caruana, Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad, "Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2015, pp. 1721–1730.

[6] J. Chen, L. Song, M. Wainwright, and M. Jordan, "Learning to explain: An information-theoretic perspective on model interpretation, in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2018, pp. 883–892.

[7] L P. Chew, "There are planar graphs almost as good as the complete graph," *J. Comput. Syst. Sci.*, vol. 39, no. 2, pp. 205–219, 1989.

[8] H. Cui et al., "BrainGB: A benchmark for brain network analysis with graph neural networks," *IEEE Trans. Med. Imag.*, vol. 42, no. 2, pp. 493–506, Feb. 2023.

[9] H. Cui, W. Dai, Y. Zhu, X. Li, L. He, and C. Yang, "Interpretable graph neural networks for connectome-based brain disorder analysis," in *Proc. 25th Int. Conf. Med. Image Comput. Comput. Assist. Intervent.*, Singapore, Springer, 2022, pp. 375–385.

[10] P. Dabkowski and Y. Gal, "Real time image saliency for black box classifiers," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6967–6976.

[11] A. Kumar Debnath, R. L. L. de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity," *J. Med. Chem.*, vol. 34, no. 2, pp. 786–797, 1991.

[12] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 3844–3852.

[13] P. Erdds and A. R&wi, "On random graphs I," *Pub. Math. Debrecen*, vol. 6, pp. 290–297, 1959.

[14] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing higher-layer features of a deep network," *Univ. Montreal*, vol. 1341, no. 3, 2019, Art. no. 1.

[15] C. R. Fong and A. Vedaldi, "Interpretable explanations of black boxes by meaningful perturbation," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 3429–3437.

[16] Edgar N. Gilbert, "Random graphs," *Ann. Math. Statist.*, vol. 30, no. 4, pp. 1141–1144, 1959.

[17] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1263–1272.

[18] F. MatthewGlasser and D. C.V. Essen, "Mapping human cortical areas in vivo based on myelin content as revealed by T1-and T2-weighted MRI," *J. Neurosci.*, vol. 31, no. 32, pp. 11597–11616, 2011.

[19] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proc. Int. Joint Conf. Neural Netw.*, 2005, pp. 729–734.

[20] A. Grover, A. Zweig, and S. Ermon, "Graphite: Iterative generative modeling of graphs," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 2434–2444.

[21] W. Guo, S. Huang, Y. Tao, X. Xing, and L. Lin, "Explaining deep learning models–a Bayesian non-parametric approach," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 4514–4524.

[22] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, "LEMNA: Explaining deep learning based security applications," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 364–379.

[23] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1024–1034.

[24] F. Han, Y. Gu, G. L. Brown, X. Zhang, and X. Liu, "Neuroimaging contrast across the cortical hierarchy is the feature maximally linked to behavior and demographics," *Neuroimage*, vol. 215, 2020, Art. no. 116853.

[25] C. Helma, R. D. King, S. Kramer, and A. Srinivasan, "The predictive toxicology challenge 2000–2001," *Bioinformatics*, vol. 17, no. 1, pp. 107–108, 2001.

[26] G. B. Hermsdorff and L. M. Gunderson, "A unifying framework for spectrum-preserving graph sparsification and coarsening," 2019, *arXiv:1902.09702*.

[27] P. D. Hoff, A. E. Raftery, and M. S. Handcock, "Latent space approaches to social network analysis," *J. Amer. Stat. Assoc.*, vol. 97, no. 460, pp. 1090–1098, 2002.

[28] L. Holdijk, M. Boon, S. Henckens, and L. deJong, "[Re] parameterized explainer for graph neural network," *ReScience C*, vol. 7, no. 2, May 2021, doi: 10.5281/zenodo.4834242.

[29] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *Proc. Int. Conf. Learn. Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=rkE3y85ee

[30] N. T. Kipf and M. Welling, "Variational graph auto-encoders," 2016, *arXiv:1611.07308*.

[31] N. T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2017.

[32] N. M. Kriege, M. Fey, D. Fisseler, P. Mutzel, and F. Weichert, "Recognizing cuneiform signs using graph based methods," in *Proc. Int. Workshop Cost-Sensitive Learn.*, PMLR, 2018, pp. 31–44.

[33] H. Lakkaraju, E. Kamar, R. Caruana, and J. Leskovec, "Interpretable & explorable approximations of black box models," 2017, *arXiv:1707.01154*.

[34] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: Densification laws, shrinking diameters and possible explanations," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2005, pp. 177–187.

[35] J. Li, W. Monroe, and D. Jurafsky, "Understanding neural networks through representation erasure," 2016, *arXiv:1612.08220*.

[36] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 3546–3553.

[37] S. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," Proc. Int. Conf. Neural Inf. Process. Syst., 2017, pp. 4765–4774.

[38] M. S. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4765–4774.

[39] D. Luo et al., "Learning to drop: Robust graph neural network via topological denoising," in *Proc. ACM Int. Conf. Web Search Data Mining*, 2021, pp. 779–787.

[40] J. Ma, P. Cui, K. Kuang, X. Wang, and W. Zhu, "Disentangled graph convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 4212–4221.

[41] J. Ma, W. Tang, J. Zhu, and Q. Mei, "A flexible generative framework for graph-based semi-supervised learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 3276–3285.

[42] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," in *Proc. Int. Conf. Learn. Representations*, 2017.

[43] M. Newman, *Networks*. Oxford, U.K.: Oxford Un. Press, 2018.

[44] J. Ni et al., "Co-regularized deep multi-network embedding," in *Proc. Int. Conf. World Wide Web*, 2018, pp. 469–478.

[45] M.T. Ribeiro, S. Singh, and C. Guestrin, ""Why should i trust you?" explaining the predictions of any classifier," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 1135–1144.

[46] K. Riesen and H. Bunke, "Iam graph database repository for graph based pattern recognition and machine learning, in *Proc. Joint IAPR Int. Workshops Statist. Techn. Pattern Recognit. Struct. Syntactic Pattern Recognit.*, Springer, 2008, pp. 287–297.

[47] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009.

[48] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2017, pp. 3145–3153.

[49] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 83–98, May 2013.

[50] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3319–3328.

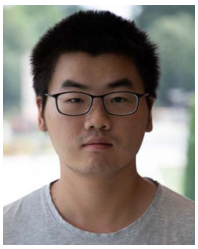
[51] J. J. Sutherland, L. A. O'brien, and D. F. Weaver, "Spline-fitting with a genetic algorithm: A method for developing classification structure-activity relationships," *J. Chem. Inf. Comput. Sci.*, vol. 43, no. 6, pp. 1906–1915, 2003.

[52] C. David et al., "The WU-minn human connectome project: An overview," *Neuroimage*, vol. 80, pp. 62–79, 2013.

[53] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, 'Graph attention networks," *Proc. Int. Conf. Learn. Representations*, 2018.

[54] C.-J. Wang, S. Chae, L. A. Bunimovich, and B. Z. Webb, "Uncovering hierarchical structure in social networks using isospectral reductions," 2017, *arXiv: 1801.03385*.
[55] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *Proc. Int. Conf. Learn. Representations*, 2019.
[56] C. Yang, A. Rangarajan, and S. Ranka, "Global model interpretation via recursive partitioning," in *Proc. IEEE 20th Int. Conf. High Perform. Comput. Commun.; IEEE 16th Int. Conf. Smart City; IEEE 4th Int. Conf. Data Sci. Syst.*, 2018, pp. 1563–1570.
[57] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 974–983.
[58] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "GNNExplainer: Generating explanations for graph neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 9240–9251.
[59] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "GraphRNN: Generating realistic graphs with deep auto-regressive models," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 5694–5703.
[60] H. Yuan, J.X. TangHu, and S. Ji, "XGNN: Towards model-level explanations of graph neural networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 430–438.
[61] H. Yuan, H.S. Y. Gui, and S. Ji, "Explainability in graph neural networks: A taxonomic survey," 2020, *arXiv:2012.15445*.
[62] H. Yuan, H. Yu, J. Wang, K. Li, and S. Ji, "On explainability of graph neural networks via subgraph explorations," 2021, *arXiv:2102.05152*.
[63] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 5165–5175.
[64] T. Zhao, D. Luo, X. Zhang, and S. Wang, "Faithful and consistent graph neural network explanations with rationale alignment," 2023, *arXiv:2301.02791*.
[65] T. Zhao, D. Luo, X. Zhang, and S. Wang, "Towards faithful and consistent explanations for graph neural networks," in *Proc. 16th ACM Int. Conf. Web Search Data Mining*, 2023, pp. 634–642.
[66] C. Zheng et al., "Robust graph representation learning via neural sparsification," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2020, pp. 11458–11468.
[67] Y. Zhu, W. Xu, J. Zhang, Q. Liu, S. Wu, and L. Wang, "Deep graph structure learning for robust representations: A survey," 2021, *arXiv:2103.03036*.

Dongsheng Luo (Member, IEEE) received the doctoral degree from the College of Information Science and Technology, Pennsylvania State University, supervised by Prof. Xiang Zhang. He is an assistant professor with Florida International University. His research interests include data mining and machine learning.

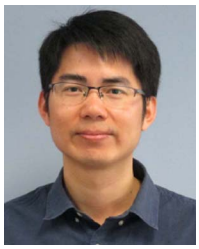


Tianxiang Zhao received the BS degree in computer science from the University of Science and Technology of China (USTC), Hefei, China, in 2017. He is currently working toward the PhD degree in IST with the Pennsylvania State University (PSU), State College, PA, under the supervision of Dr. Suhang Wang and Dr. Xiang Zhang, since 2019. His research interests are in graph neural networks, weak supervision tasks, and knowledge transfer.

Wei Cheng received the BS degree from the School of Software, Nanjing University, Nanjing, China, in 2006, the MSc degree from the School of Software, Tsinghua University, Beijing, China, in 2010, and the PhD degree from the Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, in 2015. He is currently a senior research staff member with the Data Science Department, NEC Laboratories America, Inc., Princeton, NJ, USA. His current research interests include data science, machine learning, web applications, and bioinformatics.

Dongkuan Xu received the PhD degree from Pennsylvania State University. He is currently a assistant professor with NC State, leading the NCSU Reliable and Efficient Computing Lab and working on deep learning, machine learning, and natural language processing.

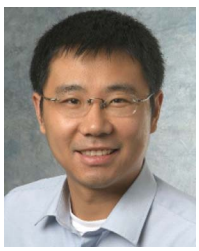
Feng Han received the PhD degree in bioengineering from the Pennsylvania State University, advised by Dr. Xiao Liu. His work aims to investigate the neural correlates of cognitive decline and amyloid deposition in Alzheimer's disease, as well as the brain function changes in normal aging and Parkinson's disease. His work also focused on brain-behavior association and brain arousal by analyzing multimodal imaging data.



Wenchao Yu received the PhD degree from the University of California, CA, USA, in 2018. He is currently a research staff member with the Data Science Department, NEC Laboratories America, Inc., Princeton, NJ, USA.

Xiao Liu received the graduate and master's degrees from Beijing University, China, and the PhD degree in biomedical engineering from the University of Minnesota. He received his PhD training from the Center for Magnetic Resonance Research (CMRR). He is the principal investigator with the Multimodal and Computational Neuroimaging Laboratory (MCNL). He joined the Nuclear Magnetic Resonance (NMR) Center, National Institutes of Health (NIH) as a postdoctoral fellow. He joined the Department of Biomedical Engineering, Penn State University, in 2016 as an Assistant Professor.

Haifeng Chen is heading the Data Science and Systems Security Department, NEC Laboratories America, Princeton, New Jersey. He and his team members are working on the various aspects of Big Data research including data management and mining, artificial intelligence, software and system security, smart services and platforms. He has served on the program committee for a number of top conferences, such as SigKDD, AAAI, IEEE BigData, and has been on the panel of NSF programs.



Xiang Zhang received the PhD degree from the University of North Carolina, Chapel Hill, in 2011. He is an associate professor with the College of Information Sciences and Technology, Pennsylvania State University. His research interests include data mining, bioinformatics, and databases.