GNNLens: A Visual Analytics Approach for Prediction Error Diagnosis of Graph Neural Networks

Zhihua Jin, Yong Wang, Qianwen Wang, Yao Ming, Tengfei Ma, and Huamin Qu, *Member, IEEE*

Abstract—Graph Neural Networks (GNNs) aim to extend deep learning techniques to graph data and have achieved significant progress in graph analysis tasks (e.g., node classification) in recent years. However, similar to other deep neural networks like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), GNNs behave like a black box with their details hidden from model developers and users. It is therefore difficult to diagnose possible errors of GNNs. Despite many visual analytics studies being done on CNNs and RNNs, little research has addressed the challenges for GNNs. This paper fills the research gap with an interactive visual analysis tool, *GNNLens*, to assist model developers and users in understanding and analyzing GNNs. Specifically, Parallel Sets View and Projection View enable users to quickly identify and validate error patterns in the set of wrong predictions; Graph View and Feature Matrix View offer a detailed analysis of individual nodes to assist users in forming hypotheses about the error patterns. Since GNNs jointly model the graph structure and the node features, we reveal the relative influences of the two types of information by comparing the predictions of three models: GNN, Multi-Layer Perceptron (MLP), and GNN Without Using Features (GNNWUF). Two case studies and interviews with domain experts demonstrate the effectiveness of *GNNLens* in facilitating the understanding of GNN models and their errors.

Index Terms—Graph Neural Networks, Error Diagnosis, Visualization.

✦

1 INTRODUCTION

GRAPHS are pervasive in various applications, such as citation networks, social media, and biology. Analyzing graph data helps us understand the hidden patterns in graphs and benefits many graph-related tasks, including node classification, link prediction, and graph classification. For example, an effective analysis of a paper citation graph can facilitate the prediction of a new paper [1], [2]. An exploration of social networks can benefit the creation of an adaptive friend recommendation system in social media [3]. By modeling molecules as graphs, where atoms and chemical bonds are treated as nodes and edges respectively, we can build machine learning techniques to predict the chemical properties (*e.g.*, solubility) of chemical compounds [4].

In recent years, graph analytics has embraced a new breakthrough—Graph Neural Networks (GNNs). A fast growing number of GNN models have been proposed to solve graph-based tasks. For example, Graph Convolutional Network (GCN) [1] adapts the convolutional operation from natural images to graphs and conducts semi-supervised learning to perform node classification on them. Graph Attention Network (GAT) [5] further integrates the attention

mechanism, which is widely used in Natural Language Processing (NLP), into the GNN model architecture and dynamically assigns weights to different neighbors to enhance the model performance. The advances of GNNs bring new opportunities to the analysis of graph data and have become increasingly popular in recent years. However, similar to other deep neural networks, GNN models also suffer from the difficulty of interpreting their working mechanisms. When developing or using GNNs, developers and users often need to evaluate the model performance and explore the causes of model errors and failures, which, unfortunately, is often hard to achieve. Therefore, how to enable convenient error diagnosis of GNN models has become a challenging but significantly important task.

Visualization has been applied to helping model developers devise new deep learning techniques, and debug and compare different types of deep neural networks [6]. For example, various visualization techniques have been proposed to facilitate the development of a variety of deep learning models, such as CNN [7], RNN [8], GAN [9], and DQN [10]. These visualizations have achieved great success in understanding and analyzing those deep learning models. However, it is very challenging to directly apply them to GNNs, since most of those techniques are exclusively designed for Euclidean data like images and text, while GNNs mainly work on non-Euclidean data such as graphs.

Another challenge for the error diagnosis of GNNs comes from the fact that GNNs often involve both the complex topological structures and high dimensional features of graphs, as well as the interplay between them. To effectively analyze GNNs, it is crucial to properly link the topological data, high dimensional features, and prediction results with

---

- *Z. Jin and H. Qu are with Hong Kong University of Science and Technology, Hong Kong, China.*
  E-mail: zjinak, huamin@cse.ust.hk.
- *Y. Wang is with Singapore Management University.*
  E-mail: yongwang@smu.edu.sg.
- *Q. Wang is with Harvard University.*
  E-mail: qianwen_wang@hms.harvard.edu.
- *Y. Ming is with Bloomberg LP.*
  E-mail: yming7@bloomberg.net.
- *T. Ma is with IBM T. J. Watson Research Center.*
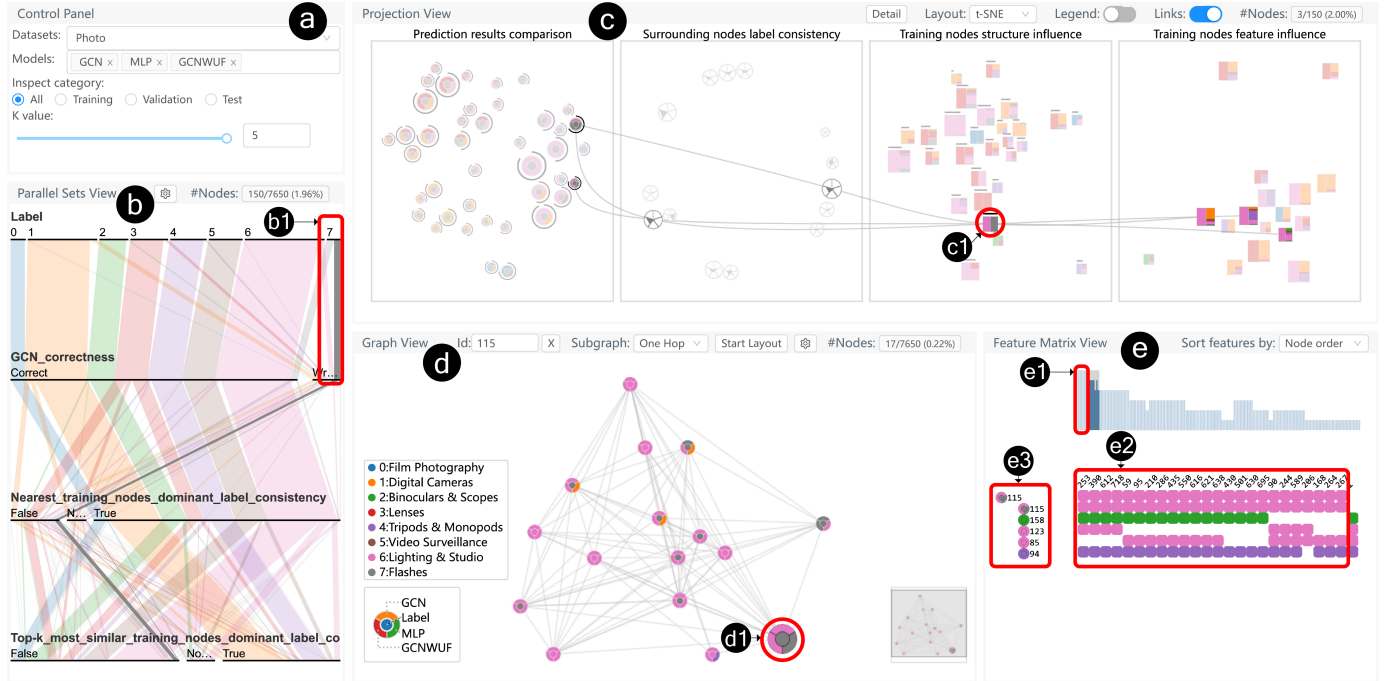 *E-mail: tengfei.ma1@ibm.com.*

Fig. 1. *GNNLens* is a visual analytics tool that helps model developers and users understand and diagnose GNNs. *GNNLens* consists of Control Panel, Parallel Sets View, Projection View, Graph View, and Feature Matrix View. (a) The Control Panel enables users to interactively configure the basic parameters (e.g., the dataset). (b) The Parallel Sets View displays the overall distribution of node properties. (c) The Projection View further visualizes four groups of key metrics of nodes that have a direct influence on GNN model performances. Interactions such as lasso-selection and linked highlighting are also supported to facilitate convenient exploration between the GNN prediction results on individual nodes and node properties. (d) The Graph View provides users with the interactive exploration of the detailed topology structures around the node of user interest, facilitating the analysis of the influence of topology structures on the GNN model performance. (e) The Feature Matrix View visualizes the feature distribution of nodes.

a comprehensive workflow. Preliminary studies [11], [12], [13] have proposed techniques to explain GNN model prediction results. Most of them focus on instance analysis, *i.e.*, explaining a prediction for single nodes. However, there still lacks the ability and research at a higher level, *i.e.*, analyzing and understanding the common causes of the classification errors of groups of nodes. Their methods make it difficult to conveniently explore the general error patterns in the prediction results of a GNN model, as well as further gain insights for model improvement. In summary, it still remains unclear how to develop new visualization techniques to facilitate the effective error diagnosis of GNNs.

In this paper, we propose a novel error-pattern-driven visual analytics system, *GNNLens*[1], to provide model developers and users with deep insights into model performance and its dependency on data characteristics. Instead of analyzing the GNN prediction results of single instances, we investigate the patterns in the prediction results shared by a group of instances to obtain generalizable insights into the model architecture. We worked closely with two GNN experts for four months to derive the design requirements of *GNNLens*. *GNNLens* comprises five views: Control Panel, Parallel Sets View, Projection View, Graph View, and Feature Matrix View, as shown in Fig. 1. The Parallel Sets View enables users to see the distribution of node-level metrics (Fig. 1(b)). The Projection View presents a set of 2D projections of the selected nodes according to metrics summarized

from different perspectives, enabling users to extract potential clusters of nodes (Fig. 1(c)). Node glyphs are proposed to help users conveniently learn about multiple metrics of nodes and extract general error patterns in the Projection View. The Graph View shows the node information and topological structure of the whole graph. Users can also zoom in and focus on the topological structure of a specific node (Fig. 1(d)). The Feature Matrix View shows the detailed features of selected individual nodes (Fig. 1(e)). All the five views of *GNNLens* are linked together to help users analyze GNN models simultaneously from multiple angles and facilitate the exploration of error patterns of the GNN models. We conducted two case studies and expert interviews to demonstrate the effectiveness and usability of *GNNLens* in helping model developers understand and diagnose GNNs.

The contributions of our work are listed as follows:

- A visual analytics system to assist model developers and users in understanding and diagnosing GNNs.
- Case studies on analyzing error patterns in GNN prediction results and interviews with domain experts to demonstrate the effectiveness and usability of the proposed system.

The remainder of this paper is organized as follows. Section 2 discusses the related work of this paper, including GNNs, visual analytics in deep learning, and GNN explainability. Section 3 provides a brief introduction to the basic concepts of GNNs, such as typical architectures GCN and GAT. By working closely with domain experts, we

summarize the design requirements of understanding and diagnosing GNN models in Section 4 and further introduce the technical details of *GNNLens* in Section 5. We evaluate our approach through case studies and expert interviews in Section 6 and discuss the possible limitations and future work of our approach in Section 7. Section 8 concludes the paper with a brief summary of the proposed method.

2 RELATED WORK

The related work of this paper can be categorized into three groups: graph neural networks, visual analytics in deep learning, and graph neural networks explainability.

2.1 Graph Neural Networks

GNNs have been developed to analyze graph data by extending CNNs or RNNs to the graph domain [14] in the past few years. These neural networks have gained promising prediction results for analyzing graphs.

For the GNNs derived from CNN, they can be categorized into spectral approaches and spatial approaches [14]. Spectral approaches define convolution on the spectral representation of graphs [1], [15], [16]. The work done by Bruna et al. [15] is the first attempt to generalize the convolution concept from natural images to the graph domain. Defferrard et al. [16] approximated the spectral convolution as Chebyshev polynomials of the diagonal matrix of eigenvalues, resulting in further low computational cost. Kipf and Welling [1] further simplified the Chebyshev polynomials by using the first order of polynomials and renormalization tricks, known as GCNs, which have inspired many follow-up studies. Spatial approaches directly define convolution on spatially close neighbors [2], [17], [18], [19], [20], [21], [22]. Hamilton et al. [2] proposed GraphSAGE, which uses sampling methods and aggregators defined over the neighborhood to reduce dependence on processing whole graphs. Their approach greatly accelerates the GNN used in large scale graphs. Another direction is to extend RNN to the graph domain. Prior studies have attempted to utilize the gate function in GNNs to improve its ability to propagate information across graph structure [23], [24], [25], [26], [27].

Researchers have also made significant progress in analyzing GNN models. For example, Li et al. [28] showed that the graph convolution of a GCN is merely a Laplacian smoothing operation but when the number of layers increases, the risk of over smoothing will be increased. Also, they showed that when few training labels are given to train GCN models, co-training methods and self-training methods will improve the performance of GCN models. Xu et al. [29] provided a theoretical framework to analyze expressive power for GNNs and proved that their proposed model is as expressive as the Weisfeiler Lehman graph isomorphism test. Different from these studies, this study is aimed at extracting general error patterns of GNN models and further helps model developers understand and diagnose the models.

2.2 Visual Analytics in Deep Learning

Nowadays, there is a growing trend to use visualizations to understand, compare, and diagnose deep neural networks [6]. Prior studies on using visual analytics to enhance the interpretability of deep neural networks can generally be categorized into two types: model-agnostic visualizations and model-specific visualizations. For model-agnostic visualizations, prior studies mainly focus on visualizing the model input and output to provide insights into the correlation between them [30], [31] or using surrogate models to explain the deep neural networks [32], [33]. However, these model-agnostic visualizations avoid showing the hidden states of the deep neural networks and fail to reveal the inner working mechanism of different models.

To support a dive into the deep learning models, researchers have also proposed a series of model-specific visualizations for explaining deep learning models. Previous model-specific visualizations have covered a wide range of deep learning models, including CNN, RNN, and GAN. A variety of visualization techniques and interactions have been designed based on the data type, the model structures, and the working mechanism of different deep learning models. Since CNN and RNN are the most widely-used deep learning models [34], [35], a majority of model-specific visual analytics are proposed for both types of models. For example, CNNs are usually modeled using the directed acyclic graph visualization, and the output of each layer is usually displayed using matrix-based visualizations [7], [36], [37]. To open the black box of RNNs, clustering methods and correlation visualizations have been proposed to uncover the dynamic hidden states and learned patterns in RNNs [8], [38], [39]. Recently, visual analytics methods tailored for generative models [9], [40], [41] and reinforcement learning models [10] have also been proposed.

Despite much work having been done by using visualization approaches to improve the explainability of deep learning models, little research has been conducted to enhance the explainability of GNNs through visualizations. To fill this research gap, this paper proposes a visualization tool to assist in the understanding and diagnosis of GNNs.

2.3 Graph Neural Network Explainability

According to our survey, only a few studies have attempted to explain GNN models. For instance, Baldassarre et al. [11] explored the possibilities of adapting explanation techniques from CNNs to GNNs. They empirically evaluate three widely-used CNN explanation methods, *i.e.*, Sensitivity Analysis (SA), Guided Back Propagation (GBP), and Layer-wise Relevance Propagation (LRP) when explaining GNN decisions. They found that explanations produced by SA or GBP tend to be inconsistent with the human interpretation, while LRP produces more natural explanation results. Meanwhile, Ying et al. [12] proposed GNNExplainer, which uses a subgraph to explain the GNN model prediction. Given a trained GNN model, they formulate an optimization task to maximize the mutual information between the trained model prediction and distribution of possible graph structures. They regard the subgraph as the explanation results. Li et al. [13] further extended GNNExplainer which is designed for an undirected unweighted graph to suit a directed weighted graph.

Previous studies have mainly focused on providing an instance-based explanation of the prediction results of GNN models. Different from them, our work mainly focuses on

analyzing error patterns made by GNN models, including group-level and instance-level errors.
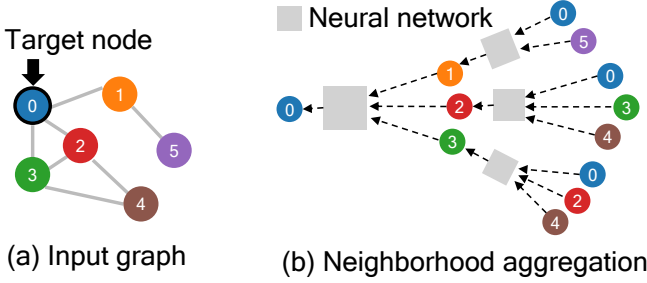

Fig. 2. Given an input graph (a), GNN predicts the label of the target node (*e.g.*, the blue node) by aggregating the information from neighboring nodes (b).

3 BACKGROUND

GNNs are deep neural networks that directly operate on graphs (*i.e.*, networks). A graph can be represented as $G = (V, E)$, where V denotes the vertex set and E denotes the edge set. $X \in \mathbb{R}^{N \times d}$ is the feature matrix of the graph, where N denotes the number of nodes in the vertex set and d is the dimension of each node feature. The labels of the nodes in the graph are often denoted as Y. In this paper, we do not consider the features in edges.

We adopt similar notations introduced in [42] to illustrate the concept of GNNs. GNNs can be generally expressed in a neighborhood aggregation or message passing scheme [43], as shown in Fig. 2. A general message passing function for GNN is shown as below:

$$\mathbf{x}_i^{(k)} = \mathcal{C}^{(k)} \left(\mathbf{x}_i^{(k-1)}, \mathcal{A}_{j \in \mathcal{N}(i)} \phi^{(k)} \left(\mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)} \right) \right) \quad (1)$$

where $x_i^{(k-1)}$ denotes node features of Node i in Layer $k - 1$. $\mathcal{N}(i)$ denotes the neighborhood of Node i. $\mathcal{A}$ denotes a differentiable permutation-invariant function, e.g., sum. $\mathcal{C}$ and ϕ denote differentiable functions such as Multi-Layer Perceptrons (MLPs).

GCNs and GATs are two popular GNNs. According to the study by Kipf et al. [1], the message passing function of GCN can be defined as follows:

$$\mathbf{x}_i^{(k)} = \sigma \left(\sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{1}{\sqrt{\deg(i)} \sqrt{\deg(j)}} \left(\mathbf{W} \mathbf{x}_j^{(k-1)} \right) \right) \quad (2)$$

where the features of the neighbors of Node i are first transformed by a weight matrix $\mathbf{W}$. Then they are normalized by their degree and finally summed up. Finally, a non-linear activation function σ is applied to process it.

GAT is first proposed by Veličković et al. [5] and its message passing function is defined as follows:

$$\mathbf{x}_i^{(k)} = \sigma \left(\sum_{j \in \mathcal{N}(i) \cup \{i\}} \alpha_{ij} \mathbf{W} \mathbf{x}_i^{(k-1)} \right) \quad (3)$$

where $\mathbf{W}$ and σ are similarly defined as above. Different from GCN, GAT assigns different weights (attention coefficients) to each neighbor. The attention coefficients $\alpha_{i,j}$ are computed as:

$$\alpha_{i,j} = \frac{\exp \left(\text{LeakyReLU} \left(\mathbf{a}^\top \left[\mathbf{W} \mathbf{x}_i \| \mathbf{W} \mathbf{x}_j \right] \right) \right)}{\sum_{k \in \mathcal{N}(i) \cup \{i\}} \exp \left(\text{LeakyReLU} \left(\mathbf{a}^\top \left[\mathbf{W} \mathbf{x}_i \| \mathbf{W} \mathbf{x}_k \right] \right) \right)} \quad (4)$$

where $\mathbf{a}$ is a weight vector and LeakyReLU is an activation function which is defined as $\text{LeakyReLU}(x) = \max(0, x) + \text{negative\_slope} * \min(0, x)$.

The GNN models are mainly applied to the tasks of node classification and link prediction in individual graphs. In this paper, we take node classification as an example to illustrate how our approach can improve the interpretation of GNN models and facilitate model diagnosis. Such kinds of node classification tasks are often done in a semi-supervised way. Given a set of labeled nodes (*i.e.*, training nodes) in a graph, a GNN model will be trained to predict the labels of the rest of the nodes in the graph.

4 DESIGN REQUIREMENT ANALYSIS

We work closely with two GNN experts, who are also co-authors of this work, to collect their feedback on the GNN interpretation issues they are facing and their current practices of understanding and diagnosing GNN models. One expert (E1) is a senior researcher who specializes in developing new kinds of GNNs. The other expert (E2) is a deep learning developer with strong experience in applying GNNs to modeling and analyzing the topology data from different application domains such as online education and visualization. Also, the development of *GNNLens* was conducted in an iterated way. After we finished each version of the system, we asked experts to use the pilot system, comment on the limitations, and suggest possible improvements. By combining the original requirements proposed by the experts and their subsequent comments on the limitations of the systems, we complied a list of major design requirements proposed by the domain experts, which can be summarized as follows:

R1: Provide an overview of GNN results. All experts commented that an overview of the GNN performance is crucial for the GNN analysis. To gain an overview of the dataset and classification results, the system needs to summarize various types of information, such as degree distribution and ground truth label distribution. This information, covering various aspects of a GNN model, needs to be organized and presented in a clear manner. Meanwhile, the correlation among this information should be presented to help users develop initial hypotheses about any possible error patterns in GNN results, *i.e.*, a set of wrong predictions that share similar characteristics.

R2: Identify error patterns. After developing initial hypotheses about the error patterns, users need more detailed information to verify them. Specifically, users need to examine the characteristics shared by a set of wrong predictions and verify whether error patterns formed by these characteristics make sense in analyzing GNNs based on their domain knowledge. During the interview, experts agreed that they usually use several characteristics to group

the wrong predictions and identify error patterns. For example, one expert stated that *"misclassified nodes usually have a relatively large shortest path distance to the labeled nodes."* Therefore, the system should support users in examining these characteristics and identifying error patterns.

R3: Analyze the cause of error patterns. After identifying error patterns, finding the causes of these errors is important for users to understand, diagnose, and improve the GNNs. More detailed information is needed to understand the possible causes of error patterns. Specifically, users need to inspect the graph structures and node features to determine the causes of error patterns. According to the feedback from expert interviews, there are two main sources of wrong GNN predictions: noise in the training data and inaccurate feature aggregation in GNNs. To predict the label of a node, GNN aggregates the node's own feature with the features of the neighboring nodes at each layer. Noise in the training data, e.g., the same nodes but different labels, can confuse the GNN and lead to wrong predictions. Inaccurate feature aggregation at any layer will also influence the GNN prediction of the node.

5 GNNLENS

This section describes the details of the proposed approach, *GNNLens*. We first provide a system overview. Next, detailed information on proxy models and metrics is provided. Finally, we introduce the detailed visualization design of each view in *GNNLens*.

5.1 System Overview

GNNLens consists of three major modules: storage, data processing, and visualization. The storage module stores and manages graph data and models. The data processing module implements the necessary procedures for analyzing the graph and model predictions, especially for calculating various kinds of metrics. The processed data is then passed to the visualization module, which supports the interactive visual analysis of the GNNs. The storage and data processing modules are developed using Python and Deep Graph Library (DGL) [44] and integrated into a back-end web server that is built with Flask. The GNN models are implemented with PyTorch. We implement the visualization module as a front-end application using React, Typescript, and D3.

5.2 Proxy Models Training and Metrics Definition

Inspired by the fact that GNN prediction results are influenced by both graph structure and node features [14], we define two proxy models to analyze the influence of the graph structure and node features on GNN prediction results. Through expert interviews, experts are concerned about whether the graph structure or node features have a greater impact on GNN prediction, and then determine which components will have more impact. Hence, similar to the ablation study when evaluating GNN models [29], we define two proxy models such as GNN Without Using Features (GNNWUF) and Multi-Layer Perceptron (MLP). The two models are chosen, since the two proxy models have the same model architectures as the GNN but are

trained using different input data. GNNWUF is trained only using the graph structure while MLP is trained only using the node features. When training GNNWUF, we use one hot encoding as the node feature for each node, meaning GNNWUF considers only the graph structures. When GNN considers only the features of the node itself, then it can degenerate into an MLP model. Hence, MLP is chosen as the other GNN proxy model that only considers the node features and is used to evaluate the influence of node structures. We train both proxy models with the same settings as the training of GNN.

To further help users understand the impact of the graph structure and node features, we also provide a number of metrics, including graph structure based metrics that take into account the graph structure but ignore the node features, and node feature based metrics that take the node features into account but ignore the graph structure. Those metrics are derived from expert interviews. Details are presented in the following paragraphs.
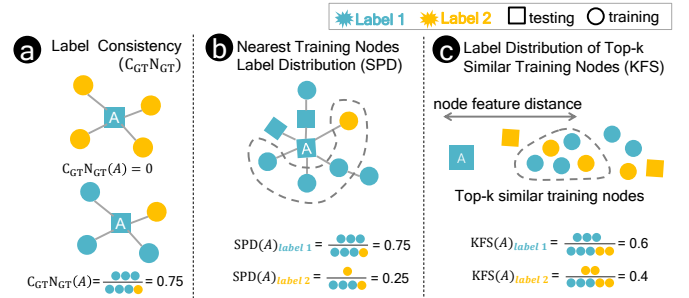

Fig. 3. Examples illustrating the computation of metrics, including Label Consistency (a), Nearest Training Nodes Label Distribution (b), Label Distribution of Top-k Similar Training Nodes (c).

5.2.1 Graph Structure based Metrics

- *Node degree*. GNNs mainly operate over the neighborhood of each node and the number of neighbors can affect the final performance of a GNN model [45]. Therefore, the *node degree* is considered in this study.

- *Center-neighbor consistency rate*. The *center-neighbor consistency rate* depicts how consistent the *labels* of the *current node* and its surrounding neighbors are [46]. It can be divided into four major categories by considering both the *ground truth labels* and their predictions: (1) **Label consistency** shows the percentage of neighbors that have the same *ground truth label* as the *current node* (Fig. 3(a)); (2) **Label-Prediction consistency** describes the percentage of neighbors whose *GNN prediction labels* are the same as the *current node's ground truth label*; (3) **Prediction-Label consistency** delineates the percentage of neighbors whose *ground truth labels* are the same as the *current node's GNN prediction label*; and (4) **Prediction consistency** refers to the percentage of neighbors which have the same *GNN prediction label* as the *current node*. These values can indirectly reflect how many neighbors satisfy the constraints. If the *node degree* is zero, then the *consistency rate* is set to zero. These metrics help users check whether the one-hop neighborhood exerts

influence on the *GNN prediction result* on the node of user interest.

- *Shortest path distance to training nodes*. We use the breadth-first search (BFS) algorithm to calculate the *shortest path distance from the current node to the training nodes*. The algorithm will first start traversing the *current node* and then the neighbors of the visited nodes. When it first detects a node in the training set, the algorithm will regard the distance from that node to the *current node* as the *shortest path distance from the current node to the training nodes*. The distribution of the training nodes, also called labeled nodes, can have a significant influence on *GNN prediction* [47].

- *Nearest training nodes label distribution*. To investigate the influence of training nodes distribution on model training, we calculate the *nearest training nodes label distribution*. To calculate the *label distribution of the nearest training node(s)*, we first find the closest training nodes to the *current node* in terms of shortest path distance. Then we count the frequency of the labels of these training nodes and normalize them into $[0, 1]$. The normalized frequencies are considered to be the *nearest training nodes label distribution* (Fig. 3(b)). Analyzing these metrics helps to investigate the influence of training nodes distribution on model training [47].

- *Nearest training nodes dominant label consistency.* In order to help users quickly capture the dominant information of *nearest training nodes label distribution* and further diagnose the causes of errors in *GNN prediction results*, we define the *nearest label* as the label that most frequently occurs in the training nodes closest to a specific node in terms of topological distance. Then, we further consider whether the *nearest label* is consistent with the *ground truth label* of this specific node. If yes, we set the *nearest training nodes dominant label consistency* for this node to *True*; otherwise, it is set to *False*. Sometimes, there may be multiple *nearest labels*. Then we directly set the *nearest training nodes dominant label consistency* to *Not Sure*. Such a metric is derived from *nearest training nodes label distribution* and the *ground truth label* of the *current node*. If it is true, the current node can get information from the structure and the training nodes and it has a high chance of being correctly classified. Otherwise, it has high probability to be misclassified [47].

5.2.2 Node Features based Metrics

- *The label distribution of the top-k training nodes with the most similar features.* The feature similarity between two nodes is defined as the cosine distance between the feature vectors of two nodes [48]. We first find the *top-k training nodes with the most similar features* to the node of user interest. Then we count the frequency of labels of those training nodes and normalize them into $[0, 1]$. They are defined as the *label distribution of the top-k training nodes with the most similar features* (Fig. 3(c)). With this metric, we can analyze the influence of node features on *GNN predictions*. We set the default value of k to 5 in our current implementations.

- *Top-k most similar training nodes dominant label consistency.* Similar to the definition of the previous metric, we can also calculate the *top-k most similar training nodes*

dominant label consistency in the same way. The major difference is that this metric reflects the influence of training node features on the model prediction results on the current node [48].

5.3 Visualization

As shown in Fig. 1, *GNNLens* visualization module consists of a Control Panel (a), a Parallel Sets View (b), a Projection View (c), a Graph View (d), and a Feature Matrix View (e). The Control Panel allows users to choose a graph dataset, select inspected models (Appendix A), and explore different subsets of the dataset (*e.g.*, *all, training, validation,* and *testing*). It also allows users to choose the k value to calculate node features based metrics. The Parallel Sets View (Fig. 1(b)) visualizes the distribution of node-level metrics, which are defined in Section 5.2. Users can select a subset of metrics to inspect their distribution and correlation. Then users can select a subgroup of nodes and check them in the Projection View (Fig. 1(c)). The Projection View presents a set of 2D projections of the selected nodes according to metrics summarized from different perspectives. Users can lasso-select a cluster of nodes in one plane to see their locations on the whole graph in the Graph View (Fig. 1(d)) and their feature distributions in the Feature Matrix View (Fig. 1(e)).

5.3.1 Parallel Sets View

In order to provide a high-level summary (**R1**) and help users understand the datasets and identify error patterns of *GNN models prediction* (**R2**), we design the Parallel Sets View to visualize node-level metrics using Parallel Sets [49]. Previous work [37], [50], [51], [52], [53] explored the selection of a subset of sample properties to study machine learning models. Inspired by previous research, we use this strategy to investigate error patterns in GNN models. We propose to use Parallel Sets to investigate error patterns in GNN prediction results, following the previous work [54], [55]. Users can select what metrics are to be displayed in the Parallel Sets through Parallel Sets Settings Modal. In general, displaying fewer than five axes in Parallel Sets is a good practice to reduce visual clutter and make efficient use of functions in Parallel Sets. Due to the constraint that the Parallel Sets are used to display the categorical variables, we need to convert the continuous metrics to categorical variables by grouping a range of values into one category. Then we can also show them in the Parallel Sets View.

As shown in Fig. 1(b), each axis of the Parallel Sets shows a categorical variable. The axis is partitioned into multiple segments representing different categories of the variable. The width of each segment represents the number of nodes falling into that category. We can directly see the distribution of the categories on the axis. Between two consecutive axes, multiple ribbons are shown to connect the two axes, each simultaneously representing the nodes that satisfy the conditions specified by the two axes.

Users can easily select a subset of nodes in the dataset and further investigate their node metrics and the *GNN model prediction results*. When users click on a segment, the corresponding category of that axis will be selected. Also, when users click on the ribbon in the Parallel Sets, the

corresponding set of nodes will be selected. Besides, the axes in the Parallel Sets can be easily reordered by users through drag-and-drop. By filtering the nodes according to node-level metrics such as *correctness* and *label*, users can easily select a node subset of their interest for further analysis.

A common alternative for visualizing multivariate data is the Parallel Coordinates Plot (PCP) [56]. Each data point is visualized as a single line across different attributes. However, when it comes to categorical data, it is challenging to identify the proportions of data that fall into specific categories. Compared with PCP, Parallel Sets intuitively show the distribution of the categories in each axis and the correlation between multiple axes. Thus, Parallel Sets are chosen to display the overall distribution of node attributes.

5.3.2 Projection View

With the overview of the dataset and GNN models provided by the Parallel Set View, we further design the Projection View to give users more insights into the subset of nodes selected in the Parallel Sets View (**R2, R3**). We group a subset of node-level metrics, display them in glyphs, and further project them to the 2D plane. The Projection View allows users to investigate the similarity of nodes regarding different perspectives. It can be helpful for investigating whether the nodes with similar node metrics share similar error patterns.

In the Projection View, we provide a set of linked projection planes of the nodes that use different features. Different from similar designs in EmbeddingVis [57], we design different node glyphs to display different combinations of node-level metrics. To project those node glyphs in 2D plane, users can choose to use the t-SNE [58] or UMAP [59] projection as the basic layout algorithm. Moreover, the force-directed collision-avoidance method is integrated into the basic layout algorithm to prevent the overlapping of node glyphs. When users lasso-select a set of nodes in a projection plane, the links between the same nodes in different planes will be shown to help users identify the nodes and other aspects of those nodes' properties, as shown in Fig. 4. After users hover on the node glyphs, the legend and detailed information of those node glyphs will be displayed. However, due to the limited screen space, it cannot display hundreds let alone thousands of node glyphs. Therefore, we apply a hierarchical clustering algorithm with complete linkage to cluster these nodes based on the corresponding distance function [60]. Two clusters will be merged into one cluster when the distance of two clusters is less than or equal to a threshold, which is empirically set to 0.5. In terms of categorical metrics such as *ground truth label* and *prediction results*, the majority of a categorical metric in one cluster is regarded as the aggregate result of that metric. For continuous metrics such as *center-neighbor consistency rate* and *degree*, we take their average as the aggregated results. Cluster-level node glyphs are designed based on the aggregate results of the node-level metrics for individual nodes in the clusters. In order to help users further inspect individual nodes, after users select a subset of cluster-level node glyphs, users can switch to "Detail" mode, then the Projection View will display individual node glyphs for nodes in such a cluster. This design greatly enhances the scalability of the Projection View.

In our implementation, we categorize the metrics into four groups and provide four projections for each group of node metrics. The four projection planes are *prediction results comparison*, *surrounding nodes label consistency*, *training nodes structure influence*, and *training nodes feature influence*, respectively. Different glyph designs are also proposed for the nodes. We will introduce them one by one in the following paragraphs.

A. Prediction results comparison. This plane aims to help users compare different *models prediction results* and reveal the relative influence of graph structures and features for each node or cluster. The metrics used in this plane include the *ground truth label GT*, the *prediction label of three models*, *i.e.*, GNN, GNNWUF, MLP $P = [P_1, P_2, P_3]$, and the *Confidence* of GNN prediction $CONF$. As shown in Fig. 4(b), *three model prediction results* can be found in the pie chart. The inner circle encodes the *ground truth label*. The outer circular ring encodes the *confidence*. The radius of the whole node glyph encodes the size of the clusters. Through such a node glyph, users can easily compare the *ground truth label* and *model prediction results* and understand how confidently GNN models make predictions. For example, if the glyph shows consistent color across the inner circle and the three sectors of the pie chart, it means that the model can use either the node features or the graph structure information to correctly predict the node labels. Through the projection, the nodes with similar metrics will be in close proximity. Users can see if there are clusters of nodes with the same *ground truth labels* and *predictions*, which helps GNN model developers and users further analyze what causes the model to make such predictions. For projection and clustering, the distance between Node a and Node b in this plane is defined as below:

$$D_1^2(a, b) = \mathbb{I}\{GT_a \neq GT_b\} + \sum_{j=1}^{3} \mathbb{I}\{P_{aj} \neq P_{bj}\} \\ + (CONF_a - CONF_b)^2 \tag{5}$$

where $\mathbb{I}\{*\}$ is an indicator function, which will be 1 if the expression is true and 0 otherwise. Such a distance function guarantees that the value of each term is between 0 and 1.

B. Surrounding nodes label consistency. To help users explore the *label consistency* between a node and its neighboring nodes, we show the *ground truth label GT*, the *degree DEG*, the *center-neighbor consistent rate* $CN = [C_{GT}N_{GT}, C_{GT}N_{PT}, C_{PT}N_{GT}, C_{PT}N_{PT}] \in [0, 1]^4$ in this plane, where $C_{GT}N_{GT}$ represents *Label consistency*, $C_{GT}N_{PT}$ represents *Label - Prediction consistency*, $C_{PT}N_{GT}$ represents *Prediction - Label consistency*, and $C_{PT}N_{PT}$ represents *Prediction consistency*. The node glyph (Fig. 4(c)) is designed to show this group of metrics. The design is inherent from the Radar Chart as it can display continuous variables. The color of polygon encodes the *ground truth label*. When the five axes of the Radar Chart reach a large value, it means that the *center node* has many surrounding nodes, and the *ground truth labels* and *GNN predictions results* of that node are very consistent with the surrounding nodes. It suggests that the GNN model can effectively use the information of surrounding nodes for correct classification. The radius of the whole node glyph encodes the size of the clusters.
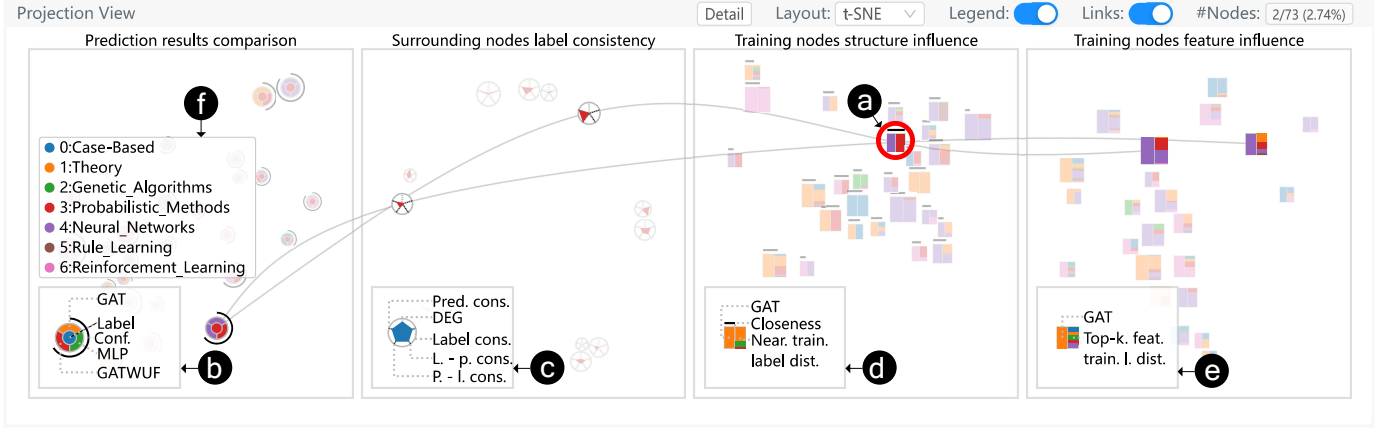
Fig. 4. Projection View enables users to analyze the node-level metrics of the subset of nodes from different perspectives: (a) The links connecting different kinds of information of the same nodes shown in different planes will be displayed when users lasso a group of nodes in one plane; (b-e) Node glyphs design in planes of the Projection View; (f) Color indicates the corresponding label.

Clusters may appear and users can easily spot them, since there will be a certain shape among those node glyphs. For the projection and clustering, the distance between Node a and Node b is defined as:

$$D_2^2(a,b) = (Norm(DEG_a) - Norm(DEG_b))^2$$
$$+\mathbb{I}\{GT_a \neq GT_b\} + \sum_{i=1}^{4}(CN_{ai} - CN_{bi})^2, \quad (6)$$

where $Norm(d) \in [0,1]$ is the normalized degree.

C. Training nodes structure influence. To help users capture the structure influence of training nodes on *GNN model prediction*, the metrics visualized in this plane include *GNN prediction label* P_1, *shortest path distance to training nodes* DIS, and *normalized nearest training nodes label distribution* $SPD \in [0,1]^C$. Here C is the number of classes. In order to encode the DIS in the node glyph and highlight the difference between smaller values, *i.e.*, $DIS \in [0,5)$, we define $Closeness = max(0, 1 - DIS * 0.2)$. It depicts the *closeness of the nearest training nodes to the current node*. The node glyph (Fig. 4(d)) is designed to show this group of metrics. The length of the line on the top of the rectangle encodes the *closeness*. The rectangle on the right-hand side of the glyph shows the *distribution of ground truth labels of training nodes with the shortest path distance* to that node. The width and height of the whole node glyph encode the size of the clusters. The left-hand side rectangle encodes the *GNN prediction label*. This helps users analyze the correlation between those variables. When the color of the left-hand side rectangle is consistent with that of the right-hand side rectangle, it means that there is a high correlation between P_1 and the dominant component of SPD, indicating that the closest training nodes have a strong influence on *GNN's prediction result of the current node*. For the projection and clustering, the distance between Node a and Node b is defined as:

$$D_3^2(a,b) = \mathbb{I}\{P_{a1} \neq P_{b1}\} + + \sum_{i=1}^{C}(SPD_{ai} - SPD_{bi})^2 \quad (7)$$
$$+(Closeness_a - Closeness_b)^2.$$

D. Training nodes feature influence. We further use another plane to help users capture the feature influence of training nodes. The metrics we used in this plane include *GNN prediction label* P_1 and $KFS \in [0,1]^C$, the *label distribution of the top-k training nodes with the most similar features*. The node glyph (Fig. 4(e)) shares a similar visual design with Fig. 4(d). The difference is that the right-hand side rectangle encoded the *top-k feature similarity training nodes ground truth label distribution* and the node glyphs do not have a line at the top. It enables users to analyze *GNN prediction results* from the perspective of features. If users find that the color of the GNN prediction result occupies a large portion of the right-hand side rectangle, it indicates that the *top-k feature similarity training nodes* have a strong influence on *GNN's prediction of the current node*. The clusters on the projection plane indicate nodes that have been similarly affected by the features. Combined with the Feature Matrix View, we can determine which feature may play a better or worse role in the GNN predictions. We use a similar distance function defined in the plane of *training nodes structure influence*:

$$D_4^2(a,b) = \mathbb{I}\{P_{a1} \neq P_{b1}\} + \sum_{i=1}^{C}(KFS_{ai} - KFS_{bi})^2. \quad (8)$$

Weights of terms in the above distance functions: For each distance function above (i.e., Equations 5 to 8), we normalize each term of the function to $[0,1]$. Also, we assume each term has equal importance in differentiating the difference between nodes and empirically set the weight of each term to 1.

Alternative designs: There are a few design alternatives for those node glyphs. For the node glyph in the plane of *prediction results comparison*, we can use a 2×2 grid to represent the *ground truth label* and *three model prediction results*. However,

such a design cannot effectively help users compare the metrics in the diagonal and will confuse users. Therefore, such a design is not adopted. For the node glyph in the plane of *surrounding nodes label consistency*, an alternative design is to use Parallel Coordinates Plot to display the five continuous metrics. However, it is generally hard for users to distinguish between two node glyphs. For the node glyph in the plane of *training nodes structure influence and feature influence*, we can use a similar node glyph in the plane of *prediction results comparison* to encode the GNN prediction result in the inner circle and encode the *label distribution* in the outer ring. However, to avoid any confusion in the node glyph between those planes, we do not use this design in *training nodes structure influence and feature influence* plane.
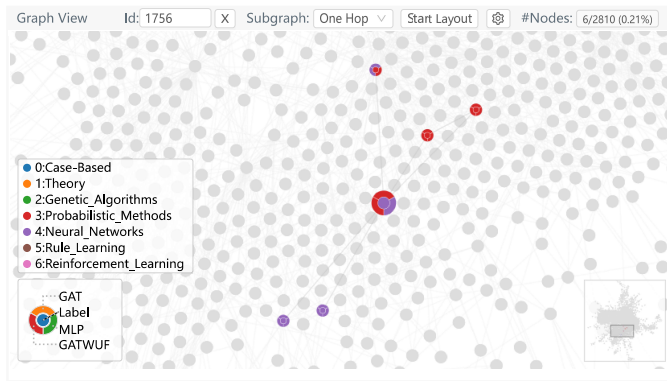

Fig. 5. Graph View enables users to inspect the graph structure. Node glyph in Graph View enables users to compare *three model prediction results* and *ground truth label* simultaneously. The color legend indicates which class the color represents. The legend for the node glyphs shows the position at which each metric is encoded. The color in the legend for node glyphs is only intended to show an example of node glyphs.

5.3.3 Graph View

We use the classic node-link diagram with the force-directed collision-avoidance layout to visualize the graph dataset. Users can get a sense of the distribution of the selected nodes in the graph, and inspect the neighborhood of the nodes (**R2, R3**).

To further facilitate the convenient exploration of the reasons for errors, we design a node glyph to encode a group of node-level metrics. The experts commented that they are interested in the *ground truth label*, and the *predictions of the GNN, GNNWUF, and MLP models*. Combining the four metrics, they are able to investigate the potential error types of the nodes. As shown in Fig. 5, the glyph designed to present the node-level metrics is similar to the design used in the Projection View. A legend for the glyph is also displayed at the corner of the Graph View as an easy reference for users.

The set of nodes selected in the Parallel Sets View or the Projection View is highlighted in the Graph View. Users can hover a node in the Graph View, which will be further highlighted with the radius doubled. The Graph View allows users to quickly check any interesting neighboring nodes. Users can also switch to the "Subgraph" mode. It will display the one-hop and two-hop neighbors of selected

nodes, enabling users to explore different hops of neighborhood nodes. Users can visualize the specific subgraphs on their own and can explore them by changing the "Subgraph" options. For example, generated subgraphs from GNN explainability methods such as GNNExplainer [12] can be loaded into the system for exploration. Please check Appendix B for more details. An overview of the graph is displayed in the bottom right-hand corner to support users navigating the graph. Users can click the specific position in the overview to navigate the displayed area of the graph. Users can choose to filter out the unfocused nodes to accelerate the rendering and reduce the visual clutter in the graph. To investigate the node features and *most similar features of training nodes*, users can click on the nodes of interest in the Graph View and further explore the node-level features in the Feature Matrix View.
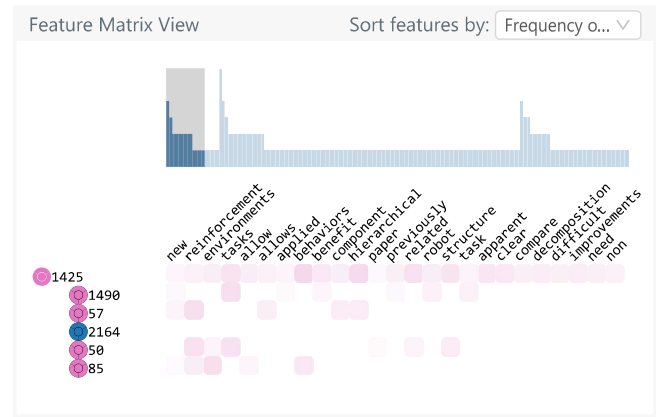

Fig. 6. Feature Matrix View includes brushable bar chart (Top) and feature matrix (Bottom).

5.3.4 Feature Matrix View

We design the Feature Matrix View to help users further explore the node features (**R3**), as shown in Fig. 6. The Feature Matrix View consists of two components, *i.e.*, a brushable bar chart and a feature matrix.

We first assume that all the features used in our dataset range from zero to one. For categorical features, one-hot encoding can be used to convert them to one-hot vectors, where the value of each element of the vector will be either 0 and 1 [61]. Thus, it can satisfy our assumption as well. The feature matrix indicates all the node features. The color encodes the prediction label of that node and the opacity encodes the specific feature value. In the brushable bar chart, the bar height encodes the count of any features with a value larger than 0 in the feature matrix. Users can brush a range of bars in the brushable bar chart and thus the feature matrix will display the specific range of feature dimensions. This makes it really convenient for users to inspect the features of nodes with a high dimensionality and without this design, the scalability of this view is not guaranteed. Users can change the sorting methods of feature dimensions. It can be sorted based on *node ordering* or *frequency of features*. When users select a subset of nodes in Parallel Sets View and Projection View, it will display the features of selected nodes. The hierarchical clustering algorithm and

optimal leaf ordering [62] will be employed to generate the node ordering. After sorting the nodes, the similarity will be calculated between two consecutive nodes. If they are very similar, we highlight them by adding a border in the rectangle in the rows of corresponding nodes. When a node is selected in the Graph View, it will display the features of that node and the *top-k most similar feature training nodes*. The training nodes will be sorted based on feature similarity with that node. When sorting feature dimensions based on the *frequency of features*, the following strategy is used. For each dimension of the features, we first count the frequency $|N|$ and then calculate the frequency of support $|SUPP|$, *i.e.*, the number of nodes with the same features and model prediction label as the first node. We then calculate the support rate of the features $SUPPRATE$ by using formula: $SUPPRATE = |SUPP|/|N|$. Therefore, when the support rate is high, it will have a higher ranking. When the support rate between two dimensions of a feature is the same, it is sorted based on the frequency of features. Then we can figure out what features can be supportive of the predictions for the GNN model.

6 EVALUATION

In this section, we demonstrate the effectiveness and usability of *GNNLens* through two case studies and structured interviews with GNN experts. We conduct two case studies with two experts E1 and E2 who have been introduced in Section 4.
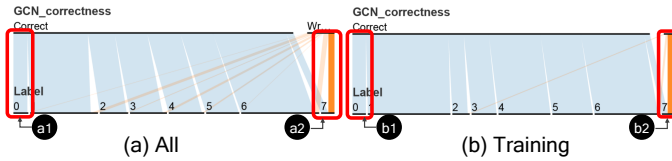

Fig. 7. The correlation between *GCN correctness* and *Label*: (a) All nodes in Amazon Photo dataset; (b) Training nodes in Amazon Photo dataset.

6.1 Case One: Error Pattern Analysis of GCN on Amazon Photo Dataset

This case study shows how our approach helps the model researcher explore the error patterns of GCN, one of the most representative GNN models, on the Amazon Photo dataset [63]. The Amazon Photo dataset is a co-purchasing network of 7,650 products. In this dataset, each node represents a product and is classified into one of the eight classes, including *File Photography, Digital Concepts, Binoculars & Scopes, Lenses, Tripods & Monopods, Video Surveillance, Lighting & Studio,* and *Flashes*. Each edge is a co-purchasing relationship, *i.e.*, products are purchased by the same customer. Each feature of the node is a vector of 0-1 value indicating whether the corresponding word appears in product reviews or not.

6.1.1 Developing Initial Hypotheses about the Possible Error Patterns in GNN Results

E1 started his analysis from the Parallel Sets View. E1 found that the GCN model achieves an accuracy of 91.15% on
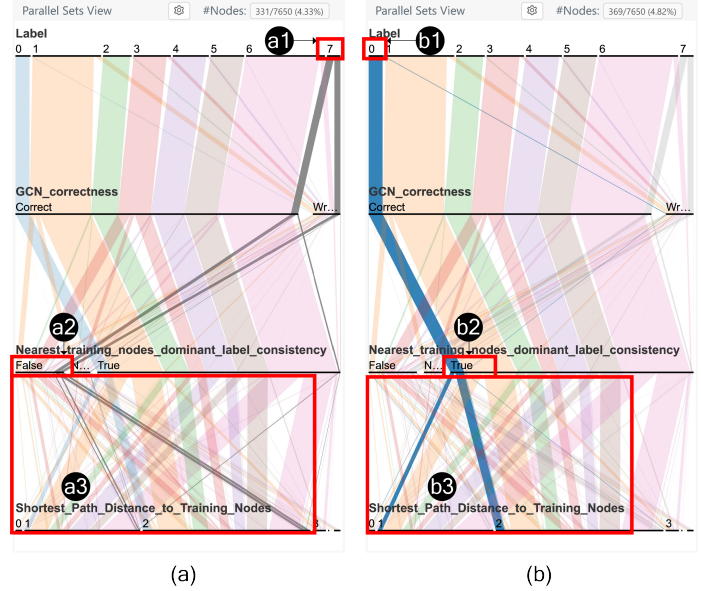

Fig. 8. The correlation among *Label*, *GCN correctness*, *nearest training nodes dominant label consistency*, and *shortest path distance to training nodes*: (a) Nodes with *Label "7"*; (b) Nodes with *Label "0"*.

the whole dataset. The test accuracy is 91.80%. The model performance is consistent with the results reported in other papers [63]. E1 changed the first axis of the Parallel Sets View to be *GCN correctness* by dragging the corresponding axis to the first axis. The total number of wrong prediction nodes is 677. Then, E1 explored the variables correlated with the wrong prediction. E1 put the *Label* in the second axis and found that the GCN model makes the most percentage of wrong predictions on the nodes of Class 7. This is indicated by the ribbon link flowing from the wrong category to the *ground truth label "7"*, which occupies the largest portion of the *ground truth label "7"* in Fig. 7(a2). E1 used the Control Panel (Fig. 1(a)) to see the training node information by ticking *"Training"*. E1 found that the training nodes are sampled with even probability from eight classes, which is shown by a similar distribution of *ground truth labels* in training nodes and all the nodes (Fig. 7(b)). The number of nodes with the *ground truth label "7"* is smaller (Fig. 7(a2)) than the number of nodes of other labels, and the number of training nodes with the ground truth label "7" is also small (Fig. 7(b2)). Perhaps this is the reason that GCN is unable to correctly classify the nodes in Class 7. However, E1 also found that the number of training nodes with the *ground truth label "0"* is also small (Fig. 7(b1)), but the GCN model correctly classifies most of the nodes in Class 0 (Fig. 7(a1)). E1 doubted his hypothesis and decided to further investigate the cause of wrong predictions (**R1**).

6.1.2 Forming the Hypothesis about Possible Error Patterns

To analyze the error patterns found in Section 6.1.1 from graph structure perspective, E1 selected four axes, including *Label*, *GCN correctness*, *nearest training nodes dominant label consistency*, and *shortest path distance to training nodes*, in the Parallel Sets View, as E1 believed that the axis *Label* can

help him select the specific label to check. The axis *GCN correctness* can help him to filter the nodes with the correct prediction or wrong prediction. The axes *nearest training nodes dominant label consistency* and *shortest path distance to training nodes* help him analyze the nodes from graph structure perspective. After hovering over *Label "0"* (Fig. 8(a1)) and *Label "7"* (Fig. 8(b1)), E1 found that for *nearest training nodes dominant label consistency*, most nodes (355/369) with *Label "0"* have a true value (Fig. 8(a2)), while most nodes (264/331) with *Label "7"* have a false value (Fig. 8(b2)). This shows that from graph structure perspective, it is easy for nodes with *Label "0"* to find training nodes with the same *ground truth label* through searching the training nodes with shortest path distance to *current nodes*, while nodes with *Label "7"* cannot satisfy these conditions. Moreover, E1 can find that *the shortest path distances to training nodes* for most nodes with *Label "0"* (Fig. 8(a3)) and *Label "7"* (Fig. 8(b3)) are less than or equal to 2. It means that nearest training nodes for most nodes with *Label "0"* and *Label "7"* are located in the input scope of GCN with two layers and will have an impact on the *prediction results of GCN*. Therefore, E1 speculated that it can be the possible reason for more classification errors when GCN is applied to the nodes of *Label "7"* (**R2**).

6.1.3 Analyzing the Cause of Error Patterns

To further verify the cause of the error patterns identified above, E1 selected 150 wrongly-classified nodes of *Label "7"* by GCN in Parallel Sets View (Fig. 1(b1)) and further explored other views. From the planes of *training nodes structure influence and feature influence* in the Projection View (Fig. 1(c)), few nodes of *Label "7"* appear in the label distribution and many prediction labels are consistent with the largest component on the right-hand side of the glyph. It indicates that the structure and the features of training nodes of *Label "7"* do not have a significant influence on the prediction results of testing nodes of *Label "7"* when compared with the influence of other training nodes. From the plane of *surrounding node label consistency*, the *label consistency* of most nodes is relatively small. It can be explained that the labels of the neighbors of these nodes are mostly inconsistent with the *label* of the *current node*. It means that the information of neighboring nodes can confuse the GCN prediction results of the *current node*.

E1 further explored three training nodes that are also misclassified. E1 lasso-selected them (Fig. 1(c1)), and then selected one of the nodes in the Graph View for further checking (Fig. 1(d1)). E1 found that the *ground truth labels* of the majority of its neighbors are different from its own *ground truth label*, and its *GCNWUF prediction result* is consistent with the *GCN prediction result* (Fig. 1(d1)). In the Feature Matrix View (Fig. 1(e)), E1 observed that the frequency of features is very high in the front of the bar chart (Fig. 1(e1)), and the last features of the *current node* also share a lot of common features with the *top-k most similar training nodes* (Fig. 1(e2)). It indicates that the *current node* shares a lot of common features with the *top-k most similar training nodes*. Moreover, the *ground truth labels* of those nodes are different from the *ground truth label* of the *current node* (Fig. 1(e3)). Therefore, from the above observations, E1 speculated that the classification error of the *current node* is probably due to the existence of a large number of neighboring nodes whose

ground truth labels are totally different from the *current node*. Also, the *current node* lacks discriminative features.

However, E1 also spotted that the *MLP prediction result* is consistent with the current node's *ground truth label* (Fig. 1(d1)). E1 quickly raised one question: *Does the MLP model memorize the features of training nodes of Label "7" and fail to generalize to other nodes with the same ground truth labels?* E1 checked the performance of MLP in the Parallel Sets View and found that MLP correctly classifies all the training nodes of *Label "7"*, but incorrectly classifies all the test nodes of *Label "7"*. E1 thought that it is very hard for GCN models to correctly classify the test nodes with a strong influence of the structures and the features of nodes of other *classes*. Thus, E1 believed that when training the GCN using cross-entropy loss, he can increase the loss of the training nodes of *Label "7"* to improve the classification of the nodes of *Label "7"*, decreasing the strong influence of nodes of other *ground truth labels* [64] (**R3**).

6.2 Case Two: Error Pattern Analysis of GAT on Cora-ML Dataset

The model developer, E2, often needs to use GNNs to model network data in real applications. This case study shows how *GNNLens* assists him in analyzing another representative GNN model (*i.e.*, the GAT model) on the Cora-ML dataset [65]. Specifically, the Cora-ML dataset is a citation network of 2,810 scientific publications. Each node in the Cora-ML dataset represents a paper and is classified into one of seven classes, including *Case-Based, Theory, Genetic Algorithms, Probabilistic Methods, Neural Networks, Rule Learning,* and *Reinforcement Learning*. Each edge is a citation relationship. Each feature of the node is a vector, with each feature element ranging from 0 to 1. A feature element larger than 0 represents that the paper abstract contains the corresponding word.

6.2.1 Forming the Hypothesis about Possible Error Patterns

In the Parallel Sets View, E2 found that the GAT achieves an accuracy of 86.16% on the whole dataset and 84.70% on the testing set. To check whether there are error patterns resulting from node features rather than graph structure, E2 selected three axes, including *nearest training nodes dominant label consistency*, *top-k most similar training nodes dominant label consistency*, and *GAT correctness*, as E2 believed that the axis *GAT correctness* can help him to filter the nodes with the correct prediction or wrong prediction. The axes *nearest training nodes dominant label consistency* and *top-k most similar training nodes dominant label consistency* can help him determine whether the source of influence on prediction results is the graph structure or the node features. E2 found an interesting set of nodes with the *nearest training nodes dominant label consistency* as true, the *top-k most similar training nodes dominant label consistency* as false, and the *GAT correctness* as wrong. E2 decided to further explore them and select those nodes in the Parallel Sets View by clicking the ribbon satisfying the above conditions.

In the Projection View (Fig. 9(a)), E2 found that in the *training nodes feature influence* plane, the left side and the right side of the glyphs have mostly the same color
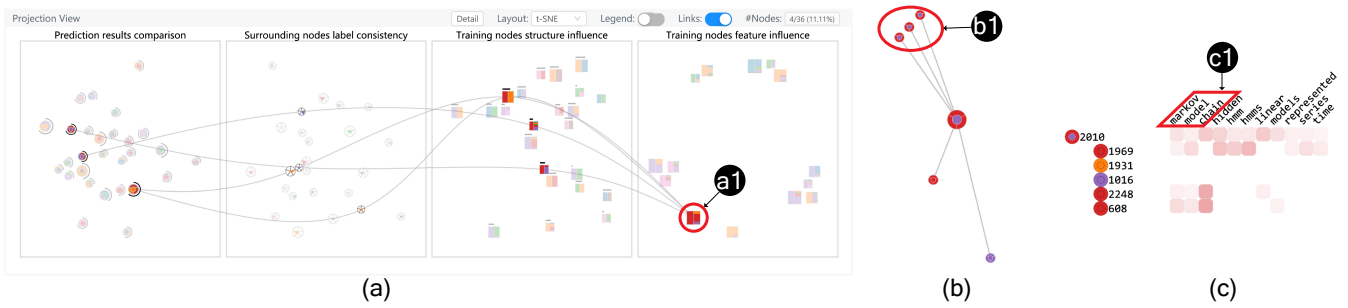
Fig. 9. E2 selected a cluster (a1) to inspect in Projection View (a). Then E2 selected a node in Graph View (b) to further inspect its neighborhood. E2 found that in Feature Matrix View (c), the first few words are "markov", "model", "chain" (c1), which are common words in the paper belong to *Probabilistic Methods*. E2 verified that this node is mislabeled and should be labeled as *Probabilistic Methods*.

(Fig. 9(a1)). This consistency means that the *GAT prediction labels* are consistent with the *top-k similar features training nodes dominant labels*. E2 speculated that the node features have a great impact on *GAT prediction*. Then, E2 selected one of the clusters, and further checked the other planes of the Projection View. In the *training nodes structure influence* plane, E2 saw that the left-hand side and the right-hand side of the highlighted node glyphs are in different colors. In the *surrounding nodes label consistency* plane, it showed that the *label consistency* is generally large, indicating that the surrounding *ground truth label* is consistent with the current node's *ground truth label*. In the *prediction results comparison* plane, it was found that the *prediction results of MLP* are consistent with the *prediction results of GAT*. It further confirms that the node features of this cluster of nodes have a great impact on the GAT predictions on them (**R1, R2**).

6.2.2 Analyzing the Cause of Error Patterns

E2 also explored the Graph View and selected a node to verify his observation. E2 found that the node has some neighbor nodes with a different *ground truth label*, as shown in Fig. 9(b). In the Feature Matrix View (Fig. 9(c)), E2 can see that the *ground truth labels* of most training nodes are the same as its *GAT prediction label*. The first few words are "markov", "model", "chain" (Fig. 9(c1)), which are common words in papers about *Probabilistic Methods*. This article has these words, but the ground truth class of this article is *Neural Network*. E2 thought this node is mislabeled rather than misclassified. To verify whether it is labeled correctly, E2 further checked its title by hovering over the glyph in the Feature Matrix View and found that the title is *"Equivalence of Linear Boltzmann Chains and Hidden Markov Models"* [66]. After further checking the content of this article, E2 concluded that the ground truth class of this article should be *Probabilistic Methods* rather than *Neural Networks*. E2 further checked the three neighboring nodes of this node, which shares the same properties that the ground truth class is *Neural Network* but all three model prediction results are *Probabilistic Methods* (Fig. 9(b1)). E2 found that they share the same title *"Gibbs-Markov Models"* [67] and should belong to the class *"Probabilistic Methods"*. E2 suggested that the wrong labels should be identified and corrected, especially

when the articles have words "markov", "model", "chain", and its ground truth class is *Neural Network* (**R3**).

6.3 Expert Interviews

6.3.1 Participants

To further evaluate the usefulness and usability of *GNNLens*, we also conducted interviews with 12 experts (E3-E14, age 22-46, $\mu = 26.33$, $\sigma = 6.09$). All 12 experts have experience in the application or design of GNN models. None of the 12 experts are co-authors of this paper.

6.3.2 Methodology

We first introduced *GNNLens* to experts and demonstrated the case study with E1. After experts learned about how *GNNLens* works, we asked them to explore *GNNLens* by following the demonstrated workflow to find the cause of the prediction errors of individual nodes and extract general error patterns in GNN prediction results. Finally, we asked them to finish a post-interview questionnaire with 5-point Likert scale questions (1-Strongly Disagree, 5-Strongly Agree) to collect their feedback on *GNNLens*. As shown in Table 1, the questionnaire mainly comprises evaluations on the effectiveness, visual design, and usability of *GNNLens*. Results and feedback are summarized accordingly.

6.3.3 Results

Effectiveness: After exploring *GNNLens*, all the experts appreciated our efforts in making such an effective system to help them understand and diagnose the GNN models. As shown in Table 1, experts agreed that *GNNLens* can help users understand the overview of dataset and GNN prediction results, identify the error patterns, and analyze the causes of error patterns. E5 commented that *"the visualization is clear and insightful, leading to the rapid discovery of the underlying error patterns, and thereby enables future endeavors in improving the GNNs."* It further supports that *GNNLens* can help users diagnose the GNNs used in node classification tasks.

Visual design: From the results on the evaluation of visual design, we observe that most of the experts (8/12) agreed that the overall visual designs of *GNNLens* are intuitive and easy to understand. The experts agreed that the Graph View is the most intuitive and easy to understand, as shown in

	Question	Score
	Effectiveness	
Q1	The system can help me understand the overview of dataset and GNN prediction results.	4.50 ± 0.65
Q2	The system can help me identify the error patterns of GNNs.	4.25 ± 0.72
Q3	The system can help me analyze the causes of error patterns of GNNs.	3.92 ± 0.76
	Visual design	
Q4	The Parallel Sets View is intuitive and easy to understand.	4.08 ± 0.95
Q5	The Projection View is intuitive and easy to understand.	3.83 ± 0.99
Q6	The Graph View is intuitive and easy to understand.	4.50 ± 0.76
Q7	The Feature Matrix View is intuitive and easy to understand.	4.00 ± 0.82
Q8	The overall system is intuitive and easy to understand.	3.83 ± 0.90
	Usability	
Q9	It is easy to learn and use the system.	3.67 ± 1.03
Q10	I would like to recommend this system to others who are working on diagnosing GNNs.	4.42 ± 0.76
Q11	I think it is useful to use this system to diagnose GNNs.	4.00 ± 0.82
Q12	I would use this system to diagnose errors of GNNs in the future.	4.08 ± 0.76

TABLE 1
Evaluation on effectiveness (Q1-Q3), visual design (Q4-Q8), and usability (Q9-Q12) of *GNNLens*. Scores (mean $\pm$ std) for each question are reported.

Table 1. In terms of the Projection View, the experts think that the glyphs can help users understand the metrics of each node but the learning curve is a bit steep.

Usability: Most of the experts (7/12) agreed that *GNNLens* is easy to use and easy to learn. As shown in Table 1, the experts agreed that they would like to recommend the system to others who are working on diagnosing GNNs. They also agreed that it is useful to diagnose GNNs and would like to use the system in the future. E7 further positively commented that *"the response time of the system is quite fast and the overall design is clear."* Such results support that *GNNLens* will be widely used by users to understand and diagnose GNNs.

Suggestions: Experts also gave helpful suggestions for improving *GNNLens* to further support their analysis of GNN models. For example, it would be more helpful if the system can provide recommendations on how to further improve GNNs if some error patterns of GNNs are found. Moreover, if the system can support customized datasets like heterogeneous graphs, the generalizability of the system will be further enhanced.

7 DISCUSSIONS AND FUTURE WORK

Generalizability: *GNNLens* can be applied to analyze various kinds of GNN models and different datasets. However, it can only be utilized to analyze the task of node classification. It does not support the analysis of link prediction and graph classification. Moreover, if the dataset has multiple relations or the edges have features, the system cannot be directly used to analyze those customized data. Also, users may want to further customize some metrics to inspect

the relationship between them and the model performance. However, the system does not support it currently.

Scalability: One limitation of *GNNLens* is its scalability and we have attempted to mitigate this issue by different means. For example, the Projection View displays the individual node glyphs for each node. Due to the limited screen space, we cannot display up to 300 nodes. We improved the Projection View by using a hierarchical clustering algorithm to make it scale up to more nodes, as described in Section 5.3.2. The node glyphs are used to represent clusters and details can be checked on demand. This significantly improves the scalability of the Projection View. For the Graph View, in order to accelerate the rendering speed, we enable users to display only the focused nodes and their neighbors, without rendering other nodes. Some scalability issues cannot be easily resolved. For example, in the Projection View, if the number of labels is increasing, the number of different colors to encode different labels will also be increased. However, too many colors will increase the cognitive load for users. Moreover, as the number of selected data points is increasing, the number of links will also increase, which can result in visual clutters.

Learning curve of the Projection View: Compared with other views of *GNNLens*, it took more time for the domain experts to understand the glyphs and metrics used in the projection view. To further help experts understand the meaning of glyphs, glyph legends have been added to the Projection View for an easy reference. The Projection View plays an important role in helping users understand the properties of the selected nodes and narrow down to one or several nodes for further analysis in the Graph View and Feature Matrix View. Without this design, it is hard to explore the properties among those selected nodes.

Future work: In the future, we plan to generalize *GNNLens* to other graph-related tasks, like link prediction and graph classification. We plan to make Parallel Sets View and Projection View more configurable, such as enabling users to define their own metrics, such as clustering coefficients to show. We also want to further improve the running performance of *GNNLens* and support the graph dataset with more nodes and higher dimensional features. Moreover, We want to generalize *GNNLens* to support the dynamic insertion of nodes and edges to see the corresponding change in GNN prediction results. It will help users understand how the graph structure benefits the model to improve performance.

8 CONCLUSION

In this paper, we present *GNNLens*, a visual analytics system to help model developers and users understand and diagnose GNN model prediction results. *GNNLens* comprises four visualization components: the Parallel Sets View enables users to see the distribution of metrics; the Projection View presents a set of 2D projections of the selected nodes according to metrics summarized from different perspectives enabling users to extract potential clusters of nodes; the Graph View shows the whole graph; and the Feature Matrix View shows the selected node feature information. It further enables users to check the detailed information of individual nodes. All four visualization components are linked together to support users to analyze GNN models

simultaneously from multiple angles and extract general error patterns in GNN prediction results. Two case studies and expert interviews demonstrate the effectiveness and usability of our system *GNNLens*.

ACKNOWLEDGMENTS

REFERENCES

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the 5th International Conference on Learning Representations*, 2016.

[2] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.

[3] L. Chen, Y. Xie, Z. Zheng, H. Zheng, and J. Xie, "Friend recommendation based on multi-social graph convolutional network," *IEEE Access*, vol. 8, pp. 43 618–43 629, 2020.

[4] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur, "Protein interface prediction using graph convolutional networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 6530–6539.

[5] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proceedings of the 6th International Conference on Learning Representations*, 2018.

[6] F. Hohman, M. Kahng, R. Pienta, and D. H. Chau, "Visual analytics in deep learning: An interrogative survey for the next frontiers," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 8, pp. 2674–2693, 2019.

[7] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu, "Towards better analysis of deep convolutional neural networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 91–100, 2016.

[8] Y. Ming, S. Cao, R. Zhang, Z. Li, Y. Chen, Y. Song, and H. Qu, "Understanding hidden memories of recurrent neural networks," in *IEEE Conference on Visual Analytics Science and Technology*, 2017, pp. 13–24.

[9] J. Wang, L. Gou, H. Yang, and H.-W. Shen, "Ganviz: A visual analytics approach to understand the adversarial game," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 6, pp. 1905–1917, 2018.

[10] J. Wang, L. Gou, H.-W. Shen, and H. Yang, "Dqnviz: A visual analytics approach to understand deep q-networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 288–298, 2018.

[11] P. E. Pope, S. Kolouri, M. Rostami, C. E. Martin, and H. Hoffmann, "Explainability methods for graph convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 772–10 781.

[12] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "Gnnexplainer: Generating explanations for graph neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 9240–9251.

[13] X. Li and J. Saude, "Explain graph neural networks to understand weighted graph features in node classification," *arXiv preprint arXiv:2002.00514*, 2020.

[14] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.

[15] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *Proceedings of the 2nd International Conference on Learning Representations*, 2014.

[16] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3844–3852.

[17] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in Neural Information Processing Systems*, 2015, pp. 2224–2232.

[18] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 1993–2001.

[19] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification," in *Proceedings of the World Wide Web Conference*, 2018, pp. 499–508.

[20] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proceedings of the 33rd International Conference on Machine Learning*, 2016, pp. 2014–2023.

[21] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1416–1424.

[22] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5115–5124.

[23] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel, "Gated graph sequence neural networks," in *Proceedings of the 4th International Conference on Learning Representations*, 2016.

[24] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, 2015, pp. 1556–1566.

[25] V. Zayats and M. Ostendorf, "Conversation modeling on reddit using a graph-structured LSTM," *Transactions of the Association for Computational Linguistics*, vol. 6, pp. 121–132, 2018.

[26] X. Liang, X. Shen, J. Feng, L. Lin, and S. Yan, "Semantic object parsing with graph LSTM," in *European Conference on Computer Vision*. Springer, 2016, pp. 125–143.

[27] Y. Wang, Z. Jin, Q. Wang, W. Cui, T. Ma, and H. Qu, "Deepdrawing: A deep learning approach to graph drawing," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 1, pp. 676–686, 2020.

[28] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[29] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proceedings of the 7th International Conference on Learning Representations*, 2019.

[30] J. Zhang, Y. Wang, P. Molino, L. Li, and D. S. Ebert, "Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 364–373, 2018.

[31] B. Alsallakh, A. Hanbury, H. Hauser, S. Miksch, and A. Rauber, "Visual methods for analyzing probabilistic classification data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 1703–1712, 2014.

[32] Y. Ming, H. Qu, and E. Bertini, "Rulematrix: Visualizing and understanding classifiers with rules," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 342–352, 2019.

[33] J. Wang, L. Gou, W. Zhang, H. Yang, and H. Shen, "Deepvid: Deep visual interpretation and diagnosis for image classifiers via knowledge distillation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 6, pp. 2168–2180, 2019.

[34] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[36] D. Liu, W. Cui, K. Jin, Y. Guo, and H. Qu, "Deeptracker: Visualizing the training process of convolutional neural networks," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 1, pp. 1–25, 2018.

[37] N. Pezzotti, T. Höllt, J. Van Gemert, B. P. Lelieveldt, E. Eisemann, and A. Vilanova, "Deepeyes: Progressive visual analytics for designing deep neural networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 98–108, 2017.

[38] H. Strobelt, S. Gehrmann, H. Pfister, and A. M. Rush, "Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 667–676, 2017.

[39] H. Strobelt, S. Gehrmann, M. Behrisch, A. Perer, H. Pfister, and A. M. Rush, "Seq2seq-vis: A visual debugging tool for sequence-

[40] M. Liu, J. Shi, K. Cao, J. Zhu, and S. Liu, "Analyzing the training processes of deep generative models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 77–87, 2017.

[41] M. Kahng, N. Thorat, D. H. P. Chau, F. B. Viégas, and M. Wattenberg, "Gan lab: Understanding complex deep generative models using interactive visual experimentation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 1–11, 2018.

[42] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[43] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *IEEE Data Engineering Bulletin*, vol. 40, no. 3, pp. 52–74, 2017.

[44] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *arXiv preprint arXiv:1909.01315*, 2019.

[45] J. Wu, J. He, and J. Xu, "Demo-net: Degree-specific graph neural networks for node and graph classification," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 406–415.

[46] B. Xu, J. Huang, L. Hou, H. Shen, J. Gao, and X. Cheng, "Label-consistency based graph neural networks for semi-supervised node classification," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020, pp. 1897–1900.

[47] Y. Yang, X. Wang, M. Song, J. Yuan, and D. Tao, "Spagan: Shortest path graph attention network," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019, pp. 4099–4105.

[48] Y. Chen, L. Wu, and M. Zaki, "Iterative deep graph learning for graph neural networks: Better and robust node embeddings," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[49] F. Bendix, R. Kosara, and H. Hauser, "Parallel sets: Visual analysis of categorical data," in *IEEE Symposium on Information Visualization*, 2005, pp. 133–140.

[50] D. Ren, S. Amershi, B. Lee, J. Suh, and J. D. Williams, "Squares: Supporting interactive performance analysis for multiclass classifiers," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 61–70, 2016.

[51] J. Wexler, M. Pushkarna, T. Bolukbasi, M. Wattenberg, F. B. Viégas, and J. Wilson, "The what-if tool: Interactive probing of machine learning models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 1, pp. 56–65, 2020.

[52] D. Dingen, M. van 't Veer, P. Houthuizen, E. H. J. Mestrom, H. H. M. Korsten, A. R. A. Bouwman, and J. J. van Wijk, "Regressionexplorer: Interactive exploration of logistic regression models with subgroup analysis," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 246–255, 2019.

[53] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. P. Chau, "Activis: Visual exploration of industry-scale deep neural network models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 88–97, 2018.

[54] Z. Vosough and V. Vasyutynskyy, "Using parallel sets for visualizing results of machine learning based plausibility checks in product costing." in *VisBIA@ AVI*, 2018, pp. 4–11.

[55] A. Chaudhuri, "A visual technique to analyze flow of information in a machine learning system," in *Visualization and Data Analysis*. Ingenta, 2018.

[56] A. Inselberg and B. Dimsdale, "Parallel coordinates: A tool for visualizing multi-dimensional geometry," in *Proceedings of the First IEEE Conference on Visualization*, 1990, pp. 361–378.

[57] Q. Li, K. S. Njotoprawiro, H. Haleem, Q. Chen, C. Yi, and X. Ma, "Embeddingvis: A visual analytics approach to comparative network embedding inspection," in *IEEE Conference on Visual Analytics Science and Technology*, 2018, pp. 48–59.

[58] L. van der Maaten and G. Hinton, "Viualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 11 2008.

[59] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.

[60] B. Everitt, S. Landau, M. Leese, and D. Stahl, *Cluster Analysis*, 5th ed. Wiley, 2011.

[61] J. T. Hancock and T. M. Khoshgoftaar, "Survey on categorical data for neural networks," *Journal of Big Data*, vol. 7, no. 1, pp. 1–41, 2020.

[62] Z. Bar-Joseph, D. K. Gifford, and T. S. Jaakkola, "Fast optimal leaf ordering for hierarchical clustering," in *Proceedings of the Ninth International Conference on Intelligent Systems for Molecular Biology*, 2001, pp. 22–29.

[63] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *Relational Representation Learning Workshop, NeurIPS*, 2018.

[64] Z.-H. Zhou and X.-Y. Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, pp. 63–77, 2005.

[65] A. Bojchevski and S. Günnemann, "Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking," in *Proceedings of the 6th International Conference on Learning Representations*, 2018.

[66] D. J. MacKay, "Equivalence of linear boltzmann chains and hidden markov models," *Neural Computation*, vol. 8, no. 1, pp. 178–181, 1996.

[67] J. D. Lafferty, "Gibbs-markov models," *Computing Science and Statistics*, pp. 370–377, 1996.
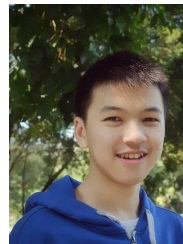
Zhihua Jin is currently a Ph.D. student at the Hong Kong University of Science and Technology (HKUST). He received his BEng degree in Computer Science and Technology from Zhejiang University in 2019. His research interests lie in the intersection of visualization and machine learning, especially explainable artificial intelligence (XAI).

Yong Wang is currently an assistant professor in School of Computing and Information Systems at Singapore Management University. His research interests include data visualization, visual analytics and explainable machine learning. He obtained his Ph.D. in Computer Science from Hong Kong University of Science and Technology in 2018. He received his B.E. and M.E. from Harbin Institute of Technology and Huazhong University of Science and Technology, respectively. For more details, please refer to http://yong-wang.org.

Qianwen Wang is a PostDoc researcher at Harvard University. Her work strives to facilitate the communication between humans and machine learning models through interactive visual analytics. Her research interests include visual analytics, explainable machine learning, and narrative visualization. She obtained her Ph.D in Hong Kong University of Science and Technology and her BS degree from Xi'an Jiaotong University. Please refer to http://wangqianwen0418.github.io for more details.

Yao Ming is a research scientist at Bloomberg LP. His research focus on visual analytics, explainable machine learning, and natural language processing. He received a Ph.D. in Computer Science from the Hong Kong University of Science and Technology and a B.S. from Tsinghua University. For more details please refer to https://www.myaooo.com.

Tengfei Ma is a research staff member of IBM Research AI. Prior to joining IBM T. J. Watson Research Center, he obtained his Ph.D. from the University of Tokyo and worked as a researcher in IBM Research Tokyo for one year. Before that he got his master's degree from Peking University and his bachelor degree from Tsinghua University. His research interests have spanned a number of different topics in machine learning and natural language processing. Particularly, his recent research is focused on graph neural networks and their applications.

Huamin Qu is a professor in the Department of Computer Science and Engineering (CSE) at the Hong Kong University of Science and Technology (HKUST) and also the director of the interdisciplinary program office (IPO) of HKUST. He obtained a BS in Mathematics from Xi'an Jiaotong University, China, an MS and a PhD in Computer Science from the Stony Brook University. His main research interests are in visualization and human-computer interaction, with focuses on urban informatics, social network analysis, E-learning, text visualization, and explainable artificial intelligence (XAI).