Interaction-Focused Anomaly Detection on Bipartite Node-and-Edge-Attributed Graphs

Rizal Fathony
Grab
Jakarta, Indonesia
rizal.fathony@grab.com

Jenn Ng
Grab
Singapore
jenn.ng@grab.com

Jia Chen
Grab
Singapore
jia.chen@grab.com

***Abstract*—Many anomaly detection applications naturally produce datasets that can be represented as bipartite graphs (user–interaction–item graphs). These graph datasets are usually supplied with rich information on both the entities (nodes) and the interactions (edges). Unfortunately, previous graph neural network anomaly models are unable to fully capture the rich information and produce high-performing detections on these graphs, as they mostly focus on homogeneous graphs and node attributes only. To overcome the problem, we propose a new graph anomaly detection model that focuses on the rich interactions in bipartite graphs. Specifically, our model takes a bipartite node-and-edge-attributed graph and produces anomaly scores for each of its edges and then for each of its bipartite nodes. We design our model as an autoencoder-type model with a customized encoder and decoder to facilitate the compression of node features, edge features, and graph structure into node-level latent representations. The reconstruction errors of each edge and node are then leveraged to spot the anomalies. Our network architecture is scalable, enabling large real-world applications. Finally, we demonstrate that our method significantly outperforms previous anomaly detection methods in the experiments.**

***Index Terms*—Anomaly Detection, Graph Neural Networks**

I. Introduction

Identifying behaviors that differ singularly from the majority has become an important area in various applications across many industries [1]. The occurrence of these (rare) anomaly behaviors may have serious implications in many domains. In the finance area, for example, the occurrence of anomaly events may indicate financial frauds such as stolen cards or money laundering [2], whereas, in a network security system, anomalous events could mean security breaches [3]. For this reason, developing machine learning algorithms for anomaly detection has become one of the essential research areas for many fields above, as well as other fields such as manufacturing, healthcare, insurance, and many others [4].

In most settings, anomaly detection is done without label supervision, i.e., in an unsupervised learning fashion [4]. This provides benefits over supervised learning techniques due to its flexibility. Getting anomaly labels is usually hard as anomalous events rarely occur [4]. Additionally, in real-world applications like fraud detection, fraudsters are incentivized to adversarially innovate their methods of conducting fraud. This makes any supervised models relying on historical labels unable to detect new types of fraud [5]. Having a high-performing unsupervised anomaly detection system is helpful in these cases.

Bipartite graphs are the main abstraction for modeling interactions between two groups, commonly referred to as the user–interaction–item graph [6], [7]. Some examples are the consumer–purchase–product graph on e-commerce websites, the user–review–movie graph on movie review websites, ip-address–request–server graph in network monitoring systems, etc. In real-world applications, those bipartite graphs are usually supplied with a lot of information on both the entity (node) and the interaction (edge). For example, in e-commerce transactions, both the consumer and product nodes have rich information, such as consumer profiles and historical preferences, product descriptions, categories, etc. The transaction itself also has rich information, such as price, discount, payment, reviews, etc. To be able to produce accurate anomaly detection systems, this rich information needs to be considered by the model, in addition to the graph structure itself. For example, two customers purchasing the same product, one may conduct a fraud transaction, whereas another one may perform a legit transaction, depending on the transaction's payment profile.

Attributed graphs provide a tool to capture the rich information by encoding them as the node and edge features. Therefore, the model's ability to work on bipartite node-and-edge-attributed graphs is crucial for the applications above. Unfortunately, previous anomaly detection models are unable to fully take into account the rich information in the graph. Most of the graph neural networks (GNN) based anomaly detection models for attributed graphs only handle homogeneous graphs (such as DOMINANT [8], AnomalyDAE [9], etc. [10]–[13]) or graphs with node attributes only (such as [8]–[14]). Other unsupervised graph models handle bipartite graphs, but only accept non-attributed graphs and mostly only focus on node embedding rather than anomaly detection [6], [7], [15]–[17]. In general, many GNN models accept edge features. However, those models are usually developed for supervised learning settings rather than unsupervised settings. Extending those models to unsupervised anomaly detection tasks is not trivial. In addition, many graph anomaly detection models, such as DOMINANT and AnomalyDAE, rely on having access to the full adjacency matrix in the learning process. This limits the practicality of the models for real-world applications as they have scalability issues when handling large-size graphs.

Motivated by the shortcomings above, we propose a new graph neural network model for unsupervised anomaly

detection on bipartite node-and-edge-attributed graphs. We name our method the **B**ipartite Node-and-**E**dge-**A**ttributed **N**etworks (**GraphBEAN**). To achieve the desired property, GraphBEAN compresses the bipartite input graph with its node and edge features into low-dimensional (node-only) latent representations for each node in both node-sets, using a series of customized graph convolution layers; and then aims to reconstruct the topological structure of the graph as well as its node and edge attributes. This network construction forces the latent node representations to encode information about the graph structure, (K-hops) neighboring nodes and edges, as well as the attributes attached to those nodes and edges. The reconstruction errors of the edges and the nodes following the encoder-decoder phase above are then leveraged to spot the anomalous edges and nodes in the input graph.

In addition, GraphBEAN is designed with scalability in mind via customized neighborhood sampling. This enables it to be applied to large-size graph data. GraphBEAN is also capable of performing inductive learning and is therefore applicable to newly observed data. We finally demonstrate the empirical benefit of our method compared to previous graph anomaly models on several public bipartite graph datasets. To the best of our knowledge, GraphBEAN is the first GNN anomaly model for bipartite node-and-edge-attributed graphs.

II. Related Works

Classical anomaly detection methods on attributed graphs can be grouped into several categories [5]. The first category focuses on anomalies that deviate significantly from other members of specific communities [18]–[20]. Other anomaly detection models focus on identifying rare substructures in the graph [21]–[23]. Shah et al. [24] provide a greedy algorithm for clustering and scoring nodes based on edge attributes. Finally, Li et al. [25] and Peng et al. [26] provide residual-based anomaly models where the anomaly score is calculated from the residual between true data and estimated data. Despite the research progress above, classical models are constrained by their shallow learning mechanism, limiting their capability to detect complex modalities of the interactions in the graph [8].

In recent years, advances in deep neural network architectures for graph data have been growing fast due to their superior performance, particularly in supervised and semi-supervised settings. Graph Convolutional Network (GCN) [27] provides a fast generalization of convolutional operation to general graph structures. GraphSAGE [28] extends the GCN architecture to enable inductive learning by sampling and aggregating features from the local neighborhood. Many other architectures with slightly different flavors, such as Graph Attention Networks (GAT) [29], and Relational GCN (RGCN) [30], were also successful in their respective learning settings.

Motivated by the success of the GNN architectures in supervised and semi-supervised settings, several methods have been proposed to bring the benefit of GNN models to anomaly detection problems. DOMINANT [8] provides an autoencoder-like architecture for graph anomaly detection, where the anomaly score is computed from the autoencoder's reconstructed error. AnomalyDAE [9] extends the architecture with two autoencoders, each one for the graph structure and the attributes. AEGIS [31] enables graph anomaly detection models to be applied in inductive settings where we want to apply the model to new unobserved sub-graphs. Other GNN models, such as AdONE [10], GAAN [11], ANEMONE [12], GUIDE [32], CONAD [13], AHEAD [14], VGOD [33], and ACT [34] were also proposed for anomaly detection problems under slightly different settings.

III. Problem Formulation

We start our problem formulation by describing our notations as follows. We represent scalar values with italic fonts (e.g., k), whereas vectors and matrices are represented with lower-case and upper-case bold fonts (e.g., $\mathbf{x}$ and $\mathbf{X}$), respectively. We also use calligraphic fonts to denote sets (e.g., $\mathcal{U}$ and $\mathcal{E}$). In some cases, sets can also be denoted with upper-case letters (e.g., U and V). In all of those symbols, we use both sub-script and super-script to annotate the variables (e.g., $\mathbf{X}^u$ and $\mathbf{x}_i$). Using the conventions above, we define the bipartite attributed graph studied in this paper as follows:

Definition 1: A bipartite node-and-edge-attributed graph $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E}, \mathbf{X}^u, \mathbf{X}^v, \mathbf{X}^e)$ *consists of:*

- *the first set of nodes* $\mathcal{U} = \{u_1, u_2, \cdots, u_{n_u}\}$*;*
- *the second set of nodes* $\mathcal{V} = \{v_1, v_2, \cdots, v_{n_v}\}$*;*
- *the set of edges* $\mathcal{E}$*, indexed using* $e_{i,j}$*,* $i \in \{1, \cdots, n_u\}$*,* $j \in \{1, \cdots, n_v\}$*, with* n_e *represents its cardinality;*
- $\mathbf{X}^u \in \mathbb{R}^{n_u \times m_u}$*, the features for every node in* $\mathcal{U}$*;*
- $\mathbf{X}^v \in \mathbb{R}^{n_v \times m_v}$*, the features for every node in* $\mathcal{V}$*;*
- $\mathbf{X}^e \in \mathbb{R}^{n_e \times m_e}$*, the features for every edge in* $\mathcal{E}$*.*

Here, n *is the number of nodes/edges, and* m *represents the dimension of node/edge features. The subscripts in* n *and* m *indicate which set the variables represent (e.g.,* n_u *for the set* $\mathcal{U}$ *and* n_e *for the set of edges* $\mathcal{E}$*).*

The structure of the attributed graph $\mathcal{G}$ is represented in an adjacency-like matrix $\mathbf{A} \in \mathbb{R}^{n_u \times n_v}$, where its (i, j)-th item $\mathbf{A}_{i,j} = 1$ if there is an edge between u_i and v_j, or otherwise $\mathbf{A}_{i,j} = 0$. Note that unlike the adjacency matrix in a regular (homogeneous) graph, matrix $\mathbf{A}$ need not be a square matrix as the number of nodes in $\mathcal{U}$ and $\mathcal{V}$ can be different. We also define $\mathcal{N}$ as the neighboring operator that returns the list of all neighboring nodes. Specifically, for a node u in the first node-set, $\mathcal{N}(u_i) = \{v_j \in \mathcal{V} \mid \mathbf{A}_{i,j} = 1; j \in \{1, \cdots, n_v\}\}$. Similarly, for a node v in the second node-set, $\mathcal{N}(v_j) = \{u_i \in \mathcal{U} \mid \mathbf{A}_{i,j} = 1; i \in \{1, \cdots, n_u\}\}$. In addition, we define another function $\mathcal{M}$ which behaves similarly to $\mathcal{N}$. However, instead of returning neighboring nodes, it returns the edges connecting the target node to the neighboring nodes. In particular, $\mathcal{M}(u_i) = \{e_{i,j} \in \mathcal{E} \mid \mathbf{A}_{i,j} = 1; j \in \{1, \cdots, n_v\}\}$ and $\mathcal{M}(v_j) = \{e_{i,j} \in \mathcal{E} \mid \mathbf{A}_{i,j} = 1; i \in \{1, \cdots, n_u\}\}$.

As in many previous anomaly detection formulations [8], [9], [12], [31], we formalize anomaly detection as a ranking problem with a scoring function, where a larger score means a higher degree of abnormality.

Problem 1: Given a bipartite node-and-edge-attributed graph $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E}, \mathbf{X}^u, \mathbf{X}^v, \mathbf{X}^e)$*, our task is to produce*

scoring functions applicable to each edge $e \in \mathcal{E}$, as well as each node $u \in \mathcal{U}$ and $v \in \mathcal{V}$, such that the edges and nodes that differ singularly from the majority in terms both the structure and attribute information should be given higher anomaly score.

Specifically, as we have three different sets (edges and two node types), we produce three scoring functions: $\text{score}_e(\cdot)$, $\text{score}_u(\cdot)$, and $\text{score}_v(\cdot)$; each for scoring the edges, the nodes in $\mathcal{U}$, and the nodes in $\mathcal{V}$, respectively.

IV. Methodology

In this section, we will describe our method for addressing anomaly detection on bipartite node-and-edge-attributed graphs (Problem 1). The key part of our network architecture is in the design of the encoder and decoder to accommodate the input graph and to produce the desired anomaly scores. However, before going deeper into the architecture, we will describe the building blocks of our network, i.e., the convolution and message passing operations.

A. Graph Convolution and Message Passing

We first describe the notations for individual node and edge features and representations. Let $\mathbf{x}_i^u$ denotes the features for node u_i in $\mathcal{U}$ with $i \in \{1, \cdots, n_u\}$, $\mathbf{x}_j^v$ denotes the features for node v_j in $\mathcal{V}$ with $i \in \{1, \cdots, n_v\}$, whereas $\mathbf{x}_{i,j}^e$ denotes the features for edge $e_{i,j}$ in $\mathcal{E}$. Our network consists of K (even number) convolution layers, where we use half of them for the encoder and another half for the feature decoder. We use the lowercase k to enumerate the convolution layer. To represent the intermediate node/edge representations of layer k, we use lowercase $\mathbf{h}$ and add $\{\cdot\}^{(k)}$ super-script to the notation. Specifically, $\{\mathbf{h}_i^u\}^{(k)}$, $\{\mathbf{h}_j^v\}^{(k)}$, and $\{\mathbf{h}_{i,j}^e\}^{(k)}$ denote the k-th layer representation for node u_i in $\mathcal{U}$, node v_j in $\mathcal{V}$, and edge $e_{i,j}$ in $\mathcal{E}$, respectively.

Our convolution operation takes the previous node representations of each node in $\mathcal{U}$ and $\mathcal{V}$, i.e.: $\{\mathbf{h}_i^u\}^{(k-1)}$ and $\{\mathbf{h}_j^v\}^{(k-1)}$, as well as the previous edge representations, i.e.: $\{\mathbf{h}_{i,j}^e\}^{(k-1)}$. It then performs message passing and outputs new node and edge representations: $\{\mathbf{h}_i^u\}^{(k)}$, $\{\mathbf{h}_j^v\}^{(k)}$, and $\{\mathbf{h}_{i,j}^e\}^{(k)}$. The design of our convolution operation is influenced by GraphSAGE [28], where instead of training distinct representation for each node, it trains a set of *aggregator functions* that learns to aggregate feature information from a node's local neighborhood. Certainly, several modifications need to be made, as we need to produce two types of node representations (for $\mathcal{U}$ and $\mathcal{V}$), as well as edge representations. In addition, the convolution operation also needs to incorporate edge features in addition to the node features.

The compute the new representations, $\{\mathbf{h}_i^u\}^{(k)}$, $\{\mathbf{h}_j^v\}^{(k)}$, and $\{\mathbf{h}_{i,j}^e\}^{(k)}$, we first collect the messages passed to the particular node/edge. The messages to a node u_i in $\mathcal{U}$ come from the aggregator functions over the neighboring node representations, the aggregator functions over the edge representations connected to u_i, and its own previous representations. The messages to node v_i in $\mathcal{V}$ follow similarly. We also collect messages passed to an edge $e_{i,j}$ in $\mathcal{E}$, where it simply comes from the nodes connected to the edge, i.e., u_i and v_j, and its own previous representation. The equations below formally describe how the messages passed to u_i, v_j, and $e_{i,j}$ are computed as:

$$\text{Msg}[\rightarrow u_i]^{(k)} = \Big\{\text{Agg}\big\{\{\mathbf{h}_j^v\}^{(k-1)} \mid \forall v_j \in \mathcal{N}(u_i)\big\} \cup \text{Agg}\big\{\{\mathbf{h}_{i,j}^e\}^{(k-1)} \mid \forall e_{i,j} \in \mathcal{M}(u_i)\big\} \cup \{\mathbf{h}_i^u\}^{(k-1)}\Big\} \quad (1)$$

$$\text{Msg}[\rightarrow v_j]^{(k)} = \Big\{\text{Agg}\big\{\{\mathbf{h}_i^u\}^{(k-1)} \mid \forall u_i \in \mathcal{N}(v_j)\big\} \cup \text{Agg}\big\{\{\mathbf{h}_{i,j}^e\}^{(k-1)} \mid \forall e_{i,j} \in \mathcal{M}(v_j)\big\} \cup \{\mathbf{h}_j^v\}^{(k-1)}\Big\} \quad (2)$$

$$\text{Msg}[\rightarrow e_{i,j}]^{(k)} = \Big\{\{\mathbf{h}_i^u\}^{(k-1)} \cup \{\mathbf{h}_j^v\}^{(k-1)} \cup \{\mathbf{h}_{i,j}^e\}^{(k-1)}\Big\} \quad (3)$$

Here, $\cup$ represents the concatenation operator, where we concatenate messages that come from different sources. The aggregation function Agg could be a simple aggregation function like Mean and Max, a combination of both, or more complex aggregation functions (e.g., LSTM and Pooling). Figure 1a depicts the flow of message passing.

After collecting the messages for each node and edge, we compute the next representations. We pass the messages to a linear operation parameterized by $\mathbf{W}$ and $\mathbf{b}$. We normalize the output of the linear operation using batch normalization (BN) and pass it to an activation function as follows:

$$\{\mathbf{h}_i^u\}^{(k)} = \text{ReLU}\big(\text{BN}\big(\mathbf{W}_u^{(k)} \cdot \text{Msg}[\rightarrow u_i]^{(k)} + \mathbf{b}_u^{(k)}\big)\big) \quad (4)$$

$$\{\mathbf{h}_j^v\}^{(k)} = \text{ReLU}\big(\text{BN}\big(\mathbf{W}_v^{(k)} \cdot \text{Msg}[\rightarrow v_j]^{(k)} + \mathbf{b}_v^{(k)}\big)\big) \quad (5)$$

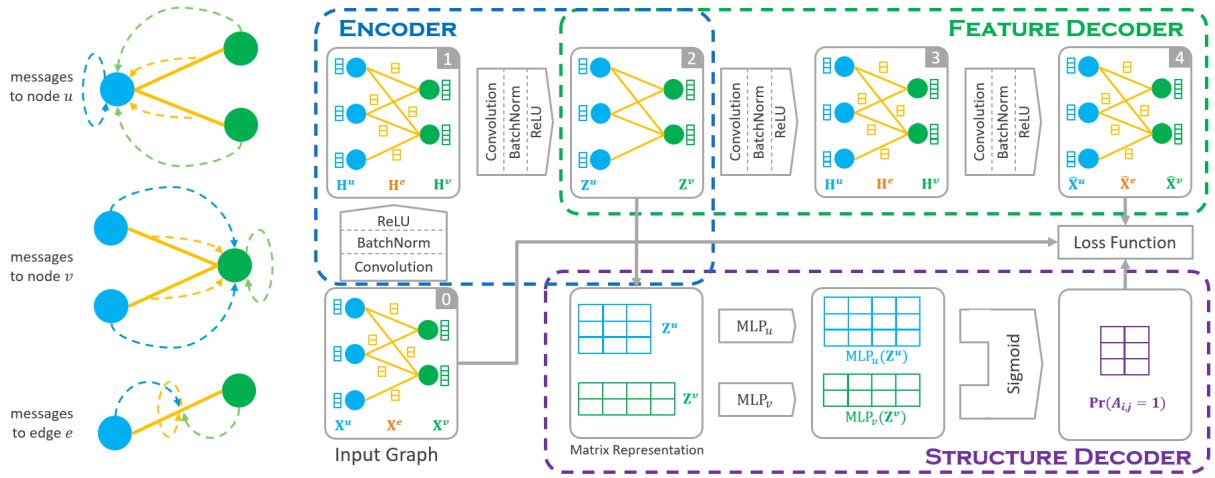
$$\{\mathbf{h}_{i,j}^e\}^{(k)} = \text{ReLU}\big(\text{BN}\big(\mathbf{W}_e^{(k)} \cdot \text{Msg}[\rightarrow e_{i,j}]^{(k)} + \mathbf{b}_e^{(k)}\big)\big) \quad (6)$$

The weight and bias parameters in the linear operation for each node-set are shared across all nodes in the set. Similarly, the parameters for the edge set are shared across all edges. Hence, in one convolution layer, we have six parameters: $\mathbf{W}^u$, $\mathbf{W}^v$, $\mathbf{W}^e$, $\mathbf{b}^u$, $\mathbf{b}^v$, and $\mathbf{b}^e$.

Note that several GraphSAGE extensions have been proposed to incorporate edge features in the convolution operation, such as PNA [35], EGNN [36], EGAT [29], etc. [30], [37]–[39]. However, most of the methods only use the edge feature as an additional input for computing the next node representations without generating new edge representations as required in our model. Our convolution and message passing scheme is most closely related to the work of You, et. al. [40], where they also generate new edge representation. The differences are in the parameterization and normalization used in the convolution layer, as well as the domain area of the use cases (handling missing data vs. anomaly detection). Our convolution and message passing scheme is designed specifically for our network architecture.

B. Network Architecture

We are now ready to present the GraphBEAN architecture for anomaly detection on bipartite node-and-edge-attributed graphs. GraphBEAN follows an autoencoder-like architecture, where a network is tasked to reconstruct the original input. The network is given a bottleneck to prevent the network from just



(a) Message passing flow

(b) GraphBEAN architecture: encoder, feature decoder, and structure decoder (4 layers)

Fig. 1. GraphBEAN architecture and message passing scheme

copying the input. For example, it has to produce a low dimensional latent variable before reconstructing the input [41].

In our architecture, the *encoder* takes a bipartite node-and-edge-attributed graph as the input and produces latent representations of the nodes only. However, the *decoder* is required to reconstruct the full original graph, i.e., the graph structure, node attributes edges attributes. The decoder is divided into two parts: the *feature decoder*, which reconstructs the node and edge features, and the *structure decoder*, which reconstructs the graph structure. This construction forces the latent representation of each node to also encode all neighboring edge features in addition to the local graphical structure and neighboring node features. Finally, we use the nodes and edges reconstruction error as the base for determining the anomaly score for each node and edge in the bipartite graph.

Even though several graph neural network architectures for anomaly detection have been proposed by previous works [8]–[14], [31]–[34], GraphBEAN architecture is distinct in several aspects. First, previous works can only detect node-level anomalies, whereas GraphBEAN detects both node-level and edge-level anomalies. Second, as GraphBEAN needs to accommodate edge representations, its message passing operation is designed differently from previous works, which do not need to handle edge representations. Third, the design of the latent representation in GraphBEAN is unique, as the node latent representations in the network need to encode all the neighboring (and its own) node features, node structures, as well as edge features that are connected to the node in K-hop. The node latent representations of the previous architectures do not need to encode the neighboring edge features. We now explain the detail of GraphBEAN architecture as follows.

1) Encoder: The encoder consists of $K/2$ graph convolution layers. We initialize the input to the first layer as the original input graph feature for each node and edge, i.e.:

$$\{\mathbf{h}_i^u\}^{(0)} = \mathbf{x}_i^u \quad \{\mathbf{h}_j^v\}^{(0)} = \mathbf{x}_j^v \quad \{\mathbf{h}_{i,j}^e\}^{(0)} = \mathbf{x}_{i,j}^e. \tag{7}$$

The subsequent layers take the output of the previous layer as their input. Every layer outputs new node and edge representations, $\{\mathbf{h}_i^u\}^{(k)}$, $\{\mathbf{h}_j^v\}^{(k)}$, and $\{\mathbf{h}_{i,j}^e\}^{(k)}$, except the last encoder layer ($K/2$-th layer) where it only outputs node representations $\{\mathbf{h}_i^u\}^{(K/2)}$ and $\{\mathbf{h}_j^v\}^{(K/2)}$. These node representations become the latent representation (denoted as $\mathbf{z}$) for each node, i.e.:

$$\mathbf{z}_i^u = \{\mathbf{h}_i^u\}^{(K/2)} \qquad \mathbf{z}_j^v = \{\mathbf{h}_j^v\}^{(K/2)}. \tag{8}$$

2) Feature Decoder: The feature decoder also consists of $K/2$ convolution layers. The first layer takes the latent representations of the nodes in $\mathcal{U}$ and $\mathcal{V}$ as the input without accepting edge representations. Therefore, the message collected in the message passing scheme is simplified to:

$$\text{Msg}[\rightarrow u_i]^{(\frac{K}{2}+1)} = \left\{\mathbf{z}_i^u \cup \text{Agg}\left\{\mathbf{z}_j^v \,\middle|\, \forall v_j \in \mathcal{N}(u_i)\right\}\right\} \tag{9}$$

$$\text{Msg}[\rightarrow v_j]^{(\frac{K}{2}+1)} = \left\{\mathbf{z}_j^v \cup \text{Agg}\left\{\mathbf{z}_i^u \,\middle|\, \forall u_i \in \mathcal{N}(v_j)\right\}\right\} \tag{10}$$

$$\text{Msg}[\rightarrow e_{i,j}]^{(\frac{K}{2}+1)} = \left\{\mathbf{z}_i^u \cup \mathbf{z}_j^v\right\}. \tag{11}$$

The remaining convolution layers follow the normal scheme as described previously. The output of the last layer becomes the reconstructed node and edge features:

$$\hat{\mathbf{x}}_i^u = \{\mathbf{h}_i^u\}^{(K)} \; \hat{\mathbf{x}}_j^v = \{\mathbf{h}_j^v\}^{(K)} \; \hat{\mathbf{x}}_{i,j}^e = \{\mathbf{h}_{i,j}^e\}^{(K)}, \tag{12}$$

which will be compared to the original input features.

3) Structure Decoder: In the feature decoder, the graph structure is given to the model as the basis for passing the messages, thus only providing limited information on the graph structure. Therefore, in order to enforce the latent variable's encoding of the graph structure, we use a structure decoder. From the latent representations $\mathbf{z}_i^u$ and $\mathbf{z}_j^v$, it predicts if node u_i in $\mathcal{U}$ is connected to node v_j in $\mathcal{V}$. It works by passing $\mathbf{z}_i^u$ to a $K/2$ layers of multi-layer perceptron (MLP_u) and passing $\mathbf{z}_j^v$ to MLP_v. It then uses the dot product of the output of each MLPs, and passes it to a sigmoid function to produce the probability of u_i connected to v_j, i.e.:

$$\Pr(\mathbf{A}_{i,j}=1) = \text{sigm}\left(\text{MLP}_u(\mathbf{z}_i^u)^\top \cdot \text{MLP}_v(\mathbf{z}_j^v)\right), \tag{13}$$

which will be compared to the adjacency matrix $\mathbf{A}$.

C. Model Training

As we have constructed our network architecture, in this section, we will describe the procedure to train the network.

1) Negative Sampling for Structure Decoder: The edge prediction formula in the structure decoder (Eq. 13) defines the probability for every pair of nodes in $\mathcal{U}$ and $\mathcal{V}$. In practice, the majority of node pairs are not connected. Therefore, to gain more efficiency, we can just compute the probability for a subset of the pairs without incurring any performance degradation. Let $\mathcal{E}^+$ denote the set of pairs where u_i and v_j are connected in the graph, i.e. $\mathcal{E}^+ = \{(i,j) \mid A_{i,j} = 1\}$, whereas $\mathcal{E}^-$ denotes the set of pairs that are not connected, i.e. $\mathcal{E}^- = \{(i,j) \mid A_{i,j} = 0\}$. We define the set $\mathcal{E}^\pm$ as the set of all indices $\mathcal{E}^\pm = \mathcal{E}^+ \cup \mathcal{E}^-$. In our training procedure, we do not use all items in $\mathcal{E}^-$. Rather, we only sample a small portion of $\mathcal{E}^-$ and combine it with $\mathcal{E}^+$ to get the set of pairs used in the training $\widetilde{\mathcal{E}}^\pm$, i.e., $\widetilde{\mathcal{E}}^\pm = \mathcal{E}^+ \cup \text{Sample}(\mathcal{E}^-)$. We use the standard uniform random sampling of the negative pairs $\mathcal{E}^-$ with a pre-specified number of samples, e.g., a small multiple of the number of positive pairs ($\mathcal{E}^+$).

2) Loss Function: Similar to other autoencoder-like architectures, the loss function we use is a type of reconstruction loss. Let $\hat{\mathbf{X}}^u$, $\hat{\mathbf{X}}^v$, and $\hat{\mathbf{X}}^e$ be the matrices that collect the network's output for all nodes in $\mathcal{U}$, all nodes in $\mathcal{V}$, and all edges, respectively. Our training objective function is the combination of the mean squared error (MSE) of node $\mathcal{U}$ feature reconstruction, the MSE of node $\mathcal{V}$ feature reconstruction, the MSE of edge feature reconstruction, and the binary cross entropy (BCE) of the structure decoder's edge prediction. The following is the formulation of our loss function ($\mathcal{L}$):

$$\mathcal{L} = \text{MSE}(\mathbf{X}^u, \hat{\mathbf{X}}^u) + \text{MSE}(\mathbf{X}^v, \hat{\mathbf{X}}^v) + \text{MSE}(\mathbf{X}^e, \hat{\mathbf{X}}^e) + \eta \text{BCE}(\widetilde{\mathcal{E}}^\pm, \mathbf{A}), \quad \text{where:} \tag{14}$$

$$\text{MSE}(\mathbf{X}, \hat{\mathbf{X}}) = \frac{1}{nm} \sum_{i=1}^{n} \sum_{l=1}^{m} \left(\mathbf{X}_{i,l} - \hat{\mathbf{X}}_{i,l}\right)^2, \text{ and} \tag{15}$$

$$\text{BCE}(\widetilde{\mathcal{E}}^\pm, \mathbf{A}) = -\frac{1}{|\widetilde{\mathcal{E}}^\pm|} \sum_{(i,j)\in\widetilde{\mathcal{E}}^\pm} \big\{\mathbf{A}_{i,j} \log(\Pr(\mathbf{A}_{i,j} = 1)) + (1 - \mathbf{A}_{i,j}) \log(1 - \Pr(\mathbf{A}_{i,j} = 1))\big\}, \tag{16}$$

with η denotes a constant for balancing the feature decoder's MSE and the structure decoder's BCE.

3) Forward and Backward Propagation: Now, we have all the components of our GraphBEAN model. In the training process, we first perform forward propagation by feeding the input data to the encoder to produce the latent representations, then pass them to both feature and structure decoders to produce the reconstructed data. We then use the loss function to compute the objective of our optimization. Algorithm 1 provides the detailed step-by-step process of performing forward propagation in GraphBEAN, covering all the network components we discussed above. Figure 1b also provides an illustration of the process. For computing backward propagation (gradient) and performing update to the model, we resort to a ML framework.

D. Anomaly Score

After we finish the training, we produce anomaly scoring functions for the edges as well as the nodes in $\mathcal{U}$ and $\mathcal{V}$. Given an input graph, the scoring functions are computed by first performing a forward propagation to the encoder and decoders and then computing individual node/edge reconstruction error. As in other autoencoder-based models, since the model optimizes a reconstruction loss, normal interactions which commonly occur in the graph can be easily reconstructed. On the other hand, anomalous interactions, which rarely occur, suffer from high reconstruction errors. We define the edge scoring functions as the weighted combination of the reconstruction error of the edge features and the BCE error of predicting if the edge should exist in the graph, i.e.:

$$\text{score}_e(e_{i,j}) = \text{MSE}(\mathbf{x}^e_{i,j}, \hat{\mathbf{x}}^e_{i,j}) + \eta \text{BCE}(e_{i,j}). \tag{17}$$

We then define the scoring function for the nodes in $\mathcal{U}$ and $\mathcal{V}$. The anomaly score of a node is the combination of the reconstruction error of its node feature and aggregation of the anomaly score of all edges connected to the node, i.e.:

$$\text{score}_u(u_i) = \text{MSE}(\mathbf{x}^u_i, \hat{\mathbf{x}}^u_i) + \underset{e_{i,j}\in\mathcal{M}(u_i)}{\text{Agg}} \text{score}_e(e_{i,j}) \tag{18}$$

Algorithm 1 GraphBEAN forward propagation

Require: graph $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E}, \mathbf{X}^u, \mathbf{X}^v, \mathbf{X}^e)$; even number of layers K; parameters: $\mathbf{W}_u^{(k)}, \mathbf{W}_v^{(k)}, \mathbf{W}_e^{(k)}, \mathbf{b}_u^{(k)}, \mathbf{b}_v^{(k)}$, and $\mathbf{b}_e^{(k)}$; structure decoder's MLPs: MLP_u and MLP_v

Ensure: loss/objective value

▷ *initial representations* ◁
$\{\mathbf{h}_i^u\}^{(0)} \leftarrow \mathbf{x}_i^u,\ \forall i \in \{1, \cdots, n_u\}$
$\{\mathbf{h}_i^v\}^{(0)} \leftarrow \mathbf{x}_j^v,\ \forall j \in \{1, \cdots, n_v\}$
$\{\mathbf{h}_{i,j}^e\}^{(0)} \leftarrow \mathbf{x}_{i,j}^e,\ \forall e_{i,j} \in \mathcal{E}$
▷ *encoder* ◁
for $k = 1, \ldots, K/2$ **do**
  compute messages: $\text{Msg}[\rightarrow u_i]^{(k)}$ and $\text{Msg}[\rightarrow v_j]^{(k)}$, for all nodes in $\mathcal{U}$ and $\mathcal{V}$ using Eq. (1) and Eq. (2)
  compute all node representations: $\{\mathbf{h}_i^u\}^{(k)}$ and $\{\mathbf{h}_j^v\}^{(k)}$, using Eq. (4) and Eq. (5)
  if $k \neq K/2$ **then**
    compute $\text{Msg}[\rightarrow e_{i,j}]^{(k)}$ for all edges using Eq. (3)
    compute edge representations: $\{\mathbf{h}_{i,j}^e\}^{(k)}$ using Eq. (6)
▷ *latent representations* ◁
$\mathbf{z}_i^u \leftarrow \{\mathbf{h}_i^u\}^{(K/2)},\ \forall i \in \{1, \cdots, n_u\}$
$\mathbf{z}_j^v \leftarrow \{\mathbf{h}_j^v\}^{(K/2)},\ \forall j \in \{1, \cdots, n_v\}$
▷ *feature decoder* ◁
for $k \leftarrow K/2 + 1, \cdots, K$ **do**
  if $k = K/2 + 1$ **then**
    compute $\text{Msg}[\rightarrow u_i]^{(k)}$, $\text{Msg}[\rightarrow v_j]^{(k)}$, and $\text{Msg}[\rightarrow e_{i,j}]^{(k)}$ Eq. (9), Eq. (10), and Eq. (11)
  else
    compute $\text{Msg}[\rightarrow u_i]^{(k)}$, $\text{Msg}[\rightarrow v_j]^{(k)}$, and $\text{Msg}[\rightarrow e_{i,j}]^{(k)}$ Eq. (1), Eq. (2), and Eq. (3)
  compute all representations: $\{\mathbf{h}_i^u\}^{(k)}$, $\{\mathbf{h}_j^v\}^{(k)}$ and $\{\mathbf{h}_{i,j}^e\}^{(k)}$, using Eq. (4), Eq. (5), and Eq. (6)
▷ *reconstructed features* ◁
$\hat{\mathbf{x}}_i^u \leftarrow \{\mathbf{h}_i^u\}^{(K)},\ \forall i \in \{1, \cdots, n_u\}$
$\hat{\mathbf{x}}_j^v \leftarrow \{\mathbf{h}_j^v\}^{(K)},\ \forall j \in \{1, \cdots, n_v\}$
$\hat{\mathbf{x}}_{i,j}^e \leftarrow \{\mathbf{h}_{i,j}^e\}^{(K)},\ \forall e_{i,j} \in \mathcal{E}$
▷ *structure decoder* ◁
perform negative sampling, combine it with $\mathcal{E}$ as $\widetilde{\mathcal{E}}^\pm$
compute edge probability $\Pr(\mathbf{A}_{i,j} = 1), \forall (i,j) \in \widetilde{\mathcal{E}}^\pm$
▷ *loss/objective value* ◁
$\ell \leftarrow$ compute loss using Eq. (14)
return ℓ

TABLE I
DATASET PROPERTIES

Dataset	#node $\mathcal{U}$	#node $\mathcal{V}$	#edge	$\mathcal{U}$ deg.	$\mathcal{V}$ deg.	#ft. $\mathcal{U}$	#ft. $\mathcal{V}$	#ft. $\mathcal{E}$	+ratio $\mathcal{U}$	+ratio $\mathcal{V}$	+ratio $\mathcal{E}$
FINEFOODS-SMALL	9,705	4,879	18,523	1.9	3.8	11	11	384	0.013	0.026	0.037
MOVIES-SMALL	9,622	6,366	28,147	2.9	4.4	11	11	384	0.013	0.019	0.020
WIKIPEDIA	8,227	1,000	18,257	2.2	18.3	174	174	345	0.023	0.071	0.025
REDDIT	10,000	984	78,516	7.9	79.8	174	174	345	0.038	0.102	0.012
FINEFOODS-LARGE	256,059	74,258	560,804	2.2	7.6	11	11	384	0.004	0.014	0.013
MOVIES-LARGE	889,176	253,059	7,834,236	8.8	31.0	11	11	384	0.001	0.005	0.002

$$\text{score}_v(v_j) = \text{MSE}(\mathbf{x}_j^v, \hat{\mathbf{x}}_j^v) + \underset{e_{i,j} \in \mathcal{M}(v_j)}{\text{Agg}} \text{score}_e(e_{i,j}). \quad (19)$$

The aggregation function AGG could be MEAN or MAX, depending on the need of the applications. The rationale of the aggregation is that since the entities (user/item) are the actors of the interactions, an anomaly in the interactions should affect the anomaly score of the entities involved.

E. Neighborhood Sampling and Inductive Learning

So far, our training procedure requires access to the full graph structure as well as all node and edge features. This full graph training is, however, not scalable to large-size graphs. To be able to scale the learning algorithm to large graph data, we have to perform stochastic training via minibatching and neighborhood sampling. Previous scalable graph models, like GraphSAGE, have proposed neighborhood sampling algorithms. However, due to our custom convolution operation and network architecture, we have to adjust the flow of the messages from one layer to the next, particularly message flows to the edges and bipartite nodes. Specifically, from the node-set used for minibatching (either $\mathcal{U}$ or $\mathcal{V}$), after we get a batch of nodes, we sample nodes from the other node-set and the edge connecting them, which we then set as the target sub-graph. We then repeatedly expand this sub-graph K times by sampling its neighboring nodes and edges, while constructing message flow graphs from one layer to the next. After K times expansion, we will get the input sub-graph to the first layer of GraphBEAN. This minibatching and neighborhood sampling step enables GraphBEAN to run on large-scale graphs.

We have also only discussed anomaly detection in a *transductive* setting, where the model provides reasoning only on the observed data. The *inductive* learning setting requires the model to come up with a general principle so that it can be applied to newly observed nodes/edges/(sub)-graphs. Fortunately, unlike many previous GNN anomaly models, the GraphBEAN inductive capability comes for free. Its architectures, particularly the convolution operation, message passing and aggregation, as well as neighborhood sampling, are built on top of the GraphSAGE architecture, a graph neural network model famously known for its inductive capability.

V. EXPERIMENTS

To evaluate our approach, we apply GraphBEAN to detect anomalies in several real-world public datasets.

A. Experiment Setup

1) Datasets: We use publicly available datasets that have been used in previous research. The WIKIPEDIA dataset [42] describes the editing activity of Wikipedia contributors to wiki pages, The REDDIT dataset [43] includes the posting activity of Reddit users to subreddits. In both datasets, the edge features are created by taking the mean and max aggregate over temporal action features constructed by Kumar et al. [44], plus some temporal statistics. Similarly, the node attributes contain statistical variables and a mean aggregate over the action features. Two other datasets, FINEFOODS and MOVIES [45] describe reviewing activities of Amazon users to products sold in Amazon website's fine-foods and movies categories, respectively. The node features contain statistics and summaries of users' reviewing activities and product reviews. The edge features contain the embedding of the review text. We process the original reviews using the SentenceBERT language model [46] to produce sentence embeddings. Since the size of FINEFOODS and MOVIES is quite large, we create two versions of the datasets. The small datasets, FINEFOODS-SMALL and MOVIES-SMALL, where we sample a subgraph containing around ten thousand nodes, and the large datasets, FINEFOODS-LARGE and MOVIES-LARGE, containing the full-size graphs. The properties of the six datasets above are shown in Table I. The table describes the dateset names (first column), the number of nodes in node-set $\mathcal{U}$ and $\mathcal{V}$ as well as the number of edges in each dataset (next three columns). It also includes the average degree of $\mathcal{U}$ and $\mathcal{V}$ nodes, i.e., the average number of edges that are connected to the nodes (next two columns), as well as the number of node and edge features (next three columns).

2) Anomaly Injection: We inject anomalies into the attributed graphs, as there are no ground truth anomalies in the datasets. The anomaly injection techniques are also common in previous anomaly detection papers [6], [8], [9], [31], [47], [48]. Specifically, we follow the anomaly injection technique described in Ding et al. [8], with some modifications to accommodate the bipartite structure and edge features as well as to simulate real-world anomalies more realistically. For the graph datasets, we inject two anomaly components, topological structure and attribute anomalies. To inject topological anomaly, we randomly select a small number of nodes in $U \in \mathcal{U}$ and $V \in \mathcal{V}$. We then create a fully dense block by making all nodes in U connected with an edge to all nodes in V, or a partially dense block, by randomly choosing a fraction

of pairs of nodes in (U, V) to be connected via an edge. The intuition is that in real-world scenarios, a small dense structure in a graph is a typical anomalous substructure in which the interactions are much more intense than average [6], [8].

For injecting attributes anomaly, we use two different techniques. The outside of a confidence interval technique [49], [50] injects a sample where a fraction of the features are replaced randomly from a truncated Gaussian distribution where the density of $x \in \hat{\mu} \pm c\hat{\sigma}$ is set to zero. Here, $\hat{\mu}$ and $\hat{\sigma}$ are the mean and standard deviation of the data, whereas c is the confidence interval constant. The scaled Gaussian noise technique [49], [51] selects one of the samples and augments it with a noise drawn from a Gaussian distribution with zero mean and $c\hat{\sigma}$ standard deviation, where c is the scaling constant.

We augment the dataset with anomalies by injecting a combination of topological structure and feature anomalies multiple times. In each one, we first inject structure anomaly with the number of nodes involved in the anomaly randomized between 2 and 20 in each node-set. We also randomize between performing full or partial dense block anomaly. If we perform a partial dense block anomaly, we randomize the fraction of the connected node pairs between 0.2 and 1.0. We then inject feature anomaly to the features of the edge we just constructed by selecting either "outside of a confidence interval" or "scaled Gaussian noise" technique, with c set randomly between 2 and 4. We also optionally inject feature anomalies into the node features. This anomaly construction provides a randomly weighted combination of structure and feature anomalies. We repeatedly inject the small block of anomalies 20 times (100 times for FINEFOODS-LARGE and MOVIES-LARGE) to simulate multiple anomaly occurrences in the dataset, each with its own characteristics. All the edges and the nodes involved in the injected anomalies are labeled as anomalous. The ratio of positive cases (anomalous) compared to all cases can be seen in the last three columns of Table I.

3) Baselines: We compare GraphBEAN with several anomaly detection techniques, both classical and deep learning models. From the classical models, we select FRAUDAR [6], a popular anomaly model for bipartite graphs, and Isolation Forest [52], a tree-based anomaly detection widely used in industry. From the pool of previous GNN anomaly models, we select the DOMINANT [8], as it is the first GNN model on anomaly detection; its direct extension AnomalyDAE [9]; and two other more recent GNN models. Note that, like many other GNN models, they work on homogeneous node-attributed graphs. The complete list of the baselines are:

1) FRAUDAR [6]. It detects dense block anomalies even with camouflage in bipartite graphs.
2) Isolation Forest [52]. It is a classical tree-based feature-only anomaly detection model.
3) DOMINANT [8]. It is a deep learning model for anomaly detection on homogeneous node-attributed graphs.
4) AnomalyDAE [9]. A dual autoencoders model for homogeneous node-attributed graphs.
5) CONAD [13]. It uses data augmentation and contrastive loss to detect anomalies in the graphs. We use the unsupervised version presented in [53].
6) AdONE [10]. An outlier-resistant embedding construction that can also be used to detect anomalies.

Since the baseline methods are designed to solve a slightly different problem, we make a few modifications to solve Problem 1. FRAUDAR only uses graph structure as the input. We use the deletion order of the edge and node for constructing the anomaly score for the node and edges, such that the later a node/edge is deleted, the higher its anomaly score. Isolation Forest uses features only. In each run, we create three separate models for node $\mathcal{U}$ features, node $\mathcal{V}$ features, and edge features. For DOMINANT, AnomalyDAE, CONAD, and AdONE, we treat the bipartite graphs as homogeneous graphs by removing the node type information and aggregating the edge features in the original graph into node features. We then run the model to produce anomaly scores for each node. The anomaly scores for each edge are produced by taking the average anomaly score of the two nodes connected to it.

B. Implementation

We implement our method, including the convolution operation, message passing, and neighborhood sampling, on top of PyTorch and PyTorch Geometric [54] frameworks. For the FRAUDAR, we use the author's implementation [6], whereas, for the Isolation Forest, we use Scikit-Learn implementation. For the deep learning baselines (DOMINANT, AnomalyDAE, CONAD, and AdONE), we use PyGOD [53] implementation of the algorithms. The experiments are conducted in a single Ubuntu Linux machine from the AWS Cloud Service with 8 CPU cores, 60GB of RAM, and an NVIDIA Tesla K-80 GPU, except for MOVIES-LARGE where we use a machine with larger RAM (120GB).

C. Experiment Setup

For each dataset, we run the experiment 10 times, except for the FINEFOODS-LARGE and MOVIES-LARGE where we run it for 5 and 1 times, respectively, due to their dataset size. In each run, we rerun the anomaly injection procedure so that each run has a different set of anomalies. We keep the set of anomalies to be the same for our method and all the baselines for the comparability of the results. In the experiment, we set the $\eta = 0.2$ for our method. For DOMINANT and AnomalyDAE, we set the parameter that balances the feature and structure decoder to be 0.8, as it also puts 0.2 weight on the structure decoder. For computing node anomaly scores, we use the Mean aggregation for WIKIPEDIA and REDDIT due to their higher average degree of $\mathcal{V}$ nodes. We use the Max aggregation for other datasets. The training procedures in FINEFOODS-SMALL, MOVIES-SMALL, WIKIPEDIA, and REDDIT experiments are conducted in a transductive setting. In FINEFOODS-LARGE and MOVIES-LARGE experiments, the training process is done in an inductive fashion. We sample the sub-subgraph in each training epoch with a maximum of 10 neighbors for each node. However, in the evaluation, we use the full graph data without sampling.

TABLE II
THE MEAN AND (IN PARENTHESIS) STANDARD DEVIATION OF THE AUC-PR METRICS OVER MULTIPLE EXPERIMENT RUNS IN SMALL DATASETS. BOLD NUMBERS INDICATE THE BEST OR NOT-SIGNIFICANTLY-WORSE-THAN-THE-BEST RESULT (WILCOXON SIGNED-RANK TEST, $\alpha = 0.05$).

Dataset / Model	FINEFOODS-SMALL			MOVIES-SMALL			WIKIPEDIA			REDDIT		
	$\mathcal{U}$	$\mathcal{V}$	$\mathcal{E}$	$\mathcal{U}$	$\mathcal{V}$	$\mathcal{E}$	$\mathcal{U}$	$\mathcal{V}$	$\mathcal{E}$	$\mathcal{U}$	$\mathcal{V}$	$\mathcal{E}$
FRAUDAR	0.256 (0.07)	0.392 (0.07)	0.279 (0.13)	0.229 (0.12)	0.188 (0.11)	0.260 (0.16)	0.102 (0.03)	0.085 (0.09)	0.043 (0.04)	0.059 (0.02)	0.101 (0.01)	0.011 (0.007)
IsoForest	0.090 (0.02)	0.166 (0.04)	0.794 (0.12)	0.127 (0.05)	0.181 (0.08)	0.827 (0.10)	0.226 (0.06)	0.499 (0.12)	0.278 (0.10)	0.361 (0.08)	**0.608** (0.07)	0.172 (0.039)
DOMINANT	0.735 (0.10)	0.721 (0.10)	0.686 (0.12)	0.631 (0.09)	0.708 (0.08)	0.389 (0.16)	0.164 (0.03)	0.179 (0.04)	0.049 (0.02)	0.121 (0.02)	0.186 (0.01)	0.016 (0.003)
AnomalyDAE	0.770 (0.09)	0.773 (0.09)	0.683 (0.12)	0.679 (0.12)	0.753 (0.10)	0.556 (0.10)	0.174 (0.03)	0.193 (0.04)	0.051 (0.02)	0.128 (0.02)	0.192 (0.02)	0.015 (0.003)
CONAD	0.740 (0.10)	0.721 (0.10)	0.691 (0.12)	0.684 (0.09)	0.695 (0.08)	0.564 (0.09)	0.165 (0.03)	0.182 (0.04)	0.052 (0.05)	0.116 (0.02)	0.180 (0.18)	0.016 (0.003)
AdOne	0.239 (0.05)	0.162 (0.03)	0.048 (0.01)	0.164 (0.03)	0.129 (0.03)	0.021 (0.01)	0.205 (0.04)	0.128 (0.03)	0.025 (0.01)	0.138 (0.02)	0.133 (0.01)	0.008 (0.001)
GraphBEAN (ours)	**0.855** (0.08)	**0.875** (0.07)	**0.876** (0.09)	**0.911** (0.04)	**0.911** (0.04)	**0.888** (0.08)	**0.441** (0.09)	**0.571** (0.03)	**0.415** (0.11)	**0.427** (0.06)	**0.631** (0.04)	**0.296** (0.038)

In all of our experiments, we use GraphBEAN with 4 layers, where we split it into 2 convolution layers for the encoder, and 2 convolution layers for the feature decoder. The MLPs in the structure decoder also comprise 2 dense layers. Similarly, we also use 4 layers for the GNN-based baselines (DOMINANT, AnomalyDAE, CONAD, and AdOne). For training GraphBEAN, we use the Adam optimizer with a learning rate of 0.01. The PyGOD implementation of the GNN-based baselines also uses the Adam optimizer. Similarly, we also set the learning rate to be 0.01. This learning rate is decided after running a few experiments using different learning rates (i.e., 0.0001, 0.001, 0.003, 0.01, 0.03), and observing that the 0.01 learning rate usually performs the best for all models. We train both GraphBEAN and other GNN models for 100 epochs. In all experiments, the dimension of latent variables is set to 32. For GraphBEAN experiments on large datasets, we use a batch size of 2048. For the Isolation Forest model, we use the default parameters in Scikit-Learn (e.g., the number of estimators, maximum sample in each estimator, bootstrap, etc.). In all experiments and all models, we set the random seed to 0. Finally, for the edge prediction in the structure decoder, we sample the negative cases (non-connected node pairs) as many as three times the number of edges (connected node pairs) in the graph (for full graph training) or in a batch (for stochastic training).

D. Experiment Results

1) Evaluation Metric: As different methods have different scoring systems, we adopt a ranking-based metric to evaluate the models. Specifically, we use the Area Under the Precision-Recall Curve (AUC-PR) as the evaluation metric. Compared with other alternatives, like the ROC curve, Precision-Recall curve provides a more informative picture of an algorithm's performance in the case of very imbalanced dataset as in anomaly detection, where the number of anomaly cases is far less than the number of normal cases (see [55], [56]).

TABLE III
THE MEAN AND (IN PARENTHESIS) STANDARD DEVIATION OF THE AUC-PR METRICS OVER MULTIPLE EXPERIMENT RUNS IN LARGE DATASETS (FINEFOODS-LARGE AND MOVIES-LARGE)

Dataset / Model	FINEFOODS-LARGE			MOVIES-LARGE		
	$\mathcal{U}$	$\mathcal{V}$	$\mathcal{E}$	$\mathcal{U}$	$\mathcal{V}$	$\mathcal{E}$
FRAUDAR	0.093 (0.02)	0.195 (0.03)	0.077 (0.02)	0.004	0.007	0.001
IsoForest	0.023 (0.00)	0.098 (0.02)	0.805 (0.05)	0.008	0.025	0.722
GraphBEAN	**0.701** (0.07)	**0.813** (0.05)	**0.875** (0.03)	**0.413**	**0.547**	**0.779**

In addition, practitioners oftentimes adjust the balance of the precision and recall of a deployed model to achieve the best business impact, making a threshold-free metric like AUC-PR suitable for anomaly detection evaluation.

2) Overall Results: The average and standard deviation of the AUC-PRs over multiple runs in each dataset are presented in Table II and Table III. The results are divided into three sections, the $\mathcal{U}$ nodes, the $\mathcal{V}$ nodes, and the edges $\mathcal{E}$ anomaly detections. For all datasets except MOVIES-LARGE, we perform hypothesis testing to see if the results are significantly different. We use the Wilcoxon signed-rank test with $\alpha = 0.05$. In the table, bold numbers indicate the best or not-significantly-worse-than-the-best result, based on the hypothesis testing.

As we can see from the tables, GraphBEAN significantly outperforms all baselines in all datasets for edge anomaly detection tasks, oftentimes by a large margin. For node anomaly detection tasks, GraphBEAN also significantly outperforms all models in nearly all datasets in both $\mathcal{U}$ and $\mathcal{V}$ nodes. Only one case in which one of the baselines (Isolation Forest) does not perform significantly worse than GraphBEAN, i.e.,

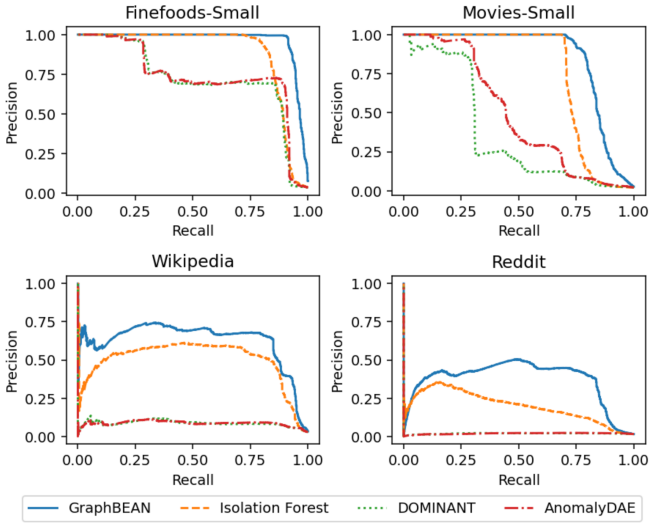


Fig. 2. Precision-recall curves of the edge anomaly detection tasks.

in Reddit's $\mathcal{V}$ node anomaly detection.[1] However, in this case, GraphBEAN still outperforms Isolation Forest. For Movies-Large datasets, even though we do not perform hypothesis testing, we can see that GraphBEAN outperforms the baselines by a considerably large margin.

As expected, FRAUDAR does not perform that well in our experiments, as it does not use the attributes in the detections. Isolation Forest provides good performance on edge anomaly but falls short on node anomaly as it treats node and edge features independently when the anomalous signal largely comes from the edges. However, its performances on edge anomaly are still significantly lower than GraphBEAN's in all datasets. The DOMINANT, AnomalyDAE, and CONAD perform reasonably well on node anomaly detection, as the node features contain aggregations of edge features. However, since the information available is just an aggregation of edge features, their performances do not match our model in detecting node anomalies. The DOMINANT, AnomalyDAE, and CONAD are also less effective in detecting edge anomaly as it is not the focus of the models. Surprisingly, AdONE performs significantly worse than DOMINANT, AnomalyDAE, CONAD, and our model. Due to the requirement of access to full adjacency matrix in their objective computation, DOMINANT, AnomalyDAE, and AdONE are not scalable to the size of FINEFOODS-LARGE and MOVIES-LARGE dataset. Therefore, we do not have their results for FINEFOODS-LARGE and MOVIES-LARGE.[2]

[1]Note that the base AUC-PR for a baseline (random) classifier is determined by the ratio of positives (P) and negatives (N) as P / (P + N) [56]. Therefore, our seemingly low value results for WIKIPEDIA and REDDIT are actually much better than the base AUC-PR, as the positive (anomalous) ratios in the datasets are very small, i.e., very imbalanced anomaly datasets (see Table I).

[2]PyGOD [53] provides unofficial neighborhood sampling implementations for DOMINANT, AnomalyDAE, and AdONE. We have tried to run the PyGOD implementations on FINEFOODS-LARGE and MOVIES-LARGE dataset with no success (memory error), since it requires the full adjacency to be stored in memory as a dense matrix. This requirement is not feasible for FINEFOODS-LARGE and MOVIES-LARGE due to their size.

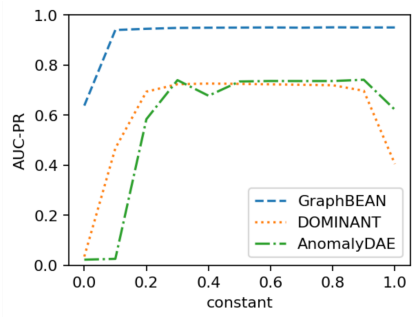


Fig. 3. The effect of balancing constant in FINEFOODS-SMALL

3) Precision-Recall Curve: The AUC-PR metric provides a good overall look at the prediction performance. However, practitioners may want to see the precision vs. recall trade-off at a given threshold. In practice, when it comes to the deployment of an anomaly detection system, practitioners need to select a threshold for the detection. The precision-recall curve provides a tool to perform trade-off analysis at multiple thresholding points. We present the precision-recall curve of the edge anomaly detection for a single experiment run on FINEFOODS-SMALL, MOVIES-SMALL, WIKIPEDIA, and REDDIT in Figure 2. We plot the precision-recall curves for GraphBEAN and the main baselines (Isolation Forest, DOMINANT, and AnomalyDAE). As we can see from the figure, at almost all thresholding points in all of those datasets, our method outperforms all the baselines, sometimes by a significant margin. The figure also shows that the benefit of our method can be seen more clearly on the thresholding points that matter to the practical use case, for example, the thresholding points that result in around 0.9 recall in FINEFOODS-SMALL, and around 0.8 recall in MOVIES-SMALL. This result confirms the superiority of our model for anomaly detection on bipartite node-end-edge-attributed graphs.

4) The effect of balancing constant: The objective function of DOMINANT, AnomalyDAE, and our method contains a parameter for balancing the feature and structure decoder losses. In this subsection, we study the effect of this constant on the performance of the model. Since our formulation is slightly different from both DOMINANT and AnomalyDAE, we adjust it accordingly. In our formulation, the weight of the feature decoder is always set as 1, whereas the weight of the structure decoder is η. In DOMINANT and AnomalyDAE, the weight of the feature decoder is set as α, and the weight of the structure decoder is $1 - \alpha$. Figure 3 shows the effect of this constant on the AUC-PR metric in one of the experiment runs for FINEFOODS-SMALL. We adjust our parameter to match DOMINANT and AnomalyDAE (α) in the figure. As we can see from the figure, the performance of all methods is relatively stable over the various value of α (from 0.2 to 0.9), except for the extreme values (closer to 0 or 1). The constant value that we used in the previous experiments ($\alpha = 0.8$) also produces one of the best-performing results in Figure 3. This adds validation to the hyperparameter choice in the main experiments.

VI. Conclusions

We proposed GraphBEAN for anomaly detection in bipartite node-and-edge-attributed graphs, which is capable of detecting node-level and edge-level anomalies. Our model is scalable, i.e., capable of running on large industrial-sized graphs. We demonstrated its performance benefits over the baselines in detecting anomalies in real-world datasets. For the future direction, we plan to extend our method to detect node and edge level anomalies in more general heterogeneous graphs.

References

[1] X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q. Z. Sheng, H. Xiong, and L. Akoglu, "A comprehensive survey on graph anomaly detection with deep learning," *IEEE TKDE*, 2021.

[2] W. Hilal, S. Gadsden, and J. Yawney, "Financial fraud: A review of anomaly detection techniques and recent advances," *Expert systems With applications*, 2022.

[3] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava, "A comparative study of anomaly detection schemes in network intrusion detection," in *SDM*, 2003.

[4] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[5] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *DMKD Journal*, 2015.

[6] B. Hooi, H. A. Song, A. Beutel, N. Shah, K. Shin, and C. Faloutsos, "Fraudar: Bounding graph fraud in the face of camouflage," in *ACM KDD*, 2016.

[7] H. Wang, C. Zhou, J. Wu, W. Dang, X. Zhu, and J. Wang, "Deep structure learning for fraud detection," in *ICDM*. IEEE, 2018.

[8] K. Ding, J. Li, R. Bhanushali, and H. Liu, "Deep anomaly detection on attributed net." in *SDM*, 2019.

[9] H. Fan, F. Zhang, and Z. Li, "Anomalydae: Dual autoencoder for anomaly detection on attributed networks," in *ICASSP*, 2020.

[10] S. Bandyopadhyay, S. V. Vivek, and M. Murty, "Outlier resistant unsupervised deep architectures for attributed network embedding," in *WSDM*, 2020.

[11] Z. Chen, B. Liu, M. Wang, P. Dai, J. Lv, and L. Bo, "Generative adversarial attributed network anomaly detection," in *CIKM*, 2020.

[12] M. Jin, Y. Liu, Y. Zheng, L. Chi, Y.-F. Li, and S. Pan, "Anemone: Graph anomaly detection with multi-scale contrastive learning," in *CIKM*, 2021.

[13] Z. Xu, X. Huang, Y. Zhao, Y. Dong, and J. Li, "Contrastive attributed network anomaly detection with data augmentation," in *PAKDD*, 2022.

[14] S. Yang, B. Zhang, S. Feng, Z. Tan, Q. Zheng, J. Zhou, and M. Luo, "Ahead: A triple attention based heterogeneous graph anomaly detection approach," *arXiv preprint*, 2022.

[15] Z. Chen and A. Sun, "Anomaly detection on dynamic bipartite graph with burstiness," in *ICDM*, 2020.

[16] T. Zhao, C. Deng, K. Yu, T. Jiang, D. Wang, and M. Jiang, "Gnn-based graph anomaly detection with graph anomaly loss," in *KDD*, 2020.

[17] W. Huang, Y. Li, Y. Fang, J. Fan, and H. Yang, "Biane: Bipartite attributed network embedding," in *ACM SIGIR*, 2020.

[18] J. Gao, F. Liang, W. Fan, C. Wang, Y. Sun, and J. Han, "On community outliers and their efficient detection in information networks," in *ACM KDD*, 2010.

[19] E. Müller, P. I. Sánchez, Y. Mülle, and K. Böhm, "Ranking outlier nodes in subspaces of attributed graphs," in *ICDEW*. IEEE, 2013.

[20] B. Perozzi, L. Akoglu, P. Iglesias Sánchez, and E. Müller, "Focused clustering and outlier detection in large attributed graphs," in *KDD*, 2014.

[21] C. C. Noble and D. J. Cook, "Graph-based anomaly detection," in *ACM KDD*, 2003.

[22] W. Eberle and L. Holder, "Discovering structural anomalies in graph-based data," in *ICDMW*, 2007.

[23] C. Liu, X. Yan, H. Yu, J. Han, and P. S. Yu, "Mining behavior graphs for "backtrace" of noncrashing bugs," in *SDM*. SIAM, 2005.

[24] N. Shah, A. Beutel, B. Hooi, L. Akoglu, S. Gunnemann, D. Makhija, M. Kumar, and C. Faloutsos, "Edgecentric: Anomaly detection in edge-attributed networks," in *ICDMW*. IEEE, 2016.

[25] J. Li, H. Dani, X. Hu, and H. Liu, "Radar: Residual analysis for anomaly detection in attributed networks." in *IJCAI*, 2017.

[26] Z. Peng, M. Luo, J. Li, H. Liu, and Q. Zheng, "Anomalous: A joint modeling approach for anomaly detection on attributed networks." in *IJCAI*, 2018.

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

[28] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *NIPS*, 2017.

[29] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.

[30] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. v. d. Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European Semantic Web Conference*, 2018.

[31] K. Ding, J. Li, N. Agarwal, and H. Liu, "Inductive anomaly detection on attributed net." in *IJCAI*, 2021.

[32] X. Yuan, N. Zhou, S. Yu, H. Huang, Z. Chen, and F. Xia, "Higher-order structure based anomaly detection on attributed networks," in *IEEE Big Data*, 2021.

[33] Y. Huang, L. Wang, F. Zhang, and X. Lin, "Are we really making much progress in unsupervised graph outlier detection?" *arXiv*, 2022.

[34] Q. Wang, G. Pang, M. Salehi, W. Buntine, and C. Leckie, "Cross-domain graph anomaly detection via anomaly-aware contrastive alignment," in *AAAI*, 2023.

[35] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković, "Principal neighbourhood aggregation for graph nets," in *NeurIPS*, 2020.

[36] L. Gong and Q. Cheng, "Exploiting edge features for graph neural networks," in *CVPR*, 2019.

[37] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. Bronstein, and J. Solomon, "Dynamic graph cnn for learning on point clouds," *ACM TOG*, 2019.

[38] B. Rozemberczki, P. Englert, A. Kapoor, M. Blais, and B. Perozzi, "Pathfinder discovery networks for neural message passing," in *WWW*, 2021.

[39] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec, "Strategies for pre-training graph neural networks," in *ICLR*, 2020.

[40] J. You, X. Ma, Y. Ding, M. J. Kochenderfer, and J. Leskovec, "Handling missing data with graph representation learning," *NeurIPS*, 2020.

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

[42] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Governance in social media: A case study of the wikipedia promotion process," in *AAAI ICWSM*, 2010.

[43] H. Lakkaraju, J. McAuley, and J. Leskovec, "What's in a name? understanding the interplay between titles, content, and communities in social media," in *AAAI ICWSM*, 2013.

[44] S. Kumar, X. Zhang, and J. Leskovec, "Predicting dynamic embedding trajectory in temporal interaction networks," in *KDD*, 2019.

[45] J. J. McAuley and J. Leskovec, "From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews," in *WWW*, 2013.

[46] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *EMNLP*. ACL, 2019.

[47] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang, "Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks," in *ACM KDD*, 2018.

[48] L. Zheng, Z. Li, J. Li, Z. Li, and J. Gao, "Addgraph: Anomaly detection in dynamic graph using attention-based temporal gcn." in *IJCAI*, 2019.

[49] G. Steinbuss and K. Böhm, "Generating artificial outliers in the absence of genuine ones—a survey," *ACM TKDD*, vol. 15, no. 2, pp. 1–37, 2021.

[50] T. S. Pham, Q. U. Nguyen, and X. H. Nguyen, "Generating artificial attack data for intrusion detection using machine learning," in *ACM SoICT*, 2014.

[51] H. Deng and R. Xu, "Model selection for anomaly detection in wireless ad hoc networks," in *CIDM*, 2007.

[52] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *ICDM*. IEEE, 2008.

[53] K. Liu, Y. Dou, Y. Zhao, X. Ding, X. Hu, R. Zhang, K. Ding, C. Chen, H. Peng, K. Shu, G. H. Chen, Z. Jia, and P. S. Yu, "PyGOD: A python library for graph outlier detection," *arXiv preprint*, 2022.

[54] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop*, 2019.

[55] J. Davis and M. Goadrich, "The relationship between Precision-Recall and ROC curves," in *ICML*, 2006.

[56] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets," *PloS ONE*, 2015.